UNIVERSITY OF CALIFORNIA,
IRVINE

Link Prediction with Deep Learning Models

THESIS

submitted in partial satisfaction of the requirements
for the degree of

MASTER OF SCIENCE

in Computer Engineering

by

Ahmet Salih Aksakal

Thesis Committee:
Associate Professor Mohammad Al Faruque, Chair
Assistant Professor Zhou Li
Professor Ozdal Boyraz

2019

DEDICATION

To my family and all my friends
for their constant support.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

I would like to thank my committee chair, Professor Mohammad Abdullah Al Faruque, for constantly guiding me and believing in me. Without his support, I would not have completed this thesis.

I would also like to thank my lab-mates Sujit Rokka Chhetri and Shih-Yuan Yu for their wonderful work with the KG embedding library: pykg2vec, and also for their intellectual support with my research.

I would also like to thank my committee members Professor Zhou Li and Professor Ozdal Boyraz, for providing me guidance and feedback.

ABSTRACT OF THE THESIS

Link Prediction with Deep Learning Models

By

Ahmet Salih Aksakal

Master of Science in Computer Engineering

University of California, Irvine, 2019

Associate Professor Mohammad Al Faruque, Chair

Deep Learning has been used extensively in many applications by researchers. With the increased attraction to Deep Learning, more and more unique models are created each year. However, sometimes some of the model details are not included in the publications. This makes using new Deep Learning models for research a time-consuming task for researchers. In order to tackle with this problem, we propose a prediction mechanism for the missing information in the model. By creating a dataset where the Deep Learning models are represented as knowledge graphs, we made it possible to use knowledge graph embedding algorithms which are specifically designed for eliminating missing information in a given data. We inspected 6 different algorithms and compared their performances on a small-scale experiment. After the comparison, we picked the most promising algorithm and used it for link prediction in Deep Learning models.

Chapter 1

Introduction

Deep Learning has been the center of attention for many of the research areas as a tool to solve complex problems. Some examples are image classification[36, 73] and pedestrian detection[9, 67] in computer vision area, smart manufacturing[37, 83] and system security[87, 29, 30] in cyber-physical systems, portfolio analysis[31, 40, 39] in finance, weather forecasting[43, 41, 70] in meteorology, and particle identification[13, 21, 51] in physics. As the number of people that use Deep Learning in their research increases, the number of unique Deep Learning models increases as well. The increased number of models brings some challenges for the researchers.

One such challenge the researchers encounter is to find a study with a specific Deep Learning model. Text-based search methods such as Google Scholar or Web of Science do not take the type of Deep Learning model used in the studies into consideration when searching. If the model has a unique name, and the published paper includes that as a key-word, only then they can find the requested studies. To find other results, researchers read the publications one by one and understand the type of model used in the study. Hence, it becomes a time-consuming task.

Yet another challenge arises for researchers when they want to use a Deep Learning model in their studies for a different application. This requires replication of the model. The first method for achieving this is to read the description of the model to understand the details and replicate it. However, the authors might not include the details of the model in the document. The second method is to find the codes written by the original authors for that model and tune it for a specific purpose. However, in some cases, the codes cannot be used directly because of a missing file or buggy code. Thus, to use the model correctly, the missing information in the model must be eliminated.

Overcoming these problems need the semantic analysis of the Deep Learning models. In other words, we must analyze the structure of the model and get the contextual information. This way, it becomes possible to compare different Deep Learning models or fill in the missing information in a given model based on its semantics. This analysis of the Deep Learning models has its challenges. They are listed below:

1) Representation of Models: Knowledge Graphs

To make a semantic analysis of Deep Learning models, we must analyze the components of the model; since without analyzing the architecture, it would not be possible to make a comparison or predict the missing information. The components of the Deep Learning models are layers, loss functions, and optimizers. The interactions of these components describe the Deep Learning model. Thus, to represent the models we need to capture the information about these components and store them in a concrete data structure.

In this manner, we keep the data without any missing information and the entire model is represented in a single format. In our previous studies[23, 81], we used knowledge graphs to represent engineering artifacts and did various learning tasks with them such as classification, inference and clustering. The results were promising and knowledge graph representation was proved to be powerful. In knowledge graphs, entities are connected to other entities in different ways. We can represent model components as entities and connect them in the

graph with different relations. For this reason, we picked knowledge graphs to represent the Deep Learning models. A more detailed background information for knowledge graphs can be found in Section 2.1.1.

2) Analysis Method: Machine Learning

To compare different models or predict the missing information the characteristics of the model must be extracted. This can be done with a rule-based approach or machine learning approach. In the rule-based method, a set of rules are defined to categorize the properties of the models. Then, they are compared to each other. In this manner, it is possible to make a prediction and comparison. For example, [90, 28, 58] used rule-based methods to make the inference task possible.

The machine learning method uses data to learn the rules that define the system. By analyzing many different samples, it captures the characteristics and adapts the parameters of the algorithm according to the information it gets from the data. Machine learning is measured to be more successful than the rule-based method in prediction and classification tasks[32]. Also in our case, defining rules for similarities of the entities is a difficult task, since there is no concrete definition of the similarity. For these reasons, we picked the machine learning approach for the semantic analysis of Deep Learning models. This brought another challenge: dataset.

3) Dataset: dL50a

We need graph representations of different Deep Learning models for the dataset we will use with our Machine Learning algorithms. To get these representations, we picked 50 different architectures based on the type of the Deep Learning model they use such as CNN, RNN, etc. We picked them in a way that the dataset includes at least one sample from each of the model types.

To convert them to knowledge graphs we inspected the codes written for 50 different models. We converted the codes to knowledge graph representations of the models. This high-level

representation of Deep Learning models eliminate the dependencies on coding languages and frameworks. We prepared the dataset manually to ensure the integrity and accuracy of the data. The details of this process are shown in Chapter 3, Section 3.2. The name of our dataset is dL50a, standing for "Deep Learning 50 Architectures".

4) Algorithm Selection: Knowledge Graph Embedding

Since we decided on using the Machine Learning approach, we need an algorithm capable of learning the architecture of a given model, then make predictions for the missing information if there is any. Link prediction for knowledge bases fits to this definition. It is the process of learning the characteristics of an entity in a graph and making connections that do not originally exist in the graph[78]. As an example, the link prediction task in a sample social network[56] is visualized in Figure 1.1. In the figure, we see the connections of different users in a social network. If two people are friends, they are connected with a solid black edge. The task is to predict the possible connections between people that are not currently connected, showed with red, dashed line. The same principle is applied to Deep Learning models in our case. For example, if we know a layer exists in the model but do not know where it is connected, we aim to predict its connection and other characteristics with link prediction.

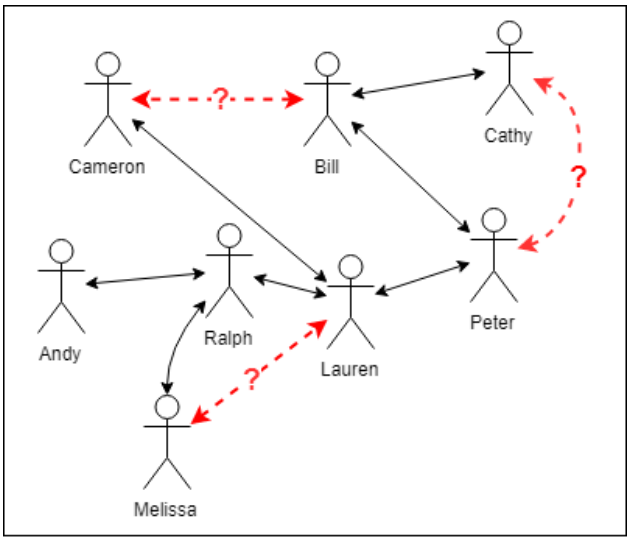

Figure 1.1: Link Prediction for Social Networks

There are various studies researching on the link prediction task with knowledge graphs[84]. The algorithms that make this possible are called Knowledge Graph Embedding algorithms. They encode the vast information of an entity to a multi-dimensional vector. In this manner, they embed the entities and relations to continuous vector space. This becomes useful when measuring the distances of any two entities. After the embedding, computing the distance is as simple as calculating the L1 distance between them, as is done in [18]. Knowledge graph embedding algorithms use these distance values to predict any missing link between entities. If two entities are embedded closer, the possibility of a link between them increases. This is the main idea of link prediction with embedding. Detailed background information about knowledge graph embedding algorithms can be found in the next chapter in Section 2.1.2.

Proposed Method

Our aim in this study is to eliminate the missing information in a given Deep Learning architecture through analysis of existing Deep Learning models. By using a Machine Learning approach, we aim to find the most probable candidates for the missing information. For this purpose, we proposed a three step process. The overall pipeline is visualized in Figure 1.2.

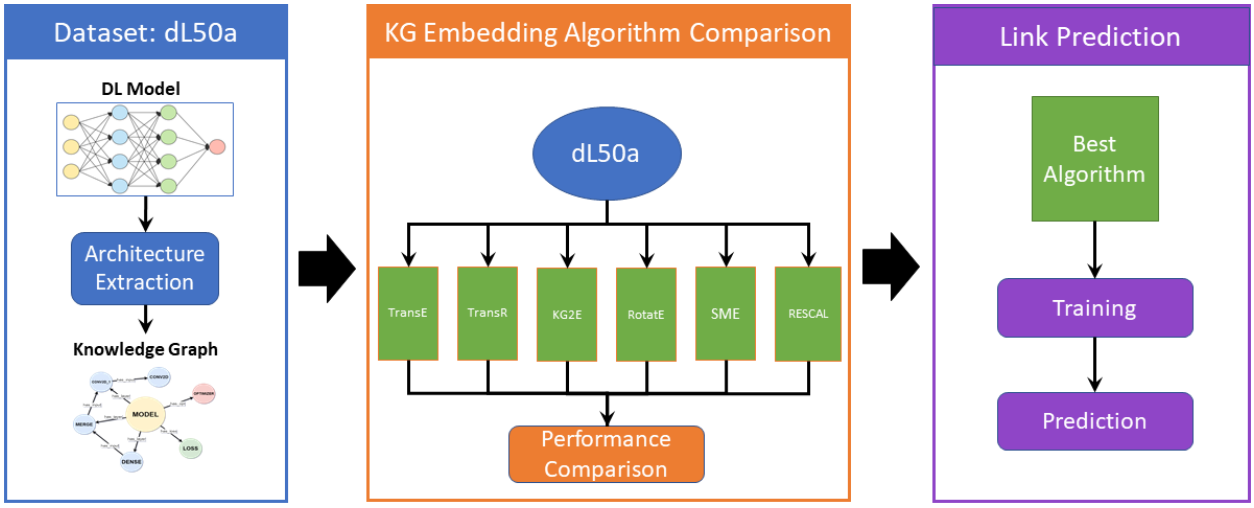

Figure 1.2: Pipeline of the Proposed Method

In the first step, we created a dataset with 50 different Deep Learning architectures. Each model is represented as a knowledge graph and combined together. The details of the first step are given in Chapter 3. In the second step, we picked 6 different knowledge graph embedding algorithms: TransE[18], TransR[57], KG2E[38], RotatE[76], SME(bilienar)[17] and RESCAL[64]. We used the algorithms with the dataset created in the first step and compared their performances in link prediction. The experimental procedure and its results are shown in Chapter 4. Finally in the third step, we selected the most promising algorithm in step 2. This algorithm is then trained and used for predicting the components of the models. The process is shown in Chapter 5.

Chapter 2

Background and Related Work

2.1 Background

2.1.1 Knowledge Graphs

We used knowledge graph as the data structure when representing the Deep Learning models in our dataset. Knowledge graph(KG) is a structure for keeping the relational information of different entities in the system. An entity is a component of the model such as layer, loss, activation etc. Each entity is defined through a set of relations with other entities.

The building block of the KG is a triple. Triples have a head, a relation and a tail. The head and the tail are the entities and the relation is the relationship they have. As an example, <Ankara, is_capital_of, Turkey> is a triple. In this triple, the information of Ankara being the capital of Turkey is stored.

A collection of triples form a KG. Different KGs can be formed to represent different knowledge. For example, there can be a KG to represent each city in a country to store the

information about their population, size, and other characteristics. Another example is a KG describing the components of a mechanical system. Since the data is kept as relations of different components, KGs can be used to define basically anything. In our work, we defined different Deep Learning models using KGs.

2.1.2 Knowledge Graph Embedding

KG Embedding task is defined as embedding the entities and the relations into a continuous vector space[57, 18, 86]. In this manner, each entity and each relation have a location in this multidimensional space. When doing the comparison of different entities, the embeddings are then used to calculate the distances of the entities for a quantitative analysis. In our study, we used KG embeddings to complete the missing information in a given Deep Learning model. This is done through comparing different values of embeddings in place of the missing information and predict the most probable candidate. This task is called Link Prediction[78].

There are various KG embedding algorithms for link prediction and they are mainly categorized with respect to the method they use [84]. There are two different main methods for KG embedding, translational distance and semantic matching models.

First method is the translational distance method. [18, 57, 38, 76] and many more algorithms use this method. In this technique, a scoring function based on the distances of the entities is used while training. The distances of the entities is calculated based on the translation of them using the relation. Basically, the relations of the entities are taken as basis to calculate the similarities of the entities.

The second method is the semantic matching models. It was used by [64, 17, 89, 74] and many other algorithms. In this technique, the entities are represented as vectors and the relations are represented as matrices. While training, they check the possibility of a given

triple by using the vector representations and the relation matrix. If the given triple makes sense(i.e. probable to exist or correct), the score will be higher, and if it does not, the score will be lower. In this manner, the weights of the vectors of the entities and the matrices of the relations will be updated. In the end, each relation and entity will have their vector embeddings. These embeddings are used to decide if a given entity is a good fit for the missing information or not.

2.2 Related Work

KG embedding has been widely used by the researchers for various applications. In [88], it is used to improve the academic paper search, [65] used it for improving the recommendation systems for sound and music, [7] used it in biological systems for different applications such as finding candidate genes of diseases or protein-protein interactions. In [44], they used a special form of KG in order to tackle with the software dependency problem. We used KG embeddings for inference of the missing information in Deep Learning models.

There are various studies that tackles code inference. However, these studies do not analyze Deep Learning models, but codes in general. They analyze the written codes for extraction of meaningful abstractions, summaries[82], coding conventions[3], repeated patterns[4] and algorithms[8] from the codes. [6, 5, 25, 14] worked on converting the code data structure to meaningful graph or sequence structure. These methods use various approaches to convert the source code to graph structures such as Abstract Syntax Trees (AST), data flow graphs, control flow graphs, Program Dependency Graph (PDG), etc.

Our Contribution

Note that Deep Learning models, can be analyzed through their components such as Layers, Loss, Optimizer etc. independent from their codes. With the abstraction over these compo-

nents, it is possible to make analysis without the limitations of the programming language, framework or libraries used for coding the model. The architectural information necessary to build a Deep Learning model is represented with these components. Their parameters and connections are the information that defines the model itself. In our study, we exploited this fact and aimed to make inference possible without depending on any coding style, library or framework.

Chapter 3

Deep Learning Dataset: dL50a

In this section, we have examined the characteristics of the dataset dL50a in depth. During the creation of dL50a, we used neural network layers to describe a given Deep Learning model. Layers are connected to each other sequentially and each of them has different characteristics. The characteristic properties of the Layers and the connection to each other creates the Deep Learning architecture. Thus, the unique characteristics of a model comes from the Layers. On top of the layers, a model also has unique loss and optimizer function definitions.

In the KG we created, each of the components used to define the model is represented as a class. We have the *Model* class representing the entire model, the *Layer* class describing a layer, the *Activation* class describing the activation function and so on. These classes are the fundamental building blocks of our dataset. Their relations with each other and characteristic properties are described in the following section.

3.1 Ontology

Before extracting the information from the deep learning models, a common vocabulary must be defined, otherwise, for different models there will be different type of definitions. The vocabulary contains the class types, relation types, and entity types. This vocabulary is used as a dictionary when interpreting data; and it is called the *Ontology*. A well-defined ontology results in better performance of the dataset.

While building the ontology we have used Protg[63], an ontology creation tool for knowledge graph databases. We created 5 different main classes namely, Model, Layer, Loss, Optimizer and Activation.

- Model

  This class is used to define the deep learning model. A Model contains many different Layers and one or more Loss and Optimizers. Each of the Model instances has a unique name; and the components of the model have this name as prefix in their name.

- Layer

  The fundamental building blocks of the architectures are the Layers. There are 21 different Layer types defined in our ontology. The types are defined as sub-classes to the main Layer class.

  Layers have sequential connection to other layers. They also might have an Activation class attached to them, representing the activation function for that Layer.

- Loss

  The class Loss is included in the Model to specify the loss function used to train the model. There are 9 types of Losses defined in the ontology.

- Optimizer

  Similar to the Loss class, Optimizer is included directly in the Model as well. The

purpose of this class is to show the type of the optimizer used while training the model. There are 5 different Optimizer types defined in the ontology.

The relations of different class instances are described using relational properties. One of the most important relational property is "has_input", which is used to describe the sequential connection of the layers. As an example <dense, has_input, flatten> describes the connection of the Flatten Layer to the Dense Layer. In this manner, we store the entire sequential structure as triples in the KG.

Other crucial information such as Layer types, Activation types, Loss and Optimizer types are described by object properties. Their function is to describe the relation of two different classes in the model. We have total of 8 different object properties, including the "has_input" property.

The characteristic properties of the Layers and other classes are described by data properties. The data properties give a quantitative description of a specific property for a class. One example of such data property is "has_shape". It shows the number of neurons in the Layer. We have 12 different data properties in total.

3.2 Capturing the Architectural Data

In order to capture essential architectural information from a given Deep Learning model, we inspected the codes and documentations written for it. For each model, we followed the step by step procedure as given below:

1. Find the Input of the Model

   Finding the input of the model is crucial, since it shows where the sequential structure starts. In addition to input point, in this step, we also obtained the input shape, which

shows the dimensions of the input data. This is also crucial information since all the remaining layers are shaped based on that.

2. Extract the Layer Information

   After obtaining the start point of the model, the next step is to get the information about the layers. We captured the sequential information for each of the layers and added it to the graph with "has_input" property. In addition to sequential structure, we obtained the characteristic information of the layers such as type, shape, kernel size, activation etc.

3. Loss and Optimizer Types

   The next step is to acquire the loss and optimizer type. Each deep learning model is defined with loss and optimizer functions. They are critical architectural information, since they change how the model behaves. The information includes the types of the loss and optimizer functions.

4. Shuffling the Data in the Dataset

   After completing the first three steps for every architecture we have, we shuffled the triples in the entire dataset. Shuffling the dataset allows for more homogeneous distribution of information, preventing problems such as information leakage and noise.

After getting the triples ready, we divided our entire data into 3 parts, training set(75% of the triples), test set(15% of the triples) and validation set(10% of the triples).

3.3 Included Papers

We have included 50 different deep learning architectures in our dataset. With 50 different models we aimed to include as many variations of Deep Learning applications as possible. Some of the authors included different variations of their model in their codes, such as a

smaller version for debugging or another similar model for comparison. We captured and added them to our dataset as well.

The papers we got the models from are listed in Table 3.1. The table also includes the number of triples we have extracted and the number of unique entities in each model. In total, there are 8021 triples describing the relations of 2705 unique entities in dL50a. The relations are described using 20 different relational properties.

Model	**Triples**	**Entities**	**Model**	**Triples**	**Entities**
Adapt_Seg[92]	164	51	GraphBit[27]	46	30
AGAN[59]	60	29	InfoGAN[22]	118	59
Alexnet[53]	86	45	License_Plate[61]	318	86
Anomaly[75]	24	17	NeuralStyle[47]	305	86
Ave_ECCV[79]	81	40	OCR[20]	117	43
BeGAN[15]	243	60	pix2pix[46]	298	88
Cabbage[77]	125	42	POSeidon[19]	212	63
CCSA[62]	83	39	Posewarp[12]	498	122
CNNSentence[49]	53	34	RCNN_Obj[55]	249	75
CNN_for_ReID[2]	91	37	Resnet50[35]	579	153
Colorful[91]	82	32	seq2seqVAE[33]	38	26
MCNN[93]	110	46	SeveralGAN[69]	85	41
CycleGAN[94]	154	53	SeveralGAN_1[69]	59	37
Deblur_GAN[54]	470	134	TCFPN_ISBA[26]	147	54
DenseNet[45]	71	35	TF_fashion_mnist[1]	17	15
DFI[80]	130	39	TrafficSigns[71]	118	44
DisAdvNET[34]	61	30	VGG16[73]	185	61
DisAdvNET_1[34]	33	19	VGG19[73]	144	46
DisAdvNET_2[34]	110	48	Visual_Redaction[66]	605	163
DiscoGAN[48]	106	50	VQA[10]	40	27
EGO[42]	27	23	WassersteinGAN[11]	132	51
Epinet[72]	321	76	WassersteinGAN_1[11]	34	19
Eve[52]	106	34	Xception[24]	425	124
Eve_1[52]	59	32	YOLOv2[68]	252	70
Eve_2[52]	114	42	Zero_Shot_GCN[85]	36	35

Table 3.1: 50 Architectures in dL50a, with Their Triple and Entity Counts

Chapter 4

Knowledge Graph Embedding Algorithms

We selected six KG embedding algorithms for link prediction on dL50a. In this chapter, we first introduced each algorithm in Section 4.1. After that, we showed the hyper-parameter tuning process for each of them in Section 4.2. Later, in Section 4.3, we showed our experimental results.

4.1 Background for Each Algorithm

In the following sub-sections we described each of the six algorithms briefly. We included the main method they used and their comparisons.

4.1.1 TransE

TransE[18] was proposed in 2013. In TransE, relations are represented as translations in the embedding space. This means that embedding of the tail is closer in the continuous space to the head vector plus the relation vector. In other words, the aim of TransE embedding is to keep $h + r \approx t$.

The head and tail entities, as well as the relations are kept as a high dimensional vectors, i.e. $h, r, t \in \mathbb{R}^k$ where k is a hyper-parameter. The learning of the algorithm is done over these vector. In the end, the TransE learns the k-dimensional vector representations of the entities and relations.

During the learning, a set of triples are sampled from the training set. Using these samples, a set of corrupt triples are created. Corrupt in this case means that either the head or the tail of a triple is replaced with a random entity and that the triple no longer makes any sense for the data. These corrupt triples are used in the scoring function to decide the possibility of a triple.

The scoring function of TransE is given in 4.1, below. γ is the margin for the score function and is a hyper-parameter. The distance function, d, can either be L_1 or L_2 distance. The parameters of the embeddings will be updated according to the gradient of this equation.

$$\mathcal{L} = \sum_{(h,r,t)\in S} \sum_{(h',r,t')\in S'_{(h,r,t)}} [\gamma \ + \ d(\boldsymbol{h} + \boldsymbol{r}, \boldsymbol{t}) \ - \ d(\boldsymbol{h'} + \boldsymbol{r}, \boldsymbol{t'})]_+ \tag{4.1}$$

The main advantage of TransE is its simplicity. Since there are not many parameters to update and the scoring function is relatively simpler than of the other algorithms, TransE has a faster convergence time. For this reason, it can be used with the large datasets.

4.1.2 TransR

Similar to TransE, TransR[57] also uses a translation based method and was proposed in 2015. In TransE, we saw that the entities and relations are embedded in the same space $\mathbb{R}^k$. However, relations and entities have completely different characteristics, which TransE misses.

TransR proposes two different spaces for the elements, entity space and relation space. Before doing the translation, TransR first projects the entities h and t from the entity space to relation space, h_r and t_r, using the operation M_r. Then the translation operation takes place exactly the same as TransE, $\boldsymbol{h_r} + \boldsymbol{r} \approx \boldsymbol{t_r}$.

The training process of TransR is exactly the same as TransE. First a set of triples are sampled from the training set and used to create a set of corrupt triples. Both of the sets are then fed into the scoring function as a batch. The score is calculated and the parameters of the embeddings are updated with respect to the gradient of the score. The part TransR differs from TransE is the scoring function which is given below in Equation 4.2.

$$\mathcal{L} = \sum_{(h,r,t)\in S} \sum_{(h',r,t')\in S'_{(h,r,t)}} max(0, f_r(h,t) + \gamma - f_r(h',t'))_+ \tag{4.2}$$

where max function ensures the score is always positive. f_r function is given as $f_r(h,t) = ||h_r + r - t_r||_2^2$, where h_r and t_r is calculated as $h_r = hM_r$ and $t_r = tM_r$ respectively. M_R is the transform matrix from entity space to relation space.

Since TransR takes entities and relations on different spaces, it does not treat them similarly. In this manner, it gets more semantic information about each of them. In the experimental results this can be seen. TransR does better link prediction compared to TransE on Freebase[16] and WordNet[60] datasets.

4.1.3 KG2E

KG2E[38], like TransE and TransR, also uses translation based method. However, it does not use entity and relation vectors directly for representation. Instead, KG2E uses their Gaussian distribution while both representing them and translating them. TransE and TransR on the other hand, use deterministic vectors to embed entities and relations in a continuous space.

The entities and relations are modeled into a Gaussian distribution. The mean vector of this distribution denotes the positions of the elements, and the covariance matrix is used to denote the uncertainties of them. The notation of this representation is as follows: The triplet (h, r, t) is shown as $\mathcal{H}, \mathcal{R}$, and $\mathcal{T}$ as the corresponding Gaussian distributions. They calculated as $\mathcal{H} \sim \mathcal{N}(\boldsymbol{\mu_h}, \boldsymbol{\Sigma_h})$(similar for $\mathcal{R}$ and $\mathcal{T}$), where μ is the mean vector and Σ is the covariance matrix.

The transformation is expressed as $\mathcal{H} - \mathcal{T}$, which is equal to the probability distribution $\mathcal{P}_e$ of entities. Similarly we get the probability distribution, $\mathcal{P}_r$ for relations. The learning is based on measuring the similarities between $\mathcal{P}_r$ and $\mathcal{P}_e$. The scoring function is given in Equation 4.3.

$$\mathcal{L} = \sum_{(h,r,t) \in S} \sum_{(h',r,t') \in S'_{(h,r,t)}} [\mathcal{E}(\boldsymbol{h}, \boldsymbol{r}, \boldsymbol{t}) \ + \ \gamma \ - \ \mathcal{E}(\boldsymbol{h'}, \boldsymbol{r'}, \boldsymbol{t'})]_+ \tag{4.3}$$

where $\mathcal{E}$ is the energy function to measure the similarities of $\mathcal{P}_e$ and $\mathcal{P}_r$, and is given as:

$$\mathcal{E}(h, r, t) = log\mathcal{E}(\mathcal{P}_e, \mathcal{P}_r) = log\mathcal{N}(0; \mu_e - \mu_r, \Sigma_e + \Sigma_r) \tag{4.4}$$

In this manner KG2E also takes the uncertainties of entities and relations in to the calculation. This gives KG2E more information about the embeddings. The reported experimental result shows that with WordNet and Freebase datasets KG2E has a better link prediction accuracy than TransE and TransR.

4.1.4 RotatE

RotatE[76] was proposed in 2019 and it uses translation based embedding similar to the previous algorithms. RotatE differs from its predecessors in the way it calculates the distances between the embeddings, and in the way it creates negative samples. The objective of RotatE is to keep $t = h \circ r$, where $|r_i| = 1$, and $\circ$ is the element-wise product operation. The modulus of each element of the relation vector is constraint to 1. In this manner, r_i is kept in the form of $e^{i\theta_{r,i}}$. This means that r is rotating the head entity in clock-wise direction in the complex plane, hence the name RotatE comes from. Since $|r_i| = 1$, it only affects the phase of the entity embedding, not the amplitude.

The distance function is defined as: $d_r(\boldsymbol{h}, \boldsymbol{t}) = ||\boldsymbol{h} \circ \boldsymbol{r} - \boldsymbol{t}||$. This distance function is then used in the following scoring function given in Equation 4.5 below.

$$\mathcal{L} = -log\,\sigma\left(\gamma - d_r(\boldsymbol{h} - \boldsymbol{t})\right) - \sum_{i=1}^{n} \frac{1}{k} log\,\sigma\left(d_r(\boldsymbol{h_i'}, \boldsymbol{t_i'}) - \gamma\right) \tag{4.5}$$

where γ is the margin, σ is the sigmoid function, and (h_i', r, t_i') is the i-th negative triplet, and k is the hidden size. RotatE also uses a new method for negative sampling. Instead of just switching the head entity or tail entity with a random entry, they propose negative sampling based on the probability distribution of the entities. The negative samples created by the former method is inefficient in the sense that many samples that are created are determined to be false immediately as the training takes place. In the new method, they aim to achieve more logical negative samples that do not suffer from this problem. The distribution function they used to create the negative samples is shown in Equation 4.6.

$$p(h_j', r, t_j' | \{(h_i, r_i, t_i)\}) = \frac{\exp \alpha f_r(\boldsymbol{h_j'}, \boldsymbol{t_j'})}{\sum_i \exp \alpha f_r(\boldsymbol{h_j'}, \boldsymbol{t_j'})} \tag{4.6}$$

where α is the temparature of sampling and is a hyper-parameter.

4.1.5 SME

Semantic Matching Energy(SME)[17] was proposed in 2014 and it uses semantic matching model to embed the entities in a KG. In this model, first the vector representations of the elements of a triple is created, head embedding E_h, relation embedding, E_r, and tail embedding E_t. Then, from these embeddings, two different embedding is created. First embedding is created from the head entity and the relation, $E_{lhs} = g_{left}(E_h, E_r)$, and the second embedding is created from the relation and the tail entity, $E_{rhs} = g_{right}(E_r, E_t)$. g_{left} and g_{right} are parametrized functions whose parameters are tuned during training. Two different variations of these functions are used in SME: one of them is bilinear, which has more parameters and the other is linear, resulting in two variations of SME. In our study, we have used the bilinear, hence we shared the details of the bilinear version in the following parts.

E_{lhs} and E_{rhs} are used to measure the semantic similarities and are calculated according to Equation 4.7a and 4.7b respectively for bilinear version.

$$E_{lhs} = g_{left}(E_h, E_r) = (W_l \bar{\times}_3 E_r^T) E_h^T \ + \ b_l^T \tag{4.7a}$$

$$E_{rhs} = g_{right}(E_t, E_r) = (W_r \bar{\times}_3 E_r^T) E_t^T \ + \ b_r^T \tag{4.7b}$$

where W is weight, and b is bias. $\bar{\times}_3$ denotes vector-tensor product along the 3^{rd} mode[50]. The measurement is done through an energy function, $\mathcal{E}(h, r, t) = (E_{lhs}^T E_{rhs})$. Overall scoring function is given in Equation 4.8. From this equation the score for each triple is calculated and the parameters for embedding function is tuned with respect to the gradient.

$$\mathcal{L} = \sum_{x \in \mathcal{D}} \sum_{\tilde{x} \sim Q(\tilde{x}|x)} max(\mathcal{E}(x) \ - \ \mathcal{E}(\tilde{x}) \ + \ 1, 0) \tag{4.8}$$

where $\mathcal{D}$ is the triple set sampled from the training set, and Q is the corrupt triple set.

4.1.6 RESCAL

RESCAL[64] was proposed in 2011 and it mainly uses matrix factorization to embed the entities and relations together. In RESCAL, entire KG is modeled as a three dimensional tensor, $\mathcal{X}$. A tensor entry $(X)_{ijk} = 1$ denotes the fact that there is the triple (i-th entity, k-th relation, j-th entity) in the KG. If a triple does not exist in the KG, the corresponding slot in the tensor is marked with a zero.

$\mathcal{X}_k$ refers to the k-th slice of the tensor $\mathcal{X}$. Each slice of the tensor can be approximated using matrix-factorization as Equation 4.9 suggests.

$$\mathcal{X}_k \approx AR_kA^T, \quad \text{for } k = 1, ...m \tag{4.9}$$

where A is the matrix representing the entities and R_k is the matrix representing the relation for the k-th slice. The A and R_k matrices can be computed by solving the factorization problem, which can be approximated using optimization techniques.

In the end, the matrix A and R_k can be used to represent different entities together. In this sense, they can also be used to make link predictions for KG data. As an example, say there is a class with 4 students in it. Two of these students were born in the same city. The relation for the city is represented with the keyword "born_in". Thus, there will exist following two triples in the KG: <Bill, born_in, Irvine> and <John, born_in, Irvine>. If the entity and relation matrices are calculated with the matrix factorization method, the product $a^T_{Bill}R_{born\_in}a_{Irvine}$ will yield a similar value to $a^T_{John}R_{born\_in}a_{Irvine}$. In this manner, RESCAL can be used for link prediction purposes as in the previous algorithms.

4.2 Tuning the Hyper-Parameters of the Algorithms for dL50a

The hyper-parameters such as the learning rate, the embedding dimension etc. affect the performance of the algorithms. The authors of the algorithms tuned these hyper-parameters for Freebase or WordNet datasets. However, when the structure of the dataset changes, the hyper-parameters need to be tuned again. For this reason, we tuned the hyper parameters for all of the algorithms with dL50a.

We divided the dataset into two parts for tuning. The first part contains 15% of the triples. This portion is allocated for testing only. During the tuning of the hyper-parameters, this portion was never shown to the algorithms. In this manner, we ensured that the test results are not affected. The remaining 85% triples are used for 5-fold cross-validation. The tuning is done by sweeping through different values for each hyper-parameters and compared the resultant filtered mean rank value to select the best hyper-parameter value. For each value of a hyper-parameter, 5-fold cross validation took place. In each iteration, we picked one fold as the validation set and the remaining folds as the training set. This procedure repeated 5 times for each unique fold being the validation set. After the 5^{th} repetition, the results are averaged and recorded. Finally, the recorded values are compared for each hyper-parameter and the value with the best result is picked.

Before sweeping through different values for each hyper-parameter the range must be selected. In order to select the ranges, we kept changing the hyper-parameter's value in one direction drastically, constantly increase and constantly decrease. For each value, we used one fold for training and one fold for validation to see how the results are affected. If the filtered mean rank kept increasing while changing the value of a parameter, this means we do not need to sweep through after that point, since the results are not improving. In this manner, we have decided on the cut-off points for each hyper-parameter for each algorithm

and swept through the values in this range. The hyper-parameters for each of the algorithms, their sweeping ranges and the optimal values are given in Table 4.1. Hidden size is the dimension used in the embedding space for the representations of entities and relations. Learning rate decides how much the weights will be adjusted based on the gradient of the scoring function. Batch size refers to the number of training examples utilized in one iteration.

algo / hp	TransE	TransR	KG2E	RotatE	SME	RESCAL
learning_rate	0.01 → 0.1 pts = 50 opt = 0.0265	0.01 → 0.1 pts = 50 opt = 0.0375	0.001 → 0.01 pts = 50 opt = 0.0034	0.0001 → 0.1 pts = 100 opt = 0.0011	0.0005 → 0.1 pts = 300 opt = 0.0093	0.01 → 0.1 pts = 50 opt = 0.0688
hidden_size	5 → 30 pts = 26 opt = 12	5 → 30 pts = 26 opt[1]= 8/14	1 → 50 pts = 50 opt = 15	1 → 64 pts = 64 opt = 20	1 → 50 pts = 50 opt = 19	1 → 32 pts = 32 opt = 5
batch_size	50 → 256 pts = 50 opt = 66	50 → 256 pts = 50 opt = 58	50 → 500 pts = 100 opt = 340	40 → 256 pts = 60 opt = 175	50 → 500 pts = 200 opt = 54	50 → 256 pts = 50 opt = 79

Table 4.1: Hyper-parameter Sweeping Ranges, the Number of Points in the Given Intervals and the Resultant Optimal Values.

[1]For TransR, since there are two different embedding space, entity and relation, there are also two hyper-parameters for their dimensions. The first value is for the embedding space and the second is for relation space.

4.3 Experimental Results

After acquiring the optimal hyper-parameters for each of the algorithm, we trained each of them for 200 epochs with a fixed margin of 1.0. There are 3 main metrics we used to evaluate the results: mean rank, hits10, and hits5.

- Mean Rank:

  Mean rank is the metric that shows the prediction accuracy of the trained algorithm. It predicts a value for the missing information and tests it with the actual value. The rank is incremented for each wrong prediction. For example, if the correct entity is predicted in the 5^{th} prediction, the rank will be 5. Then the rank values for all triples are summed and averaged, hence the name *mean* rank.

  One thing to note here is that, sometimes the algorithm makes a prediction from the corrupt entities aka the negative samples. When this is the case, the result is meaningless, since we already know that it is not the correct answer. For that reason, we filter the rank values accordingly. In other words, if the predicted entity is from the corrupt set, we do not increment the rank. This metric is called the filtered mean rank. From this point, we will refer to filtered mean rank as mean rank.

- Hits10:

  Hits10 is the metric that shows whether the prediction is successful in the first 10 predictions. The value given is the fraction of successful predictions to all of them. For example, 0.30 hits10 means that, in 30% of the predictions, the algorithm has successfully predicted the correct answer in the first 10 trials.

  Similar to the mean rank, hits10 can also be filtered. Thus, exactly like mean rank from this point we will refer filtered hits10 as hits10.

- Hits5:

  Hits5 is exactly the same as hits10, but this time it is for the first 5 predictions. Filtering also applies to hits5 as well.

The results for each algorithm with these metrics are given in Table 4.2. They are also visualized in a bar chart in Figure 4.1 and in Figure 4.2. We did not include the non-filtered values in the Table, however, for comparison we included them in the figures.

algo / metric	TransE	TransR	KG2E	RotatE	SME	RESCAL
mean_rank	**301.950**	312.496	372.559	438.117	444.691	489.655
hits10	0.260	**0.309**	0.298	0.087	0.283	0.229
hits5	0.185	**0.271**	0.266	0.031	0.246	0.182

Table 4.2: Experimental Results for 6 Algorithms

Figure 4.1: Mean Rank Values for Different Algorithms

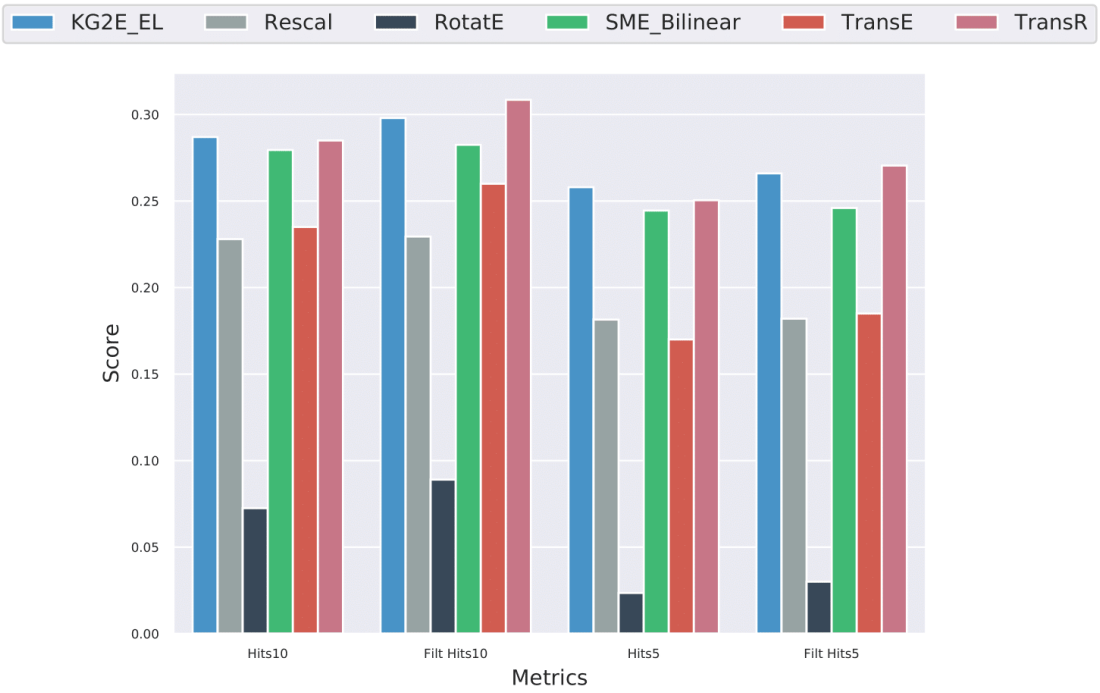

Figure 4.2: Hits10 and Hits5 Values for Different Algorithms

Based on the results we had, TransE has the lowest mean rank. This means, on the average, for all predictions it did the best link prediction. On the other hand, TransR has the best overall hits10 and hits5 rates. However, hits10 and hits5 only signifies the first predictions, not the overall performance. For this reason, we elected to precede with TransE for the next phase of the link prediction task.

The next phase of our study is to train the selected algorithm for many epochs and see the predictions it makes for missing information in triples. In the next chapter we discussed the experimental setting for this task and showed the results.

Chapter 5

Link Predictions

In the previuos chapter, we saw that TransE has the best overall performance with our dataset, dL50a. This time we will train TransE for many more iterations. In this manner, the prediction accuracy increases, so that we can see accurate results in the link prediction performance.

For training parameters we used the optimal hyper-parameters again ($learning\_rate$ = 0.0265, $hidden\_size$ = 12, $batch\_size$ = 66). The margin is kept constant at 1.0. With these paramters, we trained TransE with dL50a for 10,000 epochs.

5.1 Results

For the results, we first showed the metrics like mean rank and hits values. This time we inluded hits3 and hits1 as well. Hits1 shows how accurate the algorithm in link prediction in the first prediction. The metrics are showed in Table 5.1.

Metric	Value, 10k	Value, 200
Mean Rank	308.709	301.950
Hits10	0.3595	0.309
Hits5	0.3165	0.271
Hits3	0.2715	NA
Hits1	0.15	NA

Table 5.1: Experimental Results for TransE After 10,000 Epochs and 200 Epochs

In Table 5.1, we see that mean rank increased compared to training for 200 epochs. This is a result of overfitting. However, overfitting does not affect the first predictions. In fact, the prediction accuracy increased with more training. This can observed from the results, when the hits10 and hits5 values are compared for two different sets of training.

Overall, we got 15% accuracy for the first prediction. Some example predictions made by the algorithm are shown in Table 5.2.

	INPUT(HEAD AND RELATION)	PREDICTED TAILS	HITS
1	<posewarp/conv2d_50>, <has_activation> Correct Tail = <tanh>	$relu, lrelu, linear,$ $elu, sigmoid, \mathbf{tanh}$	6
2	<vgg19/conv2d_10>, <layer_type> Correct Tail = <conv2d>	$\mathbf{conv2d}$	1
3	<visual_redactions/conv2d_37>, <has_input> Correct Tail = <visual_redactions/add_10>	$visual\_redactions/conv2d\_37, visual\_redactions/batchnormalization\_37,$ $resnet/batchnormalization\_12, epinet/conv2d\_10, resnet/conv2d\_18,$ $visual\_redactions/conv2d\_15, resnet/conv2d\_13, yolov2/concatenate,$ $yolov2/conv2d\_21, epinet/conv2d\_15$	10+
4	<resnet/conv2d_20>, <has_kernel_size> Correct Tail = <(1,1)>	$(3,3), \mathbf{(1,1)}$	2
5	<resnet/loss>, <loss_type> Correct Tail = <categorical_crossentropy>	$wasserstein\_loss, binary\_crossentropy, fastnet/conv2d\_4,$ $eve\_big\_cnn/conv2d, \mathbf{categorical\_crossentropy}$	5
6	<discogan/dropout_2>, <has_rate> Correct Tail = <0.25>	$0.5, resnet/add\_13, 0.1, resnet/batchnormalization\_45,$ $resnet/conv2d\_47, resnet/batchnormalization\_44, \mathbf{0.25}$	7
7	<cyclegan/optimizer>, <optimizer_type> Correct Tail = <Adam>	$eve\_big\_cnn/optimizer, colorful/optimizer, pix2pix/loss,$ $(28,28,1), visual\_redactions/batchnormalization\_11,$ $epinet/batchnormalization\_2, ocr/maxpooling2d\_2,$ $fcnda/conv2d\_20, fcnda/conv2d\_17,$ $deblur\_gan/batchnormalization\_4$	10+

Table 5.2: Tail Prediction Examples for dL50a

In Table 5.2, on the left column the inputs and the correct answers are given. In the second column, the first 10 predictions made by TransE are listed. If the prediction is correct it is shown with a bold text. Also, predictions made after the correct prediction are not shown. On the right column the hit count is shown. This is the number of predictions TransE had to make in order to get the correct answer.

Based on the results we can make the following comments:

- Prediction of the Correct Class

  In the given examples, we can see that even if it does not get the answer in the first prediction, it tries to predict it among the correct class. For example, in the first example, the objective is to predict the loss type. Even though the first prediction is

not the correct type, it is still trying to predict a loss type. Similar pattern can be observed in the 6^{th} example as well.

Although this is true for most cases, in some of them it jumps to another class, missing the answer completely. This can be seen in the last example. Although the objective is to predict the optimizer type, it tries to predict it among optimizers(not the types), losses, layers etc. This causes it to miss-predict the answer each time.

- Data Property Prediction

  In the 3^{rd} example, we can see that the prediction for data properties also works. Even though the value can be everything, it predicts from the most probable ones and gets the correct answer.

- Conv2d Layer Prediction

  In the 2^{nd} example, we see that it gets the layer type in the first prediction. This can be seen in other examples that was not included here. Whenever the layer type is 2D convolution it predicts it with much higher accuracy than other layer types. The reason behind this is 2D convolution is one of the most used layers in Deep Learning models. Many of models are based on this layer. Thus, the more examples the algorithm obtains, the easier it can predict it.

Chapter 6

Conclusion

In this study we aimed to eliminate any missing information on a given Deep Learning model. The method we used was link prediction through knowledge graph embedding. We created a new dataset, dL50a, that contains 50 unique Deep Learning model in their knowledge graph representations. We used dL50a with 6 different knowledge graph embedding algorihtms and compared the performances of each of them. In this manner, we selected the most promising algorithm for extensive training with the dataset. In the end, we acquired the results for link prediction and got 15% accuracy for prediction in the first trial, 27.15% accuracy in the first three trials and 31.65% in the first five trials. We also observed that even though the algorithm cannot find the exact answer, it makes predictions in the correct category of information.

In the future, dL50a or an extended version of dL50a can be used to classify and cluster Deep Learning models. Instead of embedding of the each entity, if the entire model is embedded as a whole, classification will be possible. In this manner, a new search method can be developed for Deep Learning models, or novelty detection would become possible.

Bibliography

[1] Tensorflow tutorials: Fashion mnist example. `https://www.tensorflow.org/tutorials/`.

[2] E. Ahmed, M. Jones, and T. K. Marks. An improved deep learning architecture for person re-identification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[3] M. Allamanis, E. T. Barr, C. Bird, and C. Sutton. Learning natural coding conventions. In *Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 281–293. ACM, 2014.

[4] M. Allamanis, E. T. Barr, P. Devanbu, and C. Sutton. A survey of machine learning for big code and naturalness. *ACM Computing Surveys (CSUR)*, 51(4):81, 2018.

[5] M. Allamanis, M. Brockschmidt, and M. Khademi. Learning to represent programs with graphs. *arXiv preprint arXiv:1711.00740*, 2017.

[6] U. Alon, O. Levy, and E. Yahav. code2seq: Generating sequences from structured representations of code. *arXiv preprint arXiv:1808.01400*, 2018.

[7] M. Alshahrani, M. A. Khan, O. Maddouri, A. R. Kinjo, N. Queralt-Rosinach, and R. Hoehndorf. Neuro-symbolic representation learning on biological knowledge graphs. *Bioinformatics*, 33(17):2723–2730, 2017.

[8] M. Andrychowicz and K. Kurach. Learning efficient algorithms with hierarchical attentive memory. *arXiv preprint arXiv:1602.03218*, 2016.

[9] A. Angelova, A. Krizhevsky, V. Vanhoucke, A. Ogale, and D. Ferguson. Real-time pedestrian detection with deep network cascades. 2015.

[10] S. Antol, A. Agrawal, J. Lu, M. Mitchell, D. Batra, C. Lawrence Zitnick, and D. Parikh. Vqa: Visual question answering. In *The IEEE International Conference on Computer Vision (ICCV)*, December 2015.

[11] M. Arjovsky, S. Chintala, and L. Bottou. Wasserstein gan. *arXiv preprint arXiv:1701.07875*, 2017.

[12] G. Balakrishnan, A. Zhao, A. V. Dalca, F. Durand, and J. Guttag. Synthesizing images of humans in unseen poses. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[13] P. Baldi, P. Sadowski, and D. Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308, 2014.

[14] T. Ben-Nun, A. S. Jakobovits, and T. Hoefler. Neural code comprehension: a learnable representation of code semantics. In *Advances in Neural Information Processing Systems*, pages 3585–3597, 2018.

[15] D. Berthelot, T. Schumm, and L. Metz. Began: Boundary equilibrium generative adversarial networks. *arXiv preprint arXiv:1703.10717*, 2017.

[16] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor. Freebase: a collaboratively created graph database for structuring human knowledge. In *Proceedings of the 2008 ACM SIGMOD international conference on Management of data*, pages 1247–1250. AcM, 2008.

[17] A. Bordes, X. Glorot, J. Weston, and Y. Bengio. A semantic matching energy function for learning with multi-relational data. *Machine Learning*, 94(2):233–259, 2014.

[18] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko. Translating embeddings for modeling multi-relational data. In *Advances in neural information processing systems*, pages 2787–2795, 2013.

[19] G. Borghi, M. Venturelli, R. Vezzani, and R. Cucchiara. Poseidon: Face-from-depth for driver pose estimation. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[20] M. Busta, L. Neumann, and J. Matas. Deep textspotter: An end-to-end trainable scene text localization and recognition framework. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[21] F. Carminati, G. Khattak, M. Pierini, S. Vallecor-safa, and A. Farbin. Calorimetry with deep learning: particle classification, energy regression, and simulation for high-energy physics. In *NIPS*, 2017.

[22] X. Chen, Y. Duan, R. Houthooft, J. Schulman, I. Sutskever, and P. Abbeel. Infogan: Interpretable representation learning by information maximizing generative adversarial nets. In *Advances in neural information processing systems*, pages 2172–2180, 2016.

[23] S. R. Chhetri, J. Wan, A. Canedo, and M. A. Al Faruque. Design automation using structural graph convolutional neural networks. In *Design Automation of Cyber-Physical Systems*, pages 237–259. Springer, 2019.

[24] F. Chollet. Xception: Deep learning with depthwise separable convolutions. *CoRR*, abs/1610.02357, 2016.

[25] M. Cvitkovic, B. Singh, and A. Anandkumar. Deep learning on code with an unbounded vocabulary. In *CAV*, 2018.

[26] L. Ding and C. Xu. Weakly-supervised action segmentation with iterative soft boundary assignment. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[27] Y. Duan, Z. Wang, J. Lu, X. Lin, and J. Zhou. Graphbit: Bitwise interaction mining via deep reinforcement learning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8270–8279, 2018.

[28] R. O. Duda, P. E. Hart, and N. J. Nilsson. Subjective bayesian methods for rule-based inference systems. In *Readings in artificial intelligence*, pages 192–199. Elsevier, 1981.

[29] S. Faezi, S. R. Chhetri, A. V. Malawade, J. C. Chaput, W. H. Grover, P. Brisk, and M. A. Al Faruque. Oligo-snoop: A non-invasive side channel attack against dna synthesis machines.

[30] A. Faruque, M. Abdullah, S. R. Chhetri, A. Canedo, and J. Wan. Acoustic side-channel attacks on additive manufacturing systems. In *Proceedings of the 7th International Conference on Cyber-Physical Systems*, page 19. IEEE Press, 2016.

[31] T. Fischer and C. Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[32] W. Ganglberger, G. Gritsch, M. M. Hartmann, F. Fürbass, H. Perko, A. M. Skupch, and T. Kluge. A comparison of rule-based and machine learning methods for classification of spikes in eeg. *JCM*, 12(10):589, 2017.

[33] D. Ha and D. Eck. A neural representation of sketch drawings. *CoRR*, abs/1704.03477, 2017.

[34] N. Hadad, L. Wolf, and M. Shahar. A two-step disentanglement method. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 772–780, 2018.

[35] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.

[36] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[37] M. He and D. He. Deep learning based approach for bearing fault diagnosis. *IEEE Transactions on Industry Applications*, 53(3):3057–3065, 2017.

[38] S. He, K. Liu, G. Ji, and J. Zhao. Learning to represent knowledge graphs with gaussian embedding. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, pages 623–632. ACM, 2015.

[39] J. Heaton, N. Polson, and J. H. Witte. Deep learning for finance: deep portfolios. *Applied Stochastic Models in Business and Industry*, 33(1):3–12, 2017.

[40] J. Heaton, N. G. Polson, and J. H. Witte. Deep learning in finance. *arXiv preprint arXiv:1602.06561*, 2016.

[41] E. Hernández, V. Sanchez-Anguix, V. Julian, J. Palanca, and N. Duque. Rainfall prediction: A deep learning approach. In *International Conference on Hybrid Artificial Intelligence Systems*, pages 151–162. Springer, 2016.

[42] Y. Hoshen and S. Peleg. An egocentric look at video photographer identity. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[43] M. Hossain, B. Rekabdar, S. J. Louis, and S. Dascalu. Forecasting the weather of nevada: A deep learning approach. In *2015 international joint conference on neural networks (IJCNN)*, pages 1–6. IEEE, 2015.

[44] F. Huang and C. Smidts. Causal mechanism graph a new notation for capturing cause-effect knowledge in software dependability. *Reliability Engineering & System Safety*, 158:196–212, 2017.

[45] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[46] P. Isola, J.-Y. Zhu, T. Zhou, and A. A. Efros. Image-to-image translation with conditional adversarial networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1125–1134, 2017.

[47] J. Johnson, A. Alahi, and F. Li. Perceptual losses for real-time style transfer and super-resolution. *CoRR*, abs/1603.08155, 2016.

[48] T. Kim, M. Cha, H. Kim, J. K. Lee, and J. Kim. Learning to discover cross-domain relations with generative adversarial networks. *CoRR*, abs/1703.05192, 2017.

[49] Y. Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[50] T. G. Kolda and B. W. Bader. Tensor decompositions and applications. *SIAM review*, 51(3):455–500, 2009.

[51] P. T. Komiske, E. M. Metodiev, and M. D. Schwartz. Deep learning in color: towards automated quark/gluon jet discrimination. *Journal of High Energy Physics*, 2017(1):110, 2017.

[52] J. Koushik and H. Hayashi. Improving stochastic gradient descent with feedback. *arXiv preprint arXiv:1611.01505*, 2016.

[53] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

[54] O. Kupyn, V. Budzan, M. Mykhailych, D. Mishkin, and J. Matas. Deblurgan: Blind motion deblurring using conditional adversarial networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8183–8192, 2018.

[55] M. Liang and X. Hu. Recurrent convolutional neural network for object recognition. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[56] D. Liben-Nowell and J. Kleinberg. The link-prediction problem for social networks. *Journal of the American society for information science and technology*, 58(7):1019–1031, 2007.

[57] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu. Learning entity and relation embeddings for knowledge graph completion. In *Twenty-ninth AAAI conference on artificial intelligence*, 2015.

[58] D. Lu and J. Antony. Optimization of multiple responses using a fuzzy-rule based inference system. *International Journal of Production Research*, 40(7):1613–1625, 2002.

[59] A. Makhzani, J. Shlens, N. Jaitly, and I. J. Goodfellow. Adversarial autoencoders. *CoRR*, abs/1511.05644, 2015.

[60] G. A. Miller. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41, 1995.

[61] S. Montazzolli Silva and C. Rosito Jung. License plate detection and recognition in unconstrained scenarios. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[62] S. Motiian, M. Piccirilli, D. A. Adjeroh, and G. Doretto. Unified deep supervised domain adaptation and generalization. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[63] M. A. Musen et al. The protégé project: a look back and a look forward. *AI matters*, 1(4):4, 2015.

[64] M. Nickel, V. Tresp, and H.-P. Kriegel. A three-way model for collective learning on multi-relational data. In *ICML*, volume 11, pages 809–816, 2011.

[65] S. Oramas, V. C. Ostuni, T. D. Noia, X. Serra, and E. D. Sciascio. Sound and music recommendation with knowledge graphs. *ACM Transactions on Intelligent Systems and Technology (TIST)*, 8(2):21, 2017.

[66] T. Orekondy, M. Fritz, and B. Schiele. Connecting pixels to privacy and utility: Automatic redaction of private information in images. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[67] W. Ouyang and X. Wang. Joint deep learning for pedestrian detection. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2056–2063, 2013.

[68] J. Redmon and A. Farhadi. YOLO9000: better, faster, stronger. *CoRR*, abs/1612.08242, 2016.

[69] T. Salimans, I. J. Goodfellow, W. Zaremba, V. Cheung, A. Radford, and X. Chen. Improved techniques for training gans. *CoRR*, abs/1606.03498, 2016.

[70] A. G. Salman, B. Kanigoro, and Y. Heryadi. Weather forecasting using deep learning techniques. In *2015 International Conference on Advanced Computer Science and Information Systems (ICACSIS)*, pages 281–285. IEEE, 2015.

[71] P. Sermanet and Y. LeCun. Traffic sign recognition with multi-scale convolutional networks. In *IJCNN*, pages 2809–2813, 2011.

[72] C. Shin, H.-G. Jeon, Y. Yoon, I. So Kweon, and S. Joo Kim. Epinet: A fully-convolutional neural network using epipolar geometry for depth from light field images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4748–4757, 2018.

[73] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[74] R. Socher, D. Chen, C. D. Manning, and A. Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934, 2013.

[75] W. Sultani, C. Chen, and M. Shah. Real-world anomaly detection in surveillance videos. *CoRR*, abs/1801.04264, 2018.

[76] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang. Rotate: Knowledge graph embedding by relational rotation in complex space. *arXiv preprint arXiv:1902.10197*, 2019.

[77] S. Tang, M. Andriluka, B. Andres, and B. Schiele. Multiple people tracking by lifted multicut and person re-identification. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[78] B. Taskar, M.-F. Wong, P. Abbeel, and D. Koller. Link prediction in relational data. In *Advances in neural information processing systems*, pages 659–666, 2004.

[79] Y. Tian, J. Shi, B. Li, Z. Duan, and C. Xu. Audio-visual event localization in unconstrained videos. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[80] P. Upchurch, J. Gardner, G. Pleiss, R. Pless, N. Snavely, K. Bala, and K. Weinberger. Deep feature interpolation for image content changes. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 7064–7073, 2017.

[81] J. Wan, B. S. Pollard, S. R. Chhetri, P. Goyal, M. A. A. Faruque, and A. Canedo. Future automation engineering using structural graph convolutional neural networks. *CoRR*, abs/1808.08213, 2018.

[82] Y. Wan, Z. Zhao, M. Yang, G. Xu, H. Ying, J. Wu, and P. S. Yu. Improving automatic source code summarization via deep reinforcement learning. In *Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering*, pages 397–407. ACM, 2018.

[83] J. Wang, Y. Ma, L. Zhang, R. X. Gao, and D. Wu. Deep learning for smart manufacturing: Methods and applications. *Journal of Manufacturing Systems*, 48:144–156, 2018.

[84] Q. Wang, Z. Mao, B. Wang, and L. Guo. Knowledge graph embedding: A survey of approaches and applications. *IEEE Transactions on Knowledge and Data Engineering*, 29(12):2724–2743, 2017.

[85] X. Wang, Y. Ye, and A. Gupta. Zero-shot recognition via semantic embeddings and knowledge graphs. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2018.

[86] Z. Wang, J. Zhang, J. Feng, and Z. Chen. Knowledge graph embedding by translating on hyperplanes. In *Twenty-Eighth AAAI conference on artificial intelligence*, 2014.

[87] J. Wei and G. J. Mendis. A deep learning-based cyber-physical strategy to mitigate false data injection attack in smart grids. In *2016 Joint Workshop on Cyber-Physical Security and Resilience in Smart Grids (CPSR-SG)*, pages 1–6. IEEE, 2016.

[88] C. Xiong, R. Power, and J. Callan. Explicit semantic ranking for academic search via knowledge graph embedding. In *Proceedings of the 26th international conference on world wide web*, pages 1271–1279. International World Wide Web Conferences Steering Committee, 2017.

[89] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng. Embedding entities and relations for learning and inference in knowledge bases. *arXiv preprint arXiv:1412.6575*, 2014.

[90] S.-m. Yang, M. Nagamachi, and S.-y. Lee. Rule-based inference model for the kansei engineering system. *International Journal of Industrial Ergonomics*, 24(5):459–471, 1999.

[91] R. Zhang, P. Isola, and A. A. Efros. Colorful image colorization. In *European conference on computer vision*, pages 649–666. Springer, 2016.

[92] Y. Zhang, P. David, and B. Gong. Curriculum domain adaptation for semantic segmentation of urban scenes. In *The IEEE International Conference on Computer Vision (ICCV)*, Oct 2017.

[93] Y. Zhang, D. Zhou, S. Chen, S. Gao, and Y. Ma. Single-image crowd counting via multi-column convolutional neural network. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2016.

[94] J. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. *CoRR*, abs/1703.10593, 2017.