Markov Process-Based Graph Convolutional Networks for Entity Classification in Knowledge Graphs

Johannes Mäkelburg[1*][a],Yiwen Peng[2*][b], Mehwish Alam[2][c], Tobias Weller[3] and Maribel Acosta[1][d]
[1]*School of CIT, Technical University of Munich, Germany*
[2]*Télécom Paris, Institut Polytechnique de Paris, France*
[3] *Independent Researcher*
{*johannes.maekelburg, maribel.acosta*}*@tum.de*, {*yiwen.peng, mehwish.alam*}*@telecom-paris.fr*

Keywords: Entity Classification, Graph Convolutional Network, Markov Process, Evidential Learning, Knowledge Graph

Abstract: Despite the vast amount of information encoded in Knowledge Graphs (KGs), information about the class affiliation of entities remains often incomplete. Graph Convolutional Networks (GCNs) have been shown to be effective predictors of complete information about the class affiliation of entities in KGs. However, these models do not learn the class affiliation of entities in KGs incorporating the complexity of the task, which negatively affects the models' prediction capabilities. To address this problem, we introduce a Markov process-based architecture into well-known GCN architectures. This end-to-end network learns the prediction of class affiliation of entities in KGs within a Markov process. The number of computational steps is learned during training using a geometric distribution. At the same time, the loss function combines insights from the field of evidential learning. The experiments show a performance improvement over existing models in several studied architectures and datasets. Based on the chosen hyperparameters for the geometric distribution, the expected number of computation steps can be adjusted to improve efficiency and accuracy during training.

1 Introduction

Knowledge Graphs (KGs) encode factual knowledge in the form of triples (subject-relation-object) and have emerged as a compelling abstraction for organizing semi-structured data, capturing relationships among entities. The facts available in KGs are used in many application areas such as recommendation (Wang et al., 2019), information retrieval (Xiong and Callan, 2015), and question answering (Yasunaga et al., 2021) for improving the performance of these systems by providing background or auxiliary information. This relevance imposes a significant importance on a KG providing comprehensive information about the encoded entities. In particular, encoding knowledge about the class affiliation of entities is of great importance for automatic reasoning and inferencing of information contained in a KG. Despite the enormous effort made to keep the knowledge encoded in the KGs up-to-date and consistent (Heist et al., 2020), KGs are often incomplete majorly due to automated constructions of KGs. To complete missing knowledge, in particular missing class affiliation of entities in KGs, various methods based on machine learning have been introduced (Bordes et al., 2013a; Kipf and Welling, 2016). Neural networks, especially Graph Convolutional Neural Networks (GCNs), have proven to be very effective in completing class affiliation of entities in KGs (Schlichtkrull et al., 2018). However, in neural network based methods, the computation cost grows with the size of the input data, but not with the complexity of the problem being learned. In recent developments in automated machine learning, models perform conditional computation based on probabilistic variables that are used to dynamically adjust the number of computation steps (Banino et al., 2021). The adjustment of the computational budget, in particular the computation steps, is known as pondering. Yet, existing graph-based machine learning algorithms do not consider the complexity of the task to adjust the number of computation steps to learn the parameters. We address this issue and introduce a GCN-based model that learns to adapt the amount of computational steps based on the task at hand.

[a] https://orcid.org/0009-0001-3821-7817
[b] https://orcid.org/0009-0007-7902-4097
[c] https://orcid.org/0000-0002-7867-6612
[d] https://orcid.org/0000-0002-1209-2868
[*] These authors contributed equally to this work.

In this work, we propose Markov Process and Evidential with Regularization Loss (MPERL) that builds upon the previous idea of dynamically adjusting the number of computational steps based on the input of the model. We introduce an end-to-end Graph Convolutional Network (GCN) model that is learned within a Markov process and use recent developments in the field of evidential learning (Sensoy et al., 2018; Amini et al., 2020). Previous work has demonstrated the high performance of evidence-based models. Unlike models using a softmax function, evidence-based models are effective predictors that do not make overconfident predictions. We therefore follow an evidence-based approach as well. The Markov process in which the model is learned consists of two states: (i) the *continue* state, indicating further computational steps, and (ii) the *halt* state, indicating the end of the computational steps. The overall probability of halting at each step is modeled as a geometric distribution. Unlike previous work for entity classification in KGs, MPERL is a graph-based model that dynamically adjusts the number of computational steps according to complexity. MPERL further shows its feasibility on the task of entity type prediction, i.e., inferring the knowledge about the class affiliation of an entity. In the rest of this paper, we are treating entity type prediction as a classification task where we perform single-label classification for smaller datasets (specifically designed for this purpose) and multi-label classification on larger datasets. The experimental results show that MPERL (GCN with markov process and evidential loss) outperforms vanilla GCN as well as various other baselines. The ablation studies show that the use of both markov process and the evidential learning loss provide significant increase in the performance of the MPERL. Overall our paper makes the following contributions:

- We introduce a Graph Convolutional Network based model trained within a Markov process, using an evidential loss function.
- We demonstrate the performance of the model in predicting missing class affiliations of entities using single- and multi-label classification.
- We show the effect on the model when adapting the number of computational or Markov steps.
- We show the effectiveness of each of the components of the model on the overall results with the help of an ablation study.

The paper is structured as follows: Section 2 discusses the recent works related to representation learning over KGs for entity classification. Section 3 details the proposed approach while Section 4 shows the effectiveness of MPERL with the help of thorough experimentation over various sizes of the datasets as well as the ablation study. Finally, Section 5 concludes the study and discusses future directions.

2 Related Work

Different learning approaches have been applied to the problem of entity classification in KGs. Relational Graph Convolutional Networks (R-GCN) (Schlichtkrull et al., 2018) uses the structure of KGs to generate embeddings based on local neighbors in order to predict classes (Kipf and Welling, 2016; Hamilton et al., 2017). Due to their strong performance on graph-structured data, GCN models have been particularly used and extended in recent years to tackle entity classification (Schlichtkrull et al., 2018; Chen et al., 2019; Zangari et al., 2021), relation classification (Long et al., 2021), and KG alignment (Berrendorf et al., 2020; Wang et al., 2018). Gated Relational Graph Neural Network (GRGNN) (Chen et al., 2019) introduced a gate mechanism to leverage hidden states of current node and its neighbors to target entity classification problem in KGs. Whereas Relational Graph Attention Networks (Busbridge et al., 2019) and Multilayer Graph Attention Network (Zangari et al., 2021) use masked self-attentional layers to learn the weighting factor of neighboring node's features and were extended with intra- and inter-layer connections between nodes. Evidential Relational-Graph Convolutional Networks (*E*-R-GCN) (Weller and Paulheim, 2021) extend R-GCN (Schlichtkrull et al., 2018) with an evidential loss to represent the predictions of the model as a distribution over possible softmax outputs and estimate the associated evidence to learn both aleatory and epistemic uncertainty in entity classification. In contrast to these approaches, our work also implements a Markov process to learn the model. Moreover, translational KG embeddings (e.g, TransE (Bordes et al., 2013a) and extensions) and factorization-based KG embeddings (e.g., DistMult (Yang et al., 2015) and RESCAL (Nickel et al., 2011)) have been proposed. In general, these embeddings are particularly effective for link prediction, but less for entity classification (Dong et al., 2019). TransET (Wang et al., 2021) is an extension of TransE (Bordes et al., 2013a) that implements a convolution-based projection of entities into a type-specific representation to address entity classification. ConnectE (Zhao et al., 2020) is also a translational-based approach that learns two distinct embedding models of the entities and connects them via a joint model to predict entity types. Ridle (Weller and

Acosta, 2021) computes a distribution over the use of relations of entities using a stochastic factorization model. Besides translational and factorization-based embeddings, RDF2Vec (Ristoski and Paulheim, 2016) generates a sequence of nodes using random walks and Weisfeiler-Lehman subtree RDF graph kernels that are passed to Word2Vec language model for learning low-dimensional numerical representations of entities. The learned embeddings preserve similar entities closer in the vector space, which makes RDF2Vec suitable for entity classification (Sofronova et al., 2020; Biswas et al., 2018; Kejriwal and Szekely, 2017). Our solution differs from these approaches in the combination of evidential learning with a Markov process. This allows our approach to learn embeddings tailored to entity classification while adjusting the number of computational steps according to the complexity of the task at hand.

Other KG representations for entity classification have also been proposed which utilize semantic information related to an entity. Cat2Type (Biswas et al., 2021) creates representations for entities based on the textual information available in the Wikipedia category names using language models and the category network information. In addition to textual information related to entities, GRAND (Biswas et al., 2022) uses several kinds of graphs such as entity based, relation based, and random walks for considering the strcutured contextual information of an entity. In (Riaz et al., 2023), the authors perform entity typing based only on the labels as well as descriptions of the entities using BERT-based models. These approaches can only be applied to KGs where class affiliation can be predicted by the relation distribution (Weller and Acosta, 2021) or where additional semantic information is available. Relational aggregation graph attention network (RACE2T) (Zou et al., 2022) proposes a method consisting of an encoder which consists of the attention coefficient between entities further used to aggregate the information of relations and entities in the neighborhood of the entity. The decoder is based on a convolutional neural network. Lastly, ASSET (Zahera et al., 2021) is a semi-supervised approach that learns from massive unlabeled data for entity classification. Compared to our work and the related work above, ASSET does not learn embeddings itself, but uses existing ConnectE (Zhao et al., 2020) embeddings learned beforehand on the KG.

3 Our Approach: MPERL

In this section, we introduce our method Markov Process and Evidential with Regularization Loss (MPERL), that extends current Graph Convolutional Networks (GCN) to perform entity classification in KGs. For this purpose, we first introduce the definition of a KG and the associated research problem.

Definition 1. *A Knowledge Graph $\mathcal{KG}$ is a tuple $(\mathcal{E}, \mathcal{R}, \mathcal{L}, \mathcal{C})$, where the pair-wise disjoint sets $\mathcal{E}$, $\mathcal{R}$, $\mathcal{L}$, and $\mathcal{C}$ correspond to the set of entities, relations, literals, and types or classes, respectively. A statement in $\mathcal{KG}$ is modelled as a triple (s, r, o), with $s \in \mathcal{E} \cup \mathcal{R} \cup \mathcal{C}$, $r \in \mathcal{R}$, and $o \in \mathcal{E} \cup \mathcal{R} \cup \mathcal{L} \cup \mathcal{C}$.*

The problem of entity classification in a $\mathcal{KG}$ is to predict statements (e, r, C) that should be in $\mathcal{KG}$, where $e \in \mathcal{E}$, $r \in \mathcal{R}$ denotes the class affiliation relationship, and $C \in \mathcal{C}$ is a class. To address this problem, we present both the architecture and the learning process of MPERL in Section 3.1. In Section 3.2, the loss function for learning the parameters of the model is presented.

3.1 Markov Process Extensions for Entity Classification

Overview. We model the entity classification problem as a supervised learning problem. Figure 1 shows the overall architecture of our proposed solution MPERL, which integrates a Markov process into a GCN-based model, e.g., R-GCN (Schlichtkrull et al., 2018). First, a representation of the entities in the KG is learned (cf. Eqs. 1-3), which relies on GCNs to represent entities from the KG and are used in each step of the Markov process to calculate the hidden layers. The Markov process in which the model is learned is defined in Eqs. 4-6. We use a generalized geometric distribution to model the transition probabilities of the two states (*halt* and *continue*) of the Markov process. We learn with parameter $\lambda_n^{(i)}$ (cf. Eq. 5) a parameter from which we can derive the probability in which step of the Markov process the halt state is reached (cf. Eq. 6). By learning this parameter, the number of Markov steps and, thus, the number of epochs is adapted based on the input of the model. The final output of MPERL is given in Eqs. 7-9. In Eq. 7, the features of the individual steps of the Markov process are aggregated by weighted means and used to parameterize a Dirichlet distribution (Eq. 8). We use a Dirichlet distribution since this is the only conjugate prior for a categorical distribution used to indicate the probability of class affiliation of an entity in a KG. The prediction of a sample i is described by the expected probability of the Dirichlet distribution in Eq. 9. We use the expected probability as prediction for entity types due to its property of unbiased manners.

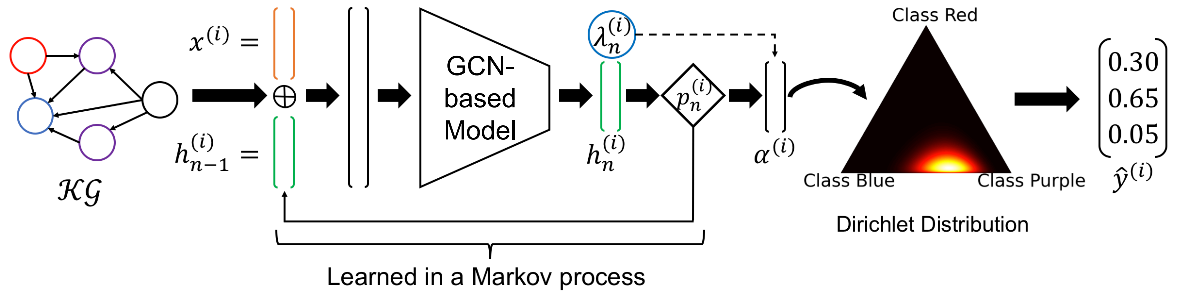


Figure 1: Approach for entity classification using MPERL. MPERL gets as input a one-hot encoding of the entity ID (denoted $x^{(i)}$) and the learned hidden features from the previous Markov step. The GCN-based model uses the structure of the KG to compute the hidden features $h_n^{(i)}$ and the halting probability $\lambda_n^{(i)}$. The prediction $\hat{y}^{(i)}$ is based on the Dirichlet parameters $\alpha^{(i)}$.

Learning Process. For each entity $e^{(i)} \in \mathcal{E}$ of the KG $\mathcal{KG}$ we initialize each entity representation by concatenating one hot encoding with hidden state of previous markov step. We denote this vector as $x^{(i)}$. This vector is concatenated with the hidden feature representation of the entity $e^{(i)}$ of the previous Markov step denoted as $h^{(i)}_{n-1}$. Initially, in step $n = 1$, $h_0^{(i)}$ is a null vector, so the feature representation is $h_0^{(i)} = \overrightarrow{0}$. $n \in [1, N]$ denotes the current step of the Markov process where N is the maximum number of Markov steps. The concatenation of the two vectors $x^{(i)}$ and $h^{(i)}_{n-1}$ is used as input to the neural network in step n in the Markov process. We denote the neural network input, i.e., in layer $l = 0$, of a sample i in Markov step n as follows:

$$h_n^{(i)[0]} = [x^{(i)} \parallel h^{(i)}_{n-1}] \tag{1}$$

where $[\ \parallel\ ]$ is the concatenation operation and the number in square brackets in superscript denotes the considered layer.

By incorporating the hidden feature representation $h^{(i)}_{n-1}$ from the previous step, the learned features are reused to enable faster convergence. The fundamental concept is similar to GCRNN, although rather than using $h^{(i)}_{n-1}$ as the only input to MPERL, the concatenation of $x^{(i)}$ and $h^{(i)}_{n-1}$ is used to avoid overfitting.

The hidden representation of an entity $e^{(i)} \in \mathcal{E}$ in layer $l+1$ is then computed using a simple propagation model to calculate the forward pass update. For updating the entity representation, we apply full sampling or partial neighbourhood sampling (for larger datasets) during message passing phase in graph neural networks.

$$h_n^{(i)[l+1]} = \phi\left(\sum_{r \in \mathcal{R}} \sum_{j \in \mathcal{N}_i^r} \frac{1}{|\mathcal{N}_i^r|} W_r^{[l]} h_n^{(j)[l]} + W_0^{[l]} h_n^{(i)[l]}\right) \tag{2}$$

ϕ denotes the ReLU function and $\mathcal{N}_i^r$ denotes the indices of neighboring nodes with relation $r \in \mathcal{R}$ to the node with index i. $l \in [1, L]$ denotes the layer with L as the number of layers in the neural network. In Eq. 2, the feature representations of the neighboring nodes of the node with index i are relation-specifically aggregated with the weight matrix $W_r^{[l]}$ and normalized by the number of neighboring nodes ($\frac{1}{|\mathcal{N}_i^r|}$). This relation-specific transformation is summed up and extended by a self-loop to include the current representation of the node itself. The ReLU function ϕ is applied as a non-linear activation function. For regularizing the network layers' weights, basis decomposition is used to avoid a rapid growth in the number of parameters with the number of relations in the graph. Basis decomposition uses a linear combination of basis transformations $V_b^{[l]} \in \mathbb{R}^{d^{[l+1]} \times d^{[l]}}$ with coefficients $a_{rb}^{[l]}$ such that only the coefficients depend on r.

$$W_r^{[l]} = \sum_{b=1}^{B} a_{rb}^{[l]} V_b^{[l]} \tag{3}$$

For ease of reading, we denote the hidden representation of the last layer of sample i in step n as $h_n^{(i)}$. MPERL learns a function $\mathcal{S}(x^{(i)}, h^{(i)}_{n-1})$ that outputs the parameters of a Dirichlet distribution $\alpha^{(i)}$, used as conjugate prior of a categorical distribution from which the predictions $\hat{y}^{(i)}$ are drawn, the hidden features $h_n^{(i)}$ and the probability of halting $\lambda_n^{(i)}$ at current step. The function $\mathcal{S}(x^{(i)}, h^{(i)}_{n-1})$ is learned within a Markov process. We use $\lambda_n^{(i)}$ to learn the optimal value n. The Markov process uses a Bernoulli random variable, which we denote as Λ_n, to represent the two states *continue* ($\Lambda_n = 0$) and *halt* ($\Lambda_n = 1$). *halt* is an absorbing state, meaning that, once entered, cannot be left. This defines the end of learning within the Markov process. The Markov process initially starts in the *continue* state, therefore $\Lambda_0 = 0$

holds. The transition probability of a sample i that the state *halt* is assumed in step n, given that the previous step was *continue* is expressed by the following conditional probability:

$$P(\Lambda_n = 1 | \Lambda_{n-1} = 0) = \lambda_n^{(i)} \quad \forall\, 1 \leq n \leq N \tag{4}$$

The conditional probability $\lambda_n^{(i)}$ is computed using a sigmoid function σ with parameters $U \in \mathbb{R}^{d^{[L]} \times 1}$ and $h_n^{(i)} \in \mathbb{R}^{d^{[L]} \times 1}$, where $d^{[L]}$ denotes the number of dimensions of the hidden features of $h_n^{(i)}$ in the last layer.

$$\lambda_n^{(i)} = \sigma(h_n^{(i)}) = \frac{1}{1 + e^{-U^T h_n^{(i)}}} \tag{5}$$

The probability of entering the state *halt* in step $n \in [1, N]$ can be derived by means of the following generalized geometric distribution p_n.

$$p_n^{(i)} = \lambda_n^{(i)} \prod_{s=1}^{n-1} (1 - \lambda_s^{(i)}) \tag{6}$$

$p_n^{(i)}$ defines for sample (i.e., entity) i the probability of entering the absorbing state $\Lambda_n = 1$ for the first time in step n, based on $\lambda_n^{(i)}$.

Once the absorbing state $\Lambda_n = 1$ has been entered, the Markov process terminates and the learned hidden features of each step, $h_s^{(i)}$ with $1 \leq s \leq n$, are aggregated. In contrast to existing work, which uses the final output $h_n^{(i)}$ (Banino et al., 2021) or the weighted average across all steps for prediction ($\sum_{s=1}^{n} \hat{y}_s \lambda_s$), we follow a different approach and use a weighted average of the hidden features across all steps as final hidden feature.

$$h^{(i)} = \sum_{s=1}^{n} h_s^{(i)} \lambda_s^{(i)} \tag{7}$$

In Eq. 7, $\lambda_s^{(i)}$ denotes for sample i the probability to enter the absorbing state in step s. A high $\lambda_s^{(i)}$ value, with $1 \leq s \leq n$, is due to a fitting feature representation $h_s^{(i)}$ to predict $\hat{y}^{(i)}$, thus, a high significance is assigned to this feature representation when aggregating $h^{(i)}$.

$h^{(i)} \in \mathbb{R}^{d^{[L]}}$ is thus an aggregation of the features of the individual steps in the Markov process and is used as parameter of the conjugate prior. The general idea is that no softmax function is used to predict the categorical values, but a conjugate prior categorical distribution from which the predictions are drawn. The advantage over a softmax function is that not just one point estimator is available for prediction, but a large number of categorical distributions that can be drawn from the conjugate prior. At the same time, the uncertainty can be quantified by the conjugate prior. Given the supervised learning problem for predicting categorical values $y^{(i)} \in \mathcal{C}$, where $\mathcal{C}$ is the set of classes in $\mathcal{KG}$ and K denotes the number of classes (i.e. $\mathcal{C} = \{C_1, \ldots, C_K\}$), we use a Dirichlet distribution as conjugate prior of a categorical distribution. Depending on the predictions of the Dirichlet distribution's parameters, the concentration of the drawn distributions can be on one class or, if uncertainty is large, it can be spread over several classes.

In order to determine the Dirichlet parameters $\alpha^{(i)} \in \mathbb{R}_+^K$ for sample i, where $K \geq 2$ always holds, we use the aggregated hidden feature representation $h^{(i)}$ (see Eq. 7) and a ReLU function ϕ to determine the parameters of the Dirichlet distribution as follows:

$$\alpha^{(i)} = \phi(h^{(i)}) + 1 \tag{8}$$

The ReLU function ϕ outputs values in the range $[0, \infty)$. Since we add 1 in the Eq. 8, the constraint of the Dirichlet parameter $\alpha^{(i)} \in \mathbb{R}_{>1}^K$ holds. The prediction $\hat{y}^{(i)}$ of a sample is the expected probability of the Dirichlet distribution with parameter $\alpha^{(i)}$.

$$\hat{y}^{(i)} = \frac{\alpha^{(i)}}{\sum_{k=1}^{K} \alpha_k^{(i)}} \tag{9}$$

3.2 Evidential with Regularization Loss

Based on existing work (Weller and Paulheim, 2021), we have chosen to use an evidential loss function rather than a cross-entropy function (Banino et al., 2021; Schlichtkrull et al., 2018). However, to simultaneously control the number of steps within the Markov process, our loss function L consists of two terms L_{ev} and L_{reg}. The evidential loss L_{Ev} optimizes the parameter for fitting the predictions $\hat{y}$ to the target values y, and the regularization loss L_{Reg} optimizes the parameter for the number of Markov steps. For the sake of readability, the following equation defines the loss function for one sample.

The loss function L combines fundamental concepts of E-R-GCN (Weller and Paulheim, 2021), PonderNet (Banino et al., 2021), and uncertainty quantification in neural networks (Sensoy et al., 2018). The loss L and the corresponding adjustment of the parameters of the model is computed and adjusted after each epoch and not after each step of the Markov process. The reason for this is that if the weights are adjusted after each Markov step, the rates of convergence are lower, because the network adjusts itself in each Markov step and, thus, produces volatile results. In contrast, computing the loss and adjusting the parameters of the model after each epoch is more natural and allows smoother convergence of the parameters.

$$
\begin{aligned}
L = & p_n \left\{ \sum_{k=1}^{K} \left(\underbrace{(y_k - \hat{y}_k)^2}_{L_{Ev}^{err}} + \underbrace{\frac{\hat{y}_k(1-\hat{y}_k)}{\sum_{k=1}^{K} \alpha_k + 1}}_{L_{Ev}^{var}} \right) \right. \\
& \left. + \delta_t \underbrace{KL\left(D(\tilde{\alpha}) \,||\, D(\langle 1, \ldots, 1 \rangle)\right)}_{L_{Ev}^{unc}} \right\} L_{Ev} \\
& + \beta \underbrace{KL(p_n \,||\, p_G(\lambda_p))}_{\text{Regularization loss } L_{Reg}}
\end{aligned}
\tag{10}
$$

In our loss function, L_{Ev} is the evidential loss across halting steps. Consistent with previous work in evidential learning, the evidential loss L_{Ev} consists of three components: minimizing the error of prediction $\hat{y}$ (L_{Ev}^{err}), minimizing the variance of the Dirichlet distribution to reduce uncertainty (L_{Ev}^{var}), and a regularization term which penalizes the predictive distribution, which does not contribute to data fit (L_{Ev}^{unc}). In order to ensure that the evidential loss is stable even for samples that do not follow the predicted distribution and, therefore, cannot be correctly classified but the loss still decreases towards zero, the Kullback-Leibler (KL) divergence is built into the evidential loss L_{ev}. In related work, it has been shown that using the KL divergence for out-of-distribution samples provides more stable performance in prediction (Sensoy et al., 2018; Weller and Paulheim, 2021), which is why we also use it in our loss L_{Ev} and define it as L_{Ev}^{unc} in Eq. 10. L_{Ev}^{unc} is multiplied by an annealing coefficient $\delta_t = min(1.0, t/10) \in [0,1]$. We gradually increase this coefficient within the first 10 epochs and keep it fixed afterwards to ensure that the influence of the annealing coefficient increases over the epochs but does not exceed. In this way, we prevent an early convergence to a uniform distribution for the misclassified samples and allow the network to explore the parameter space at the beginning.

In L_{Ev}^{unc}, $D(\tilde{\alpha})$ denotes the Dirichlet distribution with parameter $\tilde{\alpha}$ and $D(\langle 1, \ldots, 1 \rangle)$ denotes the uniform Dirichlet distribution. $\tilde{\alpha}$ is the adjusted evidence of the previous parameter α and is defined as follows.

$$
\tilde{\alpha} = y + (1 - y)\alpha \tag{11}
$$

The regularization term L_{Ev}^{unc} of the evidential loss with annealing coefficient δ_t, epoch t, gamma function $\Gamma(\cdot)$ and digamma function $\psi(\cdot)$ is as follows.

The second term of the loss function L (see Eq. 10) is the regularization loss L_{Reg}. L_{Reg} uses the Kullback-Leibler (KL) divergence to measure the difference between the distribution of halting probabilities p_n at step n and a prior geometric distribution denoted as $p_G(\lambda_p)$. The reason for using the regularization loss L_{Reg} is that it may improve generalization. In addition, it provides an incentive to keep the number of Markov steps performed no longer than the given distribution $p_G(\lambda_p)$.

$$
\begin{aligned}
& KL\left(D(\tilde{\alpha}) \,||\, D(\langle 1, \ldots, 1 \rangle)\right) = \\
& \qquad \log \left(\frac{\Gamma\left(\sum_{k=1}^{K} \tilde{\alpha}_k\right)}{\Gamma(K) \prod_{k=1}^{K} \Gamma(\tilde{\alpha}_k)} \right) \\
& \qquad + \sum_{k=1}^{K} (\tilde{\alpha}_k - 1) \left[\psi(\tilde{\alpha}_k) - \psi\left(\sum_{j=1}^{K} \tilde{\alpha}_j \right) \right]
\end{aligned}
\tag{12}
$$

The goal of KL in the regularization term L_{Reg} is to approximate the distribution of p_n to the geometric prior probability distribution $p_G(\lambda_p)$, which is defined by the hyperparameter λ_p. This distribution describes the probability that the model enters the absorbing state ($\Lambda_n = 1$) in step n as follows.

$$
p_G(\lambda_p) = (1 - \lambda_p)^n \lambda_p \tag{13}
$$

Using the geometric prior probability distribution $p_G(\lambda_p)$, an incentive is given to the network to approximate the number of Markov steps to the expected value of the geometric prior probability distribution $\mathbb{E}(p_G(\lambda_p)) = \frac{1}{\lambda_p}$, i.e. promotes exploration. This incentive can be controlled by the hyperparameter β and is 0.01 in our study. L_{Reg} in Eq. 10 is defined as follows.

$$
KL(p_n || p_G(\lambda_p)) = \log \left(\frac{\lambda_n}{\lambda_p} \right) + \frac{1}{\lambda_n} \log \left(\frac{1 - \lambda_n}{1 - \lambda_p} \right) \tag{14}
$$

In summary, the loss function L (see Eq. 10) thus has two functions. On the one hand, the conjugate prior, which in our case is a Dirichlet distribution, is to be fitted in such a way that the deviations between the target values y and the predictions $\hat{y}$ are minimized by L_{Ev}. And on the other hand, the number of Markov steps should be controlled by L_{Reg}.

4 Experiments

First, we provide the experimental configuration (§4.1). In our experimental study, we investigate the following questions: **(Q1)** How effective is MPERL on state-of-the-art benchmarks? (§4.2) **(Q2)** What are the effects of the hyperparameter λ_p of MPERL? (§4.3) **(Q3)** What is the impact of the MPERL com-

Table 1: Dataset statistics

Datasets	Entities	Relations	Triples	Labelled	Classes
AIFB	8,285	45	29,043	176	4
MUTAG	23,644	23	74,227	340	2
BGS	333,845	103	916,199	146	2
AM	1,666,764	133	5,988,321	1,000	11
FB15kET	14,951	1,345	483,142	168,313	3,584
YAGO43kET	42,335	37	331,686	462,083	45,182

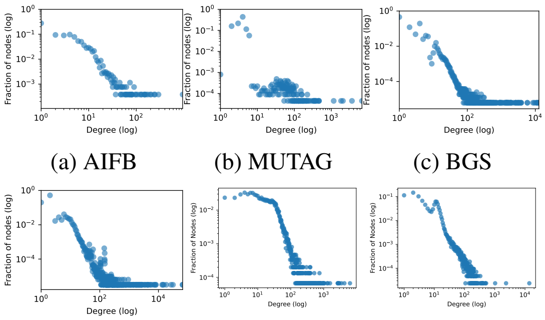
ponents on the performance? (§4.4) The source code and the datasets are available online[1].

4.1 Experimental Setup

Datasets. The evaluation is performed using the standard SOTA datasets used for entity classification, i.e., AIFB, MUTAG, BGS, and AM for evaluation (Ristoski et al., 2016). In AIFB, the class affiliation is modeled by the relation `employs` and `affiliation`, MUTAG by `isMutagenic`, BGS by `hasLithogenesis`, and AM by `material`. The triples containing these relations have been removed from training. We use predefined train/test splits, which are provided with the datasets. In addition to these benchmark datasets we consider two additional larger benchmarks derived from real-world knowledge graphs, i.e., FB15kET (Bordes et al., 2013b) and YAGO43kET (Moon et al., 2017). We follow the proposed train/valid/test split. The dataset statistics are summarized in Table 1, and the degree of distribution among the entities in the graphs is shown in Figure 2.

Metrics. We report on the accuracy and F1-macro score for the four smaller datasets. For the larger datasets, we use two ranking-based metrics in a filtered setting[2]: Mean Reciprocal Rank (MRR) and proportion of correct entity types predicted in top k (HIT@k, $k = 1, 3, 10$). Each experiment is run ten times, and we present the average performance over the test splits.

Approaches. We implemented MPERL using R-GCN as the GCN-based model in the architecture. Therefore, our experiments report on MPERL+R-GCN as our approach. The baselines used in the experiments include well-known models for entity classification in KGs. For the four smaller datasets, these baselines comprise both GCN-based models, such as R-GCN, E-R-GCN and CompGCN, and embedding-based models including RDF2Vec, ConnectE, and ASSET. In addition, we include Feat (Paulheim and Fümkranz, 2012), which uses hand-designed feature extractors, and WL (Shervashidze et al., 2011; de Vries and de Rooij, 2015), which uses graph kernels that count substructures in graphs. For all baselines, we used the recommended hyperparameter settings. As reported in previous work (Schlichtkrull et al., 2018; Ristoski and Paulheim, 2016; Ristoski et al., 2019), a linear SVM was used to classify the entities using RDF2Vec, WL, ConnectE, and ASSET. On the larger datasets, we also compare our approach with both GCN-based and embedding-based models. For the GCN-based models, we consider HMGCN, RACE2T, E-R-GCN, RGCN and CompGCN, with R-GCN and CompGCN both using Binary Cross Entropy (BCE) loss. The baselines for the embedding-based models consist of ETE (Moon et al., 2017), ConnectE, RDF2Vec, and ASSET. We reuse the hyperparameters for CompGCN, R-GCN, and E-R-GCN as suggested by their respective authors.



(a) AIFB (b) MUTAG (c) BGS

(d) AM (e) FB15kET (f) YAGO43kET

Figure 2: Degree distribution of entities in datasets

4.2 Accuracy Results

Results on Small Datasets. These datasets are designed for single-label classification, i.e., every entity belongs to one class. The hyperparameters used for MPERL across the different datasets can be found in the GitHub repository. The results for entity classification are shown in Table 2. We see that MPERL+R-GCN outperforms all the baseline methods. First, we analyse the performance of MPERL+R-GCN with respect to other GNN-based methods. Compared to R-GCN, MPERL+R-GCN does not use a softmax function but a probabilistic loss function consisting of two parts. As a result, our approach's performance is higher than R-GCN, as the loss used in MPERL+R-GCN captures the information loss between ground truth and predicted distribution. Even though E-R-GCN uses the concept of an evidential loss function as well, the end-to-end learning of R-GCN within a Markov process demonstrates a lower sensitivity to

[1] https://github.com/DE-TUM/MPERL

[2] Following (Bordes et al., 2013b), filtered setting means that all the known types of entity e in the training, validation, and test sets are first removed from the ranking, allowing us to obtain the exact rank of the correct type t_e among all types.

Table 2: Effectiveness results on small datasets

Metrics	**Accuracy**				**F1-Macro Score**			
Dataset	AIFB	MUTAG	BGS	AM	AIFB	MUTAG	BGS	AM
Embedding-based methods								
ConnectE	83.33	75.00	79.31	88.38	80.36	70.32	74.11	86.94
RDF2Vec	88.88	67.20	87.24	88.33	86.72	62.19	85.54	87.63
ASSET	86.11	76.47	75.86	89.39	82.28	67.96	73.46	87.20
Graph featurization methods								
Feat	55.55	77.94	72.41	66.66	51.69	75.82	70.88	64.87
WL	80.55	**80.88**	86.20	87.37	79.43	78.52	84.89	85.85
GNN-based methods								
CompGCN	86.39	66.32	75.17	33.08	82.89	64.38	73.65	13.35
E-R-GCN	95.56	74.56	76.55	89.85	93.21	69.63	73.98	88.76
R-GCN	92.22	73.97	75.86	89.14	87.51	70.82	74.11	79.01
Our approach								
MPERL +R-GCN	**97.22**	**80.88**	**89.66**	**90.40**	**96.13**	**79.26**	**88.26**	**89.07**

Table 3: Effectiveness results on large datasets

Datasets	FB15kET				YAGO43kET			
Metrics	**MRR**	**Hit@1**	**Hit@3**	**Hit@10**	**MRR**	**Hit@1**	**Hit@3**	**Hit@10**
Embedding-based methods								
ETE	50.00	38.51	55.33	71.93	23.00	13.73	26.28	42.18
ConnectE	59.00	49.55	64.32	79.92	28.00	16.01	30.85	47.92
RDF2Vec	59.94	50.83	64.75	77.68	32.74	24.17	35.94	49.76
ASSET	60.73	51.65	65.43	78.72	28.11	21.13	29.93	41.53
GNN-based methods								
HMGCN	51.03	39.12	54.83	72.42	25.01	14.19	27.33	43.68
RACE2T	64.14	55.56	68.40	81.36	**34.12**	**25.27**	**37.36**	**52.29**
E-R-GCN	64.59	56.94	69.12	80.09	29.67	22.63	32.43	43.51
R-GCN	63.50	53.74	69.00	**82.23**	31.56	23.32	34.53	47.49
Our approach								
MPERL +R-GCN	**65.74**	**58.18**	**70.32**	80.39	30.72	23.67	33.25	43.98

noisy neighbors due to the aggregation of the hidden features of each step in the Markov process, as well as a faster convergence over epochs due to the reuse of the hidden feature h_{n-1} in step n.

Compared to other methods, WL performs well, especially on MUTAG, which matches the highest accuracy achieved by MPERL. WL also performs well on BGS and AM, making it one of the stronger non-GNN methods. From the embeddings-based method, RDF2Vec performs better in terms of the F1-Macro score than other embeddings. When comparing results across datasets, the embeddings- and GNN-based perform worse for the MUTAG dataset. MUTAG is relatively smaller than other datasets (e.g., BGS and AM) in terms of number of entities, relations, and classes (cf. Table 1). The approaches' performance indicates that learning effective representations for entities in MUTAG is difficult since the connectivity of the entities is rather irregular, as shown in the degree distribution in Figure 2b. These results show that even in smaller datasets, entity classification can be challenging for state-of-the-art methods.

Results on Large Datasets. Next, we assess the performance of our studied approach on commonly used large KGs to mimic real-world scenarios, such as YAGO (Suchanek et al., 2007), Freebase (Bollacker et al., 2008), in which each entity can have multiple classes but some of which may be missing. Therefore, in this study, entity classification corresponds to a multi-label classification problem. This task is more challenging than single-label classification, as it requires handling multiple labels for each entity rather than assigning just a single type. Both benchmarks include a significantly higher number of types and labeled entities than those in single-label entity classification, leading to potential GPU memory problems during our experiments. To mitigate this problem, we restrict the maximum number of Markov steps to 2 and apply partial neighborhood sampling. This sampling strategy randomly selects a subset of neighbors for a given entity during message passing in graph neural networks, speeding up training and preventing overfitting. However, it may risk performance degradation if important featured neighbors are not sampled. In practice, we only conduct neighbor sampling during training, while all neighbors of the entity are used during inference. The graphs of large datasets are also augmented with type triples $(e, hastype, t_e)$ when training embeddings, as proposed by (Pan et al., 2021), which increases the prediction accuracy.

Table 3 shows the performance of our model and the results of the baselines for both benchmarks. For the FB15kET dataset, MPERL+R-GCN achieves competitive results and especially outperforms all baselines in terms of the Hit@1 metric, indicating its higher accuracy in the top prediction of the missing types. We observe that MPERL+R-GCN shows significant gains in prediction performance compared to its fundamental model, R-GCN, and slightly performs better than *E*-R-GCN, showing the usefulness of Markov steps in our proposed model. However, regarding YAGO43kET datasets, the performance of MPERL+R-GCN is less powerful compared to the FB15kET dataset. This discrepancy may arise due to key differences between the benchmarks. First, as shown in Figure 2, YAGO contains more higher-degree hub entities (with degrees exceeding 10^4), which distorts information aggregated from neighbors and reduces model performance, thereby decreasing the performance of our model. Second, as highlighted in Table 1, YAGO43kET contains approximately 12 times more classes than FB15kET, further amplifying the decline in performance. Additionally, due to limited GPU memory, the batch size is set to a small

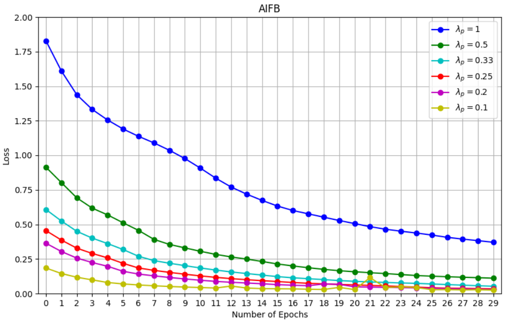


Figure 3: Learning curves for the AIFB dataset for different λ_p values

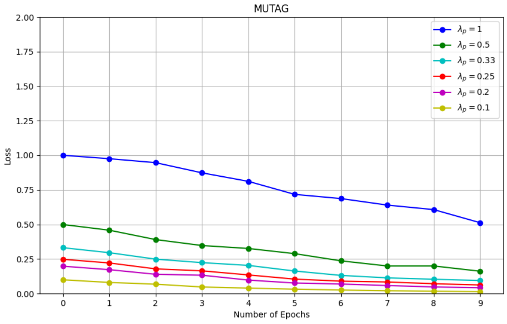


Figure 4: Learning curves for the MUTAG dataset for different λ_p values

number (16 in practice) for the YAGO43kET dataset, which may potentially lead MPERL to converge to sub-optimal solutions. Overall, MPERL+R-GCN consistently outperforms E-R-GCN in both benchmarks. This demonstrates the benefits of incorporating the Markov process, which can reduce sensitivity to noisy neighbors due to the aggregation of hidden features at each Markov step. The results also show that GNN-based methods tend to yield superior outcomes overall compared to embedding-based approaches. These observations show the potential of graph neural networks as a promising technique for addressing entity-type prediction challenges.

In summary, MPERL outperforms the state-of-the-art in small datasets. The consistently high F1-Macro scores indicate that MPERL can effectively classify entities that belong to least represented classes. In large datasets, MPERL showed very good performance in FB15kET but not in YAGO43kET. The sampling techniques implemented to scale MPERL to large datasets may have affected the learning process. These results suggest that our proposed solution is more suitable for smaller knowledge graphs, where learning meaningful representations is challenging due to the limited information contained in these datasets **(Q1)**.

4.3 Impact of the Hyperparameter λ_p

In the following, we study the impact of the hyperparameter $\lambda_p \in (0, 1]$ on the learning process of our approach MPERL on selected datasets. For this study, we chose the two smallest datasets – AIFB and MUTAG – to ensure feasibility, as the study requires training the model multiple times with different λ_p values. Furthermore, MUTAG is an interesting dataset as the performance of WL was very close to that of MPERL, providing meaningful insights. λ_p defines the geometric prior probability distribution $p_G(\lambda_p)$ (see Eq. 13), which describes the probability that the model enters the absorbing state ($\Lambda_n = 1$) in step n. The expected value of the distribution $p_G(\lambda_p)$ is $\frac{1}{\lambda_p}$. On the one hand, setting it to a low value ensures a high number of steps in the Markov process. This strategy encourages higher pondering of the model, yet also leads to increased training time due to the higher number of Markov steps and could result in higher variance. On the other hand, if λ_p is high, it would lead the model to perform only a few Markov steps. This setting leads to a reduced training time and should be used to reduce a possible variance. In the special case of $\lambda_p = 1$, one Markov step is performed, and therefore, the model corresponds to the evidence-based approach of E-R-GCN. To study this in more detail, we considered this special case and compared the learning curves between $\lambda_p = 1.0$ and smaller values of λ_p, which correspond to several Markov steps. Figures 3 and 4 show the average learning curves of ten runs on selected datasets.

On both datasets, the loss for a single Markov step ($\lambda_p = 1.0$) is initially higher than for smaller values of λ_p, including $\lambda_p = 0.5$ or two Markov steps. This is because the learned features of the previous Markov step are reused in the next one, and the prediction is based on a conjugate prior distribution, which consists of an aggregation of multiple hidden features from each Markov step. This results in much more accurate predictions and, thus, a lower loss. The learning curve for all λ_p values decreases on all datasets over the epochs. Yet, the loss of the model using $\lambda_p = 1.0$ does not drop below that of the model using smaller λ_p values. Lastly, the difference between $\lambda_p = 0.2$ and $\lambda_p = 0.1$ becomes almost negligible as the number of Epochs increases in both datasets.

Thus, we can conclude that (i) a higher number of Markov steps generally has a positive effect on the performance of the model compared to a single Markov step, and (ii) a significant decrease of the hyperparameter λ_p to 0.1 – which produces a significant increase of the number of Markov steps – does

Table 4: Ablation study on AIFB and MUTAG datasets

Dataset	AIFB		MUTAG	
Metrics	Accuracy	F1-Score	Accuracy	F1-Score
(-MP,-ERL)	94.44	95.08	66.18	43.63
(-MP,+ERL)	94.44	95.08	63.23	49.99
(+MP,-ERL)	83.33	80.12	67.65	47.69
(+MP,+ERL)	**97.22**	**96.13**	**80.88**	**79.26**

not have a major impact on the performance of the benchmark datasets used as the number of epochs increases **(Q2)**. We therefore recommend $\lambda_p = 0.2$ as the default value, representing a trade-off between accurate entity classification and reduced training time achieved with a low number of Markov steps.

4.4 Ablation Study

We conduct an ablation study to measure the impact of each component in MPERL on the performance. By systematically isolating, and evaluating each component of our approach, our ablation study aims to highlight the specific contributions and overall effectiveness of our proposed approach. The two building blocks of our approach are (i) Markov Learning Process (MP) (§3.1) and (ii) Evidential Regularized Loss function (ERL) (§3.2). When a component is not included, it is represented in the results by a '-', e.g., '-ERL' equals the usage of the cross-entropy loss function and '-MP' equals no Markov Process used. Conversely, when the component is included, it is represented as '+'. The settings for running the experiments are the ones used in §4.2. For the same reasons explained in §4.3, we selected the datasets AIFB and MUTAG. The evaluation metrics as reported in Table 4, i.e., accuracy and F1-score, were used. Including MP alone (+MP, -ERL) decreases the performance significantly for the AIFB dataset, yet for MUTAG dataset it leads to a slight increase in accuracy ($+1.47$, from 66.18 to 67.65) and in F1-Macro score ($+4.06$, from 43.63 to 47.69). These results indicate that MP alone is not so effective without ERL, and can even be detrimental in some datasets. Including ERL alone (-MP, +ERL) does not change the results for accuracy and F1-score in AIFB. In MUTAG, +ERL slightly improves the F1-score ($+6.36$, from 43.63 to 49.99) but decreases accuracy (-2.9, from 66.18 to 63.23); this suggests that +ERL may contribute to better classification of the entities for smaller classes (containing lower number of entities) in MUTAG. Still, ERL's impact is minimal without MP, and its effect depends on the dataset. Lastly, in both datasets, the results show that including both components (+MP, +ERL) improves the performance significantly and consistently to the highest values, showing that the combined effect of MP and ERL is beneficial **(Q3)**.

5 Conclusion and Future Work

In this work, we have shown how to combine Graph Convolutional Neural Networks (GCN), concepts from Evidential Learning, and PonderNet in the MPERL approach for entity classification in knowledge graphs. The model is learned within a Markov decision process, which computes halting at the current step. We implemented MPERL on top of R-GCN, a state-of-the-art GCN-based architecture. The experimental results show a performance increase in most datasets, particularly compared to the previous GCN-based models. The aggregation of the hidden features of the individual steps in the Markov process demonstrated a lower sensitivity to noisy neighbors and improved performance. The hyperparameter λ_p defines the geometric prior probability distribution and, thus, approximates the number of Markov steps. A low λ_p setting leads to a high number of Markov steps and, therefore, to a high training time, which has a positive effect on the performance compared to a model that was learned within a Markov process with only one step. The experiments have shown that $\lambda_p = 0.2$ (i.e., five Markov steps) has already led to improved performance. The reuse of the learned hidden features of the previous hidden features of the Markov process has allowed faster convergence of parameters during learning.

As future work, we want to further investigate the hyperparameter λ_p, by employing an automated approach to select the optimal values. To improve the scalability of our model in multi-label settings, we also want to replace the Dirichlet distribution with a more suitable alternative, such as the Beta distribution (Zhao et al., 2023). Lastly, we want to extend MPERL with a gated unit that automatically adjusts the weight of the hidden representation from the previous Markov step (h_{n-1}) as input during learning.

Acknowledgements: This work has been funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) - SFB 1608 - 501798263.

REFERENCES

Amini, A., Schwarting, W., Soleimany, A., and Rus, D. (2020). Deep evidential regression. In Advances in Neural Information Processing

Systems, volume 33, pages 14927–14937. Curran Associates, Inc.

Banino, A., Balaguer, J., and Blundell, C. (2021). Pondernet: Learning to ponder.

Berrendorf, M., Faerman, E., Melnychuk, V., Tresp, V., and Seidl, T. (2020). Knowledge graph entity alignment with graph convolutional networks: Lessons learned. In Advances in Information Retrieval, pages 3–11, Cham. Springer International Publishing.

Biswas, R., Portisch, J., Paulheim, H., Sack, H., and Alam, M. (2022). Entity type prediction leveraging graph walks and entity descriptions. In The Semantic Web - ISWC 2022 - 21st International Semantic Web Conference, Proceedings, volume 13489 of Lecture Notes in Computer Science, pages 392–410.

Biswas, R., Sofronova, R., Sack, H., and Alam, M. (2021). Cat2type: Wikipedia category embeddings for entity typing in knowledge graphs. In Proceedings of the 11th on Knowledge Capture Conference, K-CAP, page 81–88. Association for Computing Machinery.

Biswas, R., Türker, R., Moghaddam, F. B., Koutraki, M., and Sack, H. (2018). Wikipedia infobox type prediction using embeddings. In DL4KGS@ESWC.

Bollacker, K., Evans, C., Paritosh, P., Sturge, T., and Taylor, J. (2008). 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.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013a). Translating embeddings for modeling multi-relational data. In Advances in Neural Information Processing Systems, volume 26. Curran Associates, Inc.

Bordes, A., Usunier, N., Garcia-Duran, A., Weston, J., and Yakhnenko, O. (2013b). Translating embeddings for modeling multi-relational data. Advances in neural information processing systems, 26.

Busbridge, D., Sherburn, D., Cavallo, P., and Hammerla, N. Y. (2019). Relational graph attention networks.

Chen, Y., Zou, L., and Qin, Z. (2019). Gated relational graph neural network for semi-supervised learning on knowledge graphs. In Web Information Systems Engineering – WISE 2019, pages 617–629. Springer International Publishing.

de Vries, G. K. D. and de Rooij, S. (2015). Substructure counting graph kernels for machine learning from rdf data. Journal of Web Semantics, 35:71–84.

Dong, T., Wang, Z., Li, J., Bauckhage, C., and Cremers, A. B. (2019). Triple classification using regions and fine-grained entity typing. Proceedings of the AAAI Conference on Artificial Intelligence, 33(01):77–85.

Hamilton, W. L., Ying, R., and Leskovec, J. (2017). Inductive representation learning on large graphs. In Proceedings of the 31st International Conference on Neural Information Processing Systems, NIPS, page 1025–1035.

Heist, N., Hertling, S., Ringler, D., and Paulheim, H. (2020). Knowledge graphs on the web–an overview. In Knowledge Graphs for eXplainable Artificial Intelligence: Foundations, Applications and Challenges, pages 3–22. IOS Press.

Kejriwal, M. and Szekely, P. A. (2017). Supervised typing of big graphs using semantic embeddings. CoRR, abs/1703.07805.

Kipf, T. N. and Welling, M. (2016). Semi-supervised classification with graph convolutional networks. arXiv preprint arXiv:1609.02907.

Long, J., Wang, Y., Wei, X., Ding, Z., Qi, Q., Xie, F., Qian, Z., and Huang, W. (2021). Entity-centric fully connected GCN for relation classification. Applied Sciences, 11(4):1377.

Moon, C., Jones, P., and Samatova, N. F. (2017). Learning entity type embeddings for knowledge graph completion. In Proceedings of the 2017 ACM on conference on information and knowledge management, pages 2215–2218.

Nickel, M., Tresp, V., and Kriegel, H.-P. (2011). A three-way model for collective learning on multi-relational data. In Proceedings of the 28th International Conference on International Conference on Machine Learning, ICML, page 809–816.

Pan, W., Wei, W., and Mao, X.-L. (2021). Context-aware entity typing in knowledge graphs. In Findings of the Association for Computational Linguistics: EMNLP 2021, pages 2240–2250.

Paulheim, H. and Fümkranz, J. (2012). Unsupervised generation of data mining features from linked open data. In Proceedings of the 2nd International Conference on Web Intelligence, Mining and Semantics.

Riaz, A., Abdollahi, S., and Gottschalk, S. (2023). Entity typing with triples using language models. In The Semantic Web: ESWC 2023

Satellite Events - Hersonissos, Crete, Greece, Proceedings, volume 13998 of Lecture Notes in Computer Science, pages 169–173. Springer.

Ristoski, P., De Vries, G. K. D., and Paulheim, H. (2016). A collection of benchmark datasets for systematic evaluations of machine learning on the semantic web. In The Semantic Web–ISWC 2016: 15th International Semantic Web Conference, Kobe, Japan, October 17–21, 2016, Proceedings, Part II 15.

Ristoski, P. and Paulheim, H. (2016). Rdf2vec: Rdf graph embeddings for data mining. In The Semantic Web – ISWC 2016. Springer International Publishing.

Ristoski, P., Rosati, J., Di Noia, T., De Leone, R., and Paulheim, H. (2019). Rdf2vec: Rdf graph embeddings and their applications. Semantic Web, 10(4):721–752.

Schlichtkrull, M., Kipf, T. N., Bloem, P., van den Berg, R., Titov, I., and Welling, M. (2018). Modeling relational data with graph convolutional networks. In The Semantic Web, pages 593–607, Cham. Springer International Publishing.

Sensoy, M., Kaplan, L., and Kandemir, M. (2018). Evidential deep learning to quantify classification uncertainty. In Advances in Neural Information Processing Systems, volume 31.

Shervashidze, N., Schweitzer, P., van Leeuwen, E. J., Mehlhorn, K., and Borgwardt, K. M. (2011). Weisfeiler-lehman graph kernels. Journal of Machine Learning Research, 12(77):2539–2561.

Sofronova, R., Alam, M., and Sack, H. (2020). Entity typing based on rdf2vec using supervised and unsupervised methods.

Suchanek, F. M., Kasneci, G., and Weikum, G. (2007). Yago: a core of semantic knowledge. In Proceedings of the 16th international conference on World Wide Web, pages 697–706.

Wang, H., Zhao, M., Xie, X., Li, W., and Guo, M. (2019). Knowledge graph convolutional networks for recommender systems. In The World Wide Web Conference, page 3307–3313.

Wang, P., Zhou, J., Liu, Y., and Zhou, X. (2021). Transet: Knowledge graph embedding with entity types. Electronics, 10:1407.

Wang, Z., Lv, Q., Lan, X., and Zhang, Y. (2018). Cross-lingual knowledge graph alignment via graph convolutional networks. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing.

Weller, T. and Acosta, M. (2021). Predicting instance type assertions in knowledge graphs using stochastic neural networks. In Proceedings of the 30th ACM International Conference on Information and Knowledge Management, page 2111–2118.

Weller, T. and Paulheim, H. (2021). Evidential relational-graph convolutional networks for entity classification in knowledge graphs. In CIKM '21: The 30th ACM International Conference on Information and Knowledge Management, pages 3533–3537. ACM.

Xiong, C. and Callan, J. (2015). Esdrank: Connecting query and documents through external semi-structured data. In Proceedings of the 24th ACM International on Conference on Information and Knowledge Management, page 951–960. Association for Computing Machinery.

Yang, B., Yih, W., He, X., Gao, J., and Deng, L. (2015). Embedding entities and relations for learning and inference in knowledge bases. In 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.

Yasunaga, M., Ren, H., Bosselut, A., Liang, P., and Leskovec, J. (2021). Qa-gnn: Reasoning with language models and knowledge graphs for question answering. In North American Chapter of the Association for Computational Linguistics (NAACL).

Zahera, H. M., Heindorf, S., and Ngomo, A. N. (2021). ASSET: A semi-supervised approach for entity typing in knowledge graphs. In K-CAP '21: Knowledge Capture Conference, Virtual Event, USA, December 2-3, 2021, pages 261–264. ACM.

Zangari, L., Interdonato, R., Calió, A., and Tagarelli, A. (2021). Graph convolutional and attention models for entity classification in multilayer networks. Applied Network Science, 6(1):1–36.

Zhao, C., Du, D., Hoogs, A., and Funk, C. (2023). Open set action recognition via multi-label evidential learning. In Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, pages 22982–22991.

Zhao, Y., zhang, a., Xie, R., Liu, K., and WANG, X. (2020). Connecting embeddings for knowledge graph entity typing. In Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics.

Zou, C., An, J., and Li, G. (2022). Knowledge graph entity type prediction with relational aggregation graph attention network. In European Semantic Web Conference, pages 39–55. Springer.