Explaining Neural Matrix Factorization with Gradient Rollback

Carolin Lawrence, Timo Sztyler, Mathias Niepert
NEC Laboratories Europe, Kurfürsten-Anlage 36, 69115 Heidelberg, Germany
{carolin.lawrence, timo.sztyler, mathias.niepert}@neclab.eu

Abstract

Explaining the predictions of neural black-box models is an important problem, especially when such models are used in applications where user trust is crucial. Estimating the influence of training examples on a learned neural model's behavior allows us to identify training examples most responsible for a given prediction and, therefore, to faithfully explain the output of a black-box model. The most generally applicable existing method is based on influence functions, which scale poorly for larger sample sizes and models.

We propose *gradient rollback*, a general approach for influence estimation, applicable to neural models where each parameter update step during gradient descent touches a smaller number of parameters, even if the overall number of parameters is large. Neural matrix factorization models trained with gradient descent are part of this model class. These models are popular and have found a wide range of applications in industry. Especially knowledge graph embedding methods, which belong to this class, are used extensively. We show that gradient rollback is highly efficient at both training and test time. Moreover, we show theoretically that the difference between gradient rollback's influence approximation and the true influence on a model's behavior is smaller than known bounds on the stability of stochastic gradient descent. This establishes that gradient rollback is robustly estimating example influence. We also conduct experiments which show that gradient rollback provides faithful explanations for knowledge base completion and recommender datasets. An implementation[1] and an appendix[2] are available.

Introduction

Estimating the influence a training sample (or a set of training samples) has on the behavior of a machine learning model is a problem with several useful applications. First, it can be used to interpret the behavior of the model by providing an explanation for its output in form of a set of training samples that impacted the output the most. In addition to providing a better understanding of model behavior, influence estimation has also been used to find adversarial examples, to uncover domain mismatch, and to determine incorrect or mislabeled examples (Koh and Liang 2017). Finally, it can also be used to estimate the uncertainty for a particular output by exploring the stability of the output probability before and after removing a small number of influential training samples.

We propose *gradient rollback* (GR), a novel approach for influence estimation. GR is applicable to neural models trained with gradient descent and is highly efficient especially when the number of parameters that are significantly changed during any update step is moderately sized. This is the case for neural matrix factorization methods. Here, we focus on neural link prediction models for multi-relational graphs (also known as knowledge base embedding models), as these models subsume several other matrix factorization models (Guo et al. 2020). They have found a wide range of industry applications such as in recommender and question answering systems. They are, however, black-box models whose predictions are not inherently interpretable. Other methods, such as rule-based methods, might be more interpretable, but typically have worse performance. Hence, neural matrix factorization methods would greatly benefit from being more interpretable (Bianchi et al. 2020). For an illustration of matrix factorization and GR see Figure 1.

We explore two crucial questions regarding the utility of GR. First, we show that GR is highly efficient at both training and test time, even for large datasets and models. Second, we show that its influence approximation error is smaller than known bounds on the stability of stochastic gradient descent for non-convex problems (Hardt, Recht, and Singer 2016). The stability of an ML model is defined as the maximum change of its output one can expect on any sample when retrained on a slightly different set of training samples. The relationships between uniform stability and generalization of a learning system is a seminal result (Bousquet and Elisseeff 2002). Here, we establish a close connection between the stability of a learning system and the challenge of estimating training sample influence and, therefore, explaining the model's behavior. Intuitively, the more stable a model, the more likely is it that we can estimate the influence of training samples well. We show theoretically that the difference between GR's influence approximation and the true influence on the model behavior is (strictly) smaller than known bounds on the stability of stochastic gradient descent.

We perform experiments on standard matrix factorization datasets including those for knowledge base completion and recommender systems. Concretely, GR can explain a pre-

[1]https://github.com/carolinlawrence/gradient-rollback

[2]https://arxiv.org/abs/2010.05516

diction of a learned model by producing a ranked list of training examples, where each instance of the list contributed to changing the likelihood of the prediction and the list is sorted from highest to lowest impact. To produce this list, we (1) estimate the influence of training examples during training and (2) use this estimation to determine the contribution each training example made to a particular prediction. To evaluate whether GR selected training examples relevant for a particular prediction, we remove the set of training examples, retrain the model from scratch and check if the likelihood for the particular prediction decreased. Compared to baselines, GR can identify subsets of training instances that are highly influential to the model's behavior. These can be represented as a graph to explain a prediction to a user.

Neural Matrix Factorization

We focus on neural matrix factorization models for link prediction in multi-relational graphs (also known as knowledge graph embedding models) for two reasons. First, these models are popular with a growing body of literature. There has been increasing interest in methods for explaining and debugging knowledge graph embedding methods (Bianchi et al. 2020). Second, matrix factorization methods for recommender systems can be seen as instances of neural matrix factorization where one uses pairs of entities instead of (entity, relation type, entity)-triples (Guo et al. 2020).

We consider the following general setting of representation learning for multi-relational graphs[3]. There is a set of entities $\mathcal{E}$ and a set of relation types $\mathcal{R}$. A knowledge base $\mathcal{K}$ consists of a set of triples $d = (s, r, o)$ where $s \in \mathcal{E}$ is the subject (or head entity), $r \in \mathcal{R}$ the relation type, and $o \in \mathcal{E}$ the object (or tail entity) of the triple d. For each entity and relation type we *learn* a vector representation with hidden dimension h.[4] For each entity e and relation type r we write $\mathbf{e}$ and $\mathbf{r}$, respectively, for their vector representations. Hence, the set of parameters of a knowledge base embedding model consists of one matrix $\mathbf{E} \in \mathbb{R}^{|\mathcal{E}|\times h}$ and one matrix $\mathbf{R} \in \mathbb{R}^{|\mathcal{R}|\times h}$. The rows of matrix $\mathbf{E}$ are the entity vector representations (embeddings) and the rows of $\mathbf{R}$ are the relation type vector representations (embeddings). To refer to the set of all parameters we often use the term $\boldsymbol{w} \in \Omega$ to improve readability, with Ω denoting the parameter space. Hence, $\mathbf{e} = w[e]$, that is, the embedding of entity e is the part of $\boldsymbol{w}$ pertaining to entity e. Analogously for the relation type embeddings. Moreover, for every triple $d = (s, r, o)$, we write $\boldsymbol{w}[d] = (\boldsymbol{w}[s], \boldsymbol{w}[r], \boldsymbol{w}[o]) = (\mathbf{s}, \mathbf{r}, \mathbf{o})$ to denote the triple of parameter vectors associated with d for $\boldsymbol{w} \in \Omega$. We extend element-wise addition and multiplication to triples of vectors in the obvious way.

We can now define a *scoring function* $\phi(\boldsymbol{w}; d)$ which, given the current set of parameters $\boldsymbol{w}$, maps each triple d to a real-valued score. Note that, for a given triple d, the set of parameters involved in computing the value of $\phi(\boldsymbol{w}; d)$ are a small subset of $\boldsymbol{w}$. Usually, ϕ is a function of the vector representations of entities and relation types of the triple $d = (s, r, o)$, and we write $\phi(\mathbf{s}, \mathbf{r}, \mathbf{o})$ to make this explicit.

Example 1. *Given a triple $d = (s, r, o)$. The scoring function of* DISTMULT *is defined as $\phi(\boldsymbol{w}; d) = \langle \mathbf{s}, \mathbf{r}, \mathbf{o} \rangle$, that is, the inner product of the three embeddings.*

To train a neural matrix factorization model means running an iterative learning algorithm to find a set of parameter values $\boldsymbol{w}$ that minimize a loss function $\mathcal{L}(\boldsymbol{w}; \mathcal{D})$, which combines the scores of a set of training triples $\mathcal{D}$ into a loss-value. We consider general iterative update rules such as stochastic gradient descent (SGD) of the form $G : \Omega \to \Omega$ which map a point $\boldsymbol{w}$ in parameter space Ω to another point $G(\boldsymbol{w}) \in \Omega$. A typical loss function is $\mathcal{L}(\boldsymbol{w}; \mathcal{D}) = \sum_{d \in \mathcal{D}} \ell(\boldsymbol{w}; d)$ with

$$\ell(\boldsymbol{w}; d = (s, r, o)) = -\log \Pr(\boldsymbol{w}; o \mid s, r) \quad \textbf{and} \tag{1}$$

$$\Pr(\boldsymbol{w}; o \mid s, r) = \frac{\exp(\phi(\boldsymbol{w}; (s, r, o)))}{\sum_{o'} \exp\left(\phi(\boldsymbol{w}; (s, r, o'))\right)}. \tag{2}$$

The set of o' in the denominator is mostly a set of randomly sampled entities (negative sampling) but recent results have shown that computing the softmax over all entities can be advantageous. Finally, once a model is trained, Equation 2 can be used to determine for a test query $(s, r, ?)$ how likely each entity o is by summing over all other entities o'.

Gradient Rollback for Matrix Factorization

A natural way of explaining the model's output for a test triple d is to find the training triple d' such that retraining the model without d' changes (decreases **or** increases) $f(\boldsymbol{w}'; d)$ the most. The function f can be the loss function. Most often, however, it is the scoring function as a proxy of the loss.

$$\begin{aligned} d_{\texttt{expl}} &= \underset{d' \in \mathcal{D}}{\arg\max}\, [f(\boldsymbol{w}; d) - f(\boldsymbol{w}'; d)] \quad \textbf{or} \\ d_{\texttt{expl}} &= \underset{d' \in \mathcal{D}}{\arg\min}\, [f(\boldsymbol{w}; d) - f(\boldsymbol{w}'; d)], \end{aligned} \tag{3}$$

where $\boldsymbol{w}'$ are the parameters after retraining the model with $\mathcal{D} - \{d'\}$. This can directly be extended to obtaining the top-k most influential triples. While this is an intuitive approach, solving the optimization problems above is too expensive in practice as it requires retraining the model $|\mathcal{D}| = n$ times for each triple one seeks to explain.[5]

Instead of retraining the model, we propose gradient rollback (GR), an approach for tracking the influence each training triple has on a model's parameters during training. Before training and for each triple d' and the parameters it can influence $\boldsymbol{w}(d')$, we initialize its influence values with zero: $\forall d', \gamma_{[d', \boldsymbol{w}(d')]} = \mathbf{0}$. We now record the changes in parameter values during training that each triple causes. With stochastic gradient descent (SGD), for instance, we iterate through the training triples $d' \in \mathcal{D}$ and record the changes it makes at

[3]Our results with respect to 3-dimensional matrices apply to k-dimensional matrices with $k \geq 2$.

[4]Our results generalize to methods where relation types are represented with more than one vector such as in RESCAL (Nickel, Tresp, and Kriegel 2011) or COMPLEX (Trouillon et al. 2016).

[5]For example, FB15K-237 contains $270k$ training triples and training one model with TF2 on a RTX 2080 Ti GPU takes about 15 minutes. To explain one triple by retraining $|\mathcal{D}| = 270k$ times would take over 2 months.

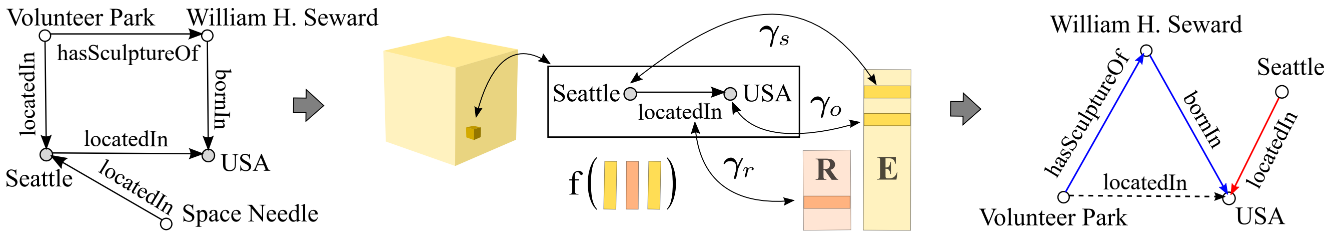

Figure 1: An illustration of the problem and the proposed method. A knowledge base of facts is represented as a sparse 3-dimensional matrix. Neural matrix factorization methods perform an implicit matrix decomposition by minimizing a loss function f operating on the entity and relation representations. Gradient rollback tracks the parameter changes γ caused by each sample during training. At test time, the aggregated updates are used to estimate triple influence without the need to retrain the model. Influence estimates are used to provide triples (blue and red) explaining the model's behavior for a test triple (dashed).

time t, by setting its influence value to

$$\gamma_{[d',\boldsymbol{w}_{t+1}(d')]} \leftarrow \gamma_{[d',\boldsymbol{w}_t(d')]} - \alpha\nabla f(\boldsymbol{w}_t; d').^6$$

After training, γ can be utilized as a look-up table: for a triple d that we want to explain, we look up the influence each triple d' had on the weights $\boldsymbol{w}(d)$ of d via $\gamma_{[d',\boldsymbol{w}(d)]} = (\gamma_s, \gamma_r, \gamma_o)$. Concretely, a triple d has an influence on d' at any point where the two triples have a common element wrt. their subject, relation or object. Consequently, explanations can be any triple d' that has either an entity or the relation in common with d.

Let $\boldsymbol{w}'$ be the parameters of the model when trained on $\mathcal{D} - \{d'\}$. At prediction time, we now approximate, for every test triple d, the difference $f(\boldsymbol{w}; d) - f(\boldsymbol{w}'; d)$ with $f(\boldsymbol{w}; d) - f(\boldsymbol{w} - \gamma_{[d',\boldsymbol{w}(d)]}; d)$, that is, by *rolling back* the influence triple d' had on the parameters of triple d during training. We then simply choose the triple d' that maximizes this difference.

There are two crucial questions determining the utility of GR, which we address in the following:

1. The resource overhead of GR during training and at test time; and
2. How closely $f(\boldsymbol{w} - \gamma_{[d',\boldsymbol{w}(d)]}; d)$ approximates $f(\boldsymbol{w}'; d)$.

Resource Overhead of Gradient Rollback

To address the first question, let us consider the computational and memory overhead for maintaining $\gamma_{[d,\boldsymbol{w}_t(d)]}$ during training for each d. The loss function's normalization term is (in expectation) constant for all triples and can be ignored. We verified this empirically and it is indeed a reasonable assumption. Modern deep learning frameworks compute the gradients and make them accessible in each update step. Updating $\gamma_{[d,\boldsymbol{w}_t(d)]}$ with the given gradients takes $O(h)$, that is, it is possible in constant time. Hence, computationally, the overhead is minimal. Now, to store $\gamma_{[d,\boldsymbol{w}_t(d)]}$ for every $d \in \mathcal{D}$ we need $h|\mathcal{D}| + 2h|\mathcal{D}|$ floats. Hence, the memory overhead of gradient rollback is $3h|\mathcal{D}|$. This is about the same size as the parameters of the link prediction model itself. In a nutshell:

Gradient rollback has a $O(h|\mathcal{D}|)$ computational and $3h|\mathcal{D}|$ memory overhead during training.

At test time, we want to understand the computational complexity of computing

$$d_{\texttt{expl}} = \underset{d' \in \mathcal{D}}{\arg\max}\, f(\boldsymbol{w}; d) - f(\boldsymbol{w} - \gamma_{[d',\boldsymbol{w}(d)]}; d) \quad \textbf{or}$$
$$d_{\texttt{expl}} = \underset{d' \in \mathcal{D}}{\arg\min}\, f(\boldsymbol{w}; d) - f(\boldsymbol{w} - \gamma_{[d',\boldsymbol{w}(d)]}; d) \tag{4}$$

We have $\gamma_{[d',\boldsymbol{w}(d)]} \neq \mathbf{0}$ only if d and d' have at least one entity or relation type in common. Hence, to explain a triple d we only have to consider triples d' adjacent to d in the knowledge graph, that is, triples where either of the arguments overlap. Let adj_{max} and adj_{avg} be the maximum and average number of adjacent triples in the knowledge graph. Then, we have the following:

Gradient rollback requires at most $\text{adj}_{\text{max}} + 1$ and on average $\text{adj}_{\text{avg}} + 1$ computations of the function f to explain a test triple d.

Approximation Error of Gradient Rollback

To address the second question above, we need, for every pair of triples d, d', to bound the expression

$$\mathbb{E}\,|f(\boldsymbol{w} - \gamma_{(d',\boldsymbol{w}(d))}; d) - f(\boldsymbol{w}'; d)|,$$

where $\boldsymbol{w}$ are the parameter values resulting from training f on all triples $\mathcal{D}$ and $\boldsymbol{w}'$ are the parameter values resulting from training f on $\mathcal{D} - \{d'\}$. If the above expression can be bounded and said bound is lower than what one would expect due to randomness of the iterative learning dynamics, the proposed gradient rollback approach would be highly useful. We use the standard notion of stability of learning algorithms (Hardt, Recht, and Singer 2016). In a nutshell, our theoretical results will establish that:

Gradient rollback can approximate, for any $d' \in \mathcal{D}$, the changes of a scoring/loss function one would observe if a model were to be retrained on $\mathcal{D} - \{d'\}$. The approximation error is in expectation lower than known bounds on the stability of stochastic gradient descent.

[6] For any iterative optimizer such as SGD and Adam, the parameter updates are readily available and can be obtained efficiently during training.

Definition 1. *A function f is L-Lipschitz if for all u, v in the domain of f we have $\|\nabla f(u)\| \leq L$. This implies that $|f(u) - f(v)| \leq L \|u - v\|$.*

We analyze the output of stochastic gradient descent on two data sets, $\mathcal{D}$ and $\mathcal{D} - \{d'\}$, that differ in precisely one triple. If f is L-Lipschitz for every example d, we have

$$\mathbb{E}\, |f(\boldsymbol{w}; d) - f(\boldsymbol{w}'; d)| \leq L\, \mathbb{E}\, \|\boldsymbol{w} - \boldsymbol{w}'\|$$

for all $\boldsymbol{w}$ and $\boldsymbol{w}'$. A vector norm in this paper is always the 2-norm. Hence, we have $\mathbb{E}\, |f(\boldsymbol{w} - \boldsymbol{\gamma}_{(d', \boldsymbol{w}(d))}; d) - f(\boldsymbol{w}'; d)| \leq L\, \mathbb{E} \left\|\boldsymbol{w} - \boldsymbol{\gamma}_{(d', \boldsymbol{w}(d))} - \boldsymbol{w}'\right\|$ and we can assess the approximation error by tracking the extent to which the parameter values $\boldsymbol{w}$ and $\boldsymbol{w}'$ from two coupled iterative learning dynamics diverge over time. Before we proceed, however, let us formally define some concepts and show that they apply to typical scoring functions of knowledge base embedding methods.

Definition 2. *A function $f : \Omega \rightarrow \mathbb{R}$ is β-smooth if for all u, v in the domain of f we have $\|\nabla f(u) - \nabla f(v)\| \leq \beta \|u - v\|$.*

To prove Lipschitz and β-smoothness properties of a function $f(\boldsymbol{w}; d)$ for all $\boldsymbol{w} \in \Omega$ and $d \in \mathcal{D}$, we henceforth assume that the norm of the entity and relation type embeddings is bounded by a constant $C > 0$. That is, we assume $\max_{r \in \mathcal{R}} \|\boldsymbol{w}[r]\| \leq C$ and $\max_{e \in \mathcal{E}} \|\boldsymbol{w}[e]\| \leq C$ for all $\boldsymbol{w} \in \Omega$. This is a reasonable assumption for two reasons. First, several regularization techniques constrain the norm of embedding vectors. For instance, the unit norm constraint, which was used in the original DistMult paper (Yang et al. 2015), enforces that $C = 1$. Second, even in the absence of such constraints we can assume a bound on the embedding vectors' norms as we are interested in the approximation error for and the stability of a given model that was obtained from running SGD a *finite* number of steps using the *same* parameter initializations. When running SGD, for $\mathcal{D}$ and all $\mathcal{D} - \{d'\}$, a finite number of steps with the same initialization, the encountered parameters $\boldsymbol{w}$ and $\boldsymbol{w}'$ in each step of each run of SGD form a finite set. Hence, for our purposes, we can assume that f's domain Ω is compact. Given this assumption, we show that the inner product, which is used in several scoring functions, is L-Lipschitz and β-smooth on Ω. The proofs of all lemmas and theorems can be found in the appendix.

Lemma 1. *Let $f : \mathbb{R}^k \times \mathbb{R}^l \times \mathbb{R}^m \rightarrow \mathbb{R}^n : (x, y, z) \rightarrow f(x, y, z)$ be Lipschitz continuous relative to, respectively, x, y, and z. Then f is Lipschitz continuous as a function $\mathbb{R}^{k+l+m} \rightarrow \mathbb{R}^n$. More specifically, $|f(u_x, u_y, u_z) - f(v_x, v_y, v_z)| \leq 2 \max\{L_x, L_y, L_z\} \|(u_x, u_y, u_z) - (v_x, v_y, u_z)\|$.*

Lemma 2. *Let ϕ be the scoring function of DistMult defined as $\phi(\boldsymbol{w}; d = (s, r, o)) = \langle \mathbf{s}, \mathbf{r}, \mathbf{o} \rangle$ with $\boldsymbol{w}(d) = (\mathbf{s}, \mathbf{r}, \mathbf{o})$, and let C be the bound on the norm of the embedding vectors for all $\boldsymbol{w} \in \Omega$. For a given triple $d = (s, r, o)$ and all $\boldsymbol{w}, \boldsymbol{w}' \in \Omega$, we have that*

$$|\phi(\boldsymbol{w}; d) - \phi(\boldsymbol{w}'; d)| \leq 2C^2 \|\boldsymbol{w} - \boldsymbol{w}'\|.$$

Lemma 3. *Let ϕ be the scoring function of DistMult defined as $\phi(\boldsymbol{w}; d = (s, r, o)) = \langle \mathbf{s}, \mathbf{r}, \mathbf{o} \rangle$ with $\boldsymbol{w}(d) = (\mathbf{s}, \mathbf{r}, \mathbf{o})$, and let C be the bound on the norm of the embedding vectors for all $\boldsymbol{w} \in \Omega$. For a given triple $d = (s, r, o)$ and all $\boldsymbol{w}, \boldsymbol{w}' \in \Omega$, we have that*

$$\|\nabla \phi(\boldsymbol{w}; d) - \nabla \phi(\boldsymbol{w}', d)\| \leq 4C \|\boldsymbol{w} - \boldsymbol{w}'\|.$$

Considering typical KG embedding loss functions and the softmax and sigmoid function being 1-Lipschitz, this implies that the following theoretical analysis of *gradient rollback* applies to a large class of neural matrix factorization models. Let us first define an additional property iterative learning methods can exhibit.

Definition 3. *An update rule $G : \Omega \rightarrow \Omega$ is η-expansive if*

$$\sup_{\mathbf{u}, \mathbf{v} \in \Omega} \frac{\|G(\mathbf{u}) - G(\mathbf{v})\|}{\|\mathbf{u} - \mathbf{v}\|} \leq \eta.$$

Consider the gradient updates $G_1, ..., G_T$ and $G'_1, ..., G'_T$ induced by running stochastic gradient descent on $\mathcal{D}$ and $\mathcal{D} - \{d'\}$, respectively. Every gradient update changes the parameters $\boldsymbol{w}$ and $\boldsymbol{w}'$ of the two coupled models. Due to the difference in size, there is one gradient update G_i whose corresponding update G'_i does not change the parameters $\boldsymbol{w}'$. Again, note that there is always a finite set of parameters $\boldsymbol{w}$ and $\boldsymbol{w}'$ encountered during training.

We can derive a stability bound for stochastic gradient descent run on the two sets, $\mathcal{D}$ and $\mathcal{D} - \{d'\}$. The following theorem and its proof are an adaptation of Theorem 3.21 in Hardt, Recht, and Singer (2016) and the corresponding proof. There are two changes compared to the original theorem. First, f is not assumed to be a loss function but *any* function that is L-Lipschitz and β-smooth. For instance, it could be that the loss function of the model is defined as $g(f(\cdot))$ for some Lipschitz and smooth function g. The proof of the original theorem does not rely on f being a loss function and it only requires that $f(\cdot; d) \in [0, 1]$ is an L-Lipschitz and β-smooth function for all inputs. Second, the original proof assumed two training datasets of the same size that differ in one of the samples. We assume two sets, $\mathcal{D}$ and $\mathcal{D} - \{d'\}$, where $\mathcal{D} - \{d'\}$ has exactly one sample less than $\mathcal{D}$. The proof of the theorem is adapted to this setting.

Theorem 1. *Let $f(\cdot; d) \in [0, 1]$ be an L-Lipschitz and β-smooth function for every possible triple d and let c be the initial learning rate. Suppose we run SGD for T steps with monotonically non-increasing step sizes $\alpha_t \leq c/t$ on two different sets of triples $\mathcal{D}$ and $\mathcal{D} - \{d'\}$. Then, for any d,*

$$\mathbb{E}\, |f(\boldsymbol{w}_T; d) - f(\boldsymbol{w}'_T; d)| \leq \frac{1 + 1/\beta c}{n - 1} (cL^2)^{\frac{1}{\beta c + 1}} T^{\frac{\beta c}{\beta c + 1}},$$

with $\boldsymbol{w}_T$ and $\boldsymbol{w}'_T$ the parameters of the two models after running SGD. We name the right term in the above inequality $\Lambda_{stab\text{-}nc}$.

The assumption $f(\cdot; d) \in [0, 1]$ of the theorem is fulfilled if we use the loss from Equation 1, as long as we are interested in the stability of the probability distribution of Equation 2. That is because the cross-entropy loss applied to a softmax distribution is 1-Lipschitz and 1-smooth where the derivative is taken with respect to the logits. Hence, the L-Lipschitz and β-smoothness assumption holds also for the probability distribution of Equation 2. In practice, estimating the influence on the probability distribution from Equation 2 is what one is interested in and what we evaluate in our experiments.

The following lemma generalizes the known $(1+\alpha\beta)$-expansiveness property of the gradient update rule (Hardt, Recht, and Singer 2016). It establishes that the increase of the distance between $\boldsymbol{w}-\boldsymbol{\gamma}$ and $\boldsymbol{w}'$ after one step of gradient descent is at most as much as the increase in distance between two parameter vectors corresponding to $\mathcal{D}$ and $\mathcal{D}-\{d'\}$ after one step of gradient descent.

Lemma 4. *Let $f : \Omega \rightarrow \mathbb{R}$ be a function and let $G(\boldsymbol{w}) = w - \alpha\nabla f(\boldsymbol{w})$ be the gradient update rule with step size α. Moreover, assume that f is β-smooth. Then, for every $\boldsymbol{w}, \boldsymbol{w}', \boldsymbol{\gamma} \in \Omega$ we have*

$$\|G(\boldsymbol{w}) - \boldsymbol{\gamma} - G(\boldsymbol{w}')\| \leq \|\boldsymbol{w} - \boldsymbol{\gamma} - \boldsymbol{w}'\| + \alpha\beta \|\boldsymbol{w} - \boldsymbol{w}'\|.$$

Lemma 5. *Let $f(\cdot; d)$ be L-Lipschitz and β-smooth function. Suppose we run SGD for T steps on two sets of triples $\mathcal{D}$ and $\mathcal{D} - \{d'\}$ for any $d' \in \mathcal{D}$ and with learning rate α_t at time step t. Moreover, let $\Delta_t = \mathbb{E}\left[\|\boldsymbol{w}_t - \boldsymbol{w}'_t\| \mid \left\|\boldsymbol{w}_{t_0} - \boldsymbol{w}'_{t_0}\right\| = 0\right]$ and $\hat{\Delta}_t = \mathbb{E}\left[\left\|\boldsymbol{w}_t - \boldsymbol{\gamma}_{[d', \boldsymbol{w}_t(d)]} - \boldsymbol{w}'_t\right\| \mid \left\|\boldsymbol{w}_{t_0} - \boldsymbol{w}'_{t_0}\right\| = 0\right]$ for some $t_0 \in \{1, ..., n\}$. Then, for all $t \geq t_0$,*

$$\hat{\Delta}_{t+1} < \left(1 - \frac{1}{n}\right)(1 + \alpha_t\beta)\Delta_t + \frac{1}{n}(\Delta_t + \alpha_t L).$$

The lemma has broader implications since it can be extended to other update rules G as long as they fulfill an η-expansive property and their individual updates are bounded. For each of these update rules the above lemma holds.

The following theorem establishes that the approximation error of gradient rollback is smaller than a known stability bound of SGD. It uses Lemma 5 and the proof of Theorem 1.

Theorem 2. *Let $f(\cdot; d) \in [0,1]$ be an L-Lipschitz and β-smooth function. Suppose we run SGD for T steps with monotonically non-increasing step sizes $\alpha_t \leq c/t$ on two sets of triples $\mathcal{D}$ and $\mathcal{D} - \{d'\}$. Let $\boldsymbol{w}_T$ and $\boldsymbol{w}'_T$, respectively, be the resulting parameters. Then, for any triple d that has at least one element in common with d' we have,*

$$\mathbb{E}\left|f(\boldsymbol{w}_T - \boldsymbol{\gamma}_{[d', \boldsymbol{w}_T(d)]}; d) - f(\boldsymbol{w}'_T; d)\right| < \Lambda_{\textit{stab-nc}}.$$

The previous results establish a connection between estimating the influence of training triples on the model's behavior using GR and the stability of SGD when used to train the model. An interesting implication is that regularization approaches that improve the stability (by reducing the Lipschitz constant and/or the expansiveness properties of the learning dynamics, cf. (Hardt, Recht, and Singer 2016)) also reduce the error bound of GR. We can indeed verify this empirically.

Related Work

The first neural link prediction method for multi-relational graphs performing an implicit matrix factorization is RESCAL (Nickel, Tresp, and Kriegel 2011). Numerous scoring functions have since been proposed. Popular examples are TRANSE (Bordes et al. 2013), DISTMULT (Yang et al. 2015), and COMPLEX (Trouillon et al. 2016). Knowledge graph embedding methods have been mainly evaluated through their accuracy on link prediction tasks. A number of papers has recently shown that with appropriate hyperparameter tuning, COMPLEX, DISTMULT, and RESCAL are highly competitive scoring functions, often achieving state-of-the-art results (Kadlec, Bajgar, and Kleindienst 2017; Ruffinelli, Broscheit, and Gemulla 2020; Jain et al. 2020). There are a number of proposals for combining rule-based and matrix factorization methods (Rocktäschel, Singh, and Riedel 2015; Guo et al. 2016; Minervini et al. 2017), which can make link predictions more interpretable. In contrast, we aim to generate faithful explanations for non-symbolic knowledge graph embedding methods.

There has been an increasing interest in understanding model behavior through adversarial attacks (Biggio, Fumera, and Roli 2014; Papernot et al. 2016; Dong et al. 2017; Ebrahimi et al. 2018). Most of these approaches are aimed at visual data. There are, however, several approaches that consider adversarial attacks on graphs (Dai et al. 2018; Zügner, Akbarnejad, and Günnemann 2018). While analyzing attacks can improve model interpretability, the authors focused on neural networks for single-relational graphs and the task of node classification. For a comprehensive discussion of adversarial attacks on graphs we refer the reader to a recent survey (Chen et al. 2020). There is prior work on adversarial samples for KGs but with the aim to improve accuracy and not model interpretability (Minervini et al. 2017; Cai and Wang 2018).

There are two recent papers that directly address the problem of explaining graph-based ML methods. First, GNNEXPLAINER (Ying et al. 2019) is a method for explaining the predictions of graph neural networks and, specifically, graph convolutional networks for node classification. Second, the work most related to ours proposes CRIAGE which aims at estimating the influence of triples in KG embedding methods (Pezeshkpour, Tian, and Singh 2019). Given a triple, their method only considers a neighborhood to be the set of triples with the same object. Moreover, they derive a first-order approximation of the influence in line with work on influence functions (Koh and Liang 2017). In contrast, GR tracks the changes made to the parameters during training and uses the aggregated contributions to estimate influence. In addition, we establish a theoretical connection to the stability of learning systems. Influence functions, a concept from robust statistics, were applied to black-box models for assessing the changes in the loss caused by changes in the training data (Koh and Liang 2017). The paper also proposed several strategies to make influence functions more efficient. It was shown in prior work (Pezeshkpour, Tian, and Singh 2019), however, that influence functions are not usable for typical knowledge base embedding methods as they scale very poorly. Consequently, our paper is the first to offer an efficient and theoretically founded method of tracking influence in matrix factorization models for explaining prediction via providing the most influential training instances.

	PD%										TC%								
	NATIONS			FB15K-237			MOVIELENS				NATIONS			FB15K-237			MOVIELENS		
	1	10	ALL	1	10	ALL	1	10	ALL		1	10	ALL	1	10	ALL	1	10	ALL
NH	54	66	82	59	67	83	53	52	92	NH	18	36	70	35	45	59	3	14	72
GR	93	97	100	77	82	96	68	82	100	GR	38	83	97	38	58	85	20	38	100
$\Delta\uparrow$	39	31	18	18	15	13	15	30	8	$\Delta\uparrow$	20	47	27	3	13	26	17	24	28
NH	59	68	80	72	83	91	53	61	91	NH	13	47	76	69	70	79	5	13	77
GR	90	95	100	80	88	99	73	71	100	GR	38	85	99	65	81	91	29	48	100
$\Delta\uparrow$	31	27	20	8	5	8	20	10	9	$\Delta\uparrow$	25	38	23	-4	11	12	24	35	23

Table 1: Results (DISTMULT at the top, COMPLEX at the bottom) of removing a set of training triples (of size 1, 10, or ALL), randomly chosen from triples adjacent to the test triples (NH) or by using gradient rollback (GR). For ALL we delete on average ($\pm$ standard deviation), NATIONS: 261$\pm$56, FB15K-237: 2.9k$\pm$2.3k and MOVIELENS: 16.7k$\pm$5k (DISTMULT). The average number of adjacent triples ($\pm$ standard deviation) is, NATIONS: 508$\pm$101, FB15K-237: 5.5k$\pm$4.5k and MOVIELENS: 23.7k$\pm$7.8k. GR removes sets that lead to a larger change in probability and top-1 predictions (difference to NH is given in row $\Delta\uparrow$).

Experiments

Identifying Explanations. We analyze the extent to which GR can approximate the true influence of a triple (or set of triples). For a given test triple $d = (s, r, o)$ and a trained model with parameters $\boldsymbol{w}$, we use GR to identify a set of training triples $\mathcal{S} \subseteq \mathcal{D}$ that has the highest influence on d. To this end, we first identify the set of triples $\mathcal{N}$ adjacent to d (that is, triples that contain at least one of s, r or o) and compute $\Delta(d', d) = \Pr(\boldsymbol{w}; o \mid s, r) - \Pr(\boldsymbol{w} - \boldsymbol{\gamma}_{[d', \boldsymbol{w}(d)]}; o \mid s, r)$ for each $d' \in \mathcal{N}$. We then let $\mathcal{S}$ be the set resulting from picking (a) exactly k triples $d' \in \mathcal{N}$ with the k largest values for $\Delta(d', d)$ or (b) *all* triples with a positive $\Delta(d', d)$. We refer to the former as GR-k and the latter as GR-ALL.

To evaluate if the set of chosen triples $\mathcal{S}$ are faithful explanations (Jacovi and Goldberg 2020) for the prediction, we follow the evaluation paradigm "RemOve And Retrain (ROAR)" of Hooker et al. (2019): We let $\mathcal{D}' = \mathcal{D} - \mathcal{S}$ and retrain the model from scratch with training set $\mathcal{D}'$ leading to a new model with parameters $\boldsymbol{w}'$. After retraining, we can now observe $\Pr(\boldsymbol{w}'; o \mid s, r)$, which is the true probability for d when removing the explanation set $\mathcal{S}$ from the training set, and we can use this to evaluate GR.[7] Since it is expensive to retrain a model for each test triple, we restrict the analysis to explaining only the triple $(s, r, \hat{o})$ with $\hat{o} = \arg\max_o \Pr(\boldsymbol{w}; o \mid s, r)$ for each test query $(s, r, ?)$, that is, the triple with the highest score under the model.

Evaluation Metrics. We use two different metrics to evaluate GR. First, if the set $\mathcal{S}$ contains triples influential to the test triple d, then the probability of d under the new model (trained without $\mathcal{S}$) should be smaller, that is, $\Pr(\boldsymbol{w}'; o \mid s, r) < \Pr(\boldsymbol{w}; o \mid s, r)$. We measure the ability of GR to identify triples causing a *P*robability *D*rop and name this measure PD%. A PD of 100% would imply that each set $\mathcal{S}$ created with GR always caused a drop in probability for d after retraining with $\mathcal{D}'$. In contrast, when removing random training triples, we would expect a PD% close to 50%. Second, we measure whether the removal of $\mathcal{S}$ causes the newly trained model to predict a different top-1 triple, that is, $\arg\max_o \Pr(\boldsymbol{w}; o \mid s, r) \neq \arg\max_o \Pr(\boldsymbol{w}'; o \mid s, r)$. This measures the ability of GR to select triples causing the *T*op-1 prediction to *C*hange and we name this TC%. If the removal of $\mathcal{S}$ causes a top-1 change, it suggests that the training samples most influential to the prediction of triple d have been removed. This also explores the ability of GR to identify triples for effective removal attacks on the model. We compare GR to two baselines: NH-k removes exactly k random triples adjacent to d and NH-ALL randomly removes the same number of triples as GR-ALL adjacent to d. In an additional experiment, we also directly compare GR with CRIAGE (Pezeshkpour, Tian, and Singh 2019).

Datasets & Training. We use DISTMULT (Yang et al. 2015) and COMPLEX (Trouillon et al. 2016) as scoring functions since they are popular and competitive (Kadlec, Bajgar, and Kleindienst 2017). We report results on three datasets: two knowledge base completion (NATIONS (Kok and Domingos 2007), FB15K-237 (Toutanova et al. 2015)) and one recommendation dataset (MOVIELENS (Harper and Konstan 2015)). MOVIELENS contains triples of 5-star ratings users have given to movies; as in prior work the set of entities is the union of movies and users and the set of relations are the ratings (Pezeshkpour, Chen, and Singh 2018). When predicting movies for a user, we simply filter out other users. Statistics and hyperparameter settings are in the appendix. Since retraining is costly, we only explain the top-1 prediction for a set of 100 random test triples for both FB15K-237 and MOVIELENS. For NATIONS we use the entire test set.

We want to emphasize that we *always retrain completely from scratch* and use hyperparameter values typical for state-of-the-art KG completion models, leading to standard results for DISTMULT (see Table 4 in the Appendix). Prior work either retrained from pretrained models (Koh and Liang 2017; Pezeshkpour, Tian, and Singh 2019) or used non-standard strategies and hyperparameters to ensure model convergence (Pezeshkpour, Tian, and Singh 2019).

Results. The top half of Table 1 lists the results using DISTMULT for GR and NH. For NATIONS and DISTMULT, re-

[7] We fix all random seeds and use the same set of negative samples during (re-)training to avoid additional randomization effects.

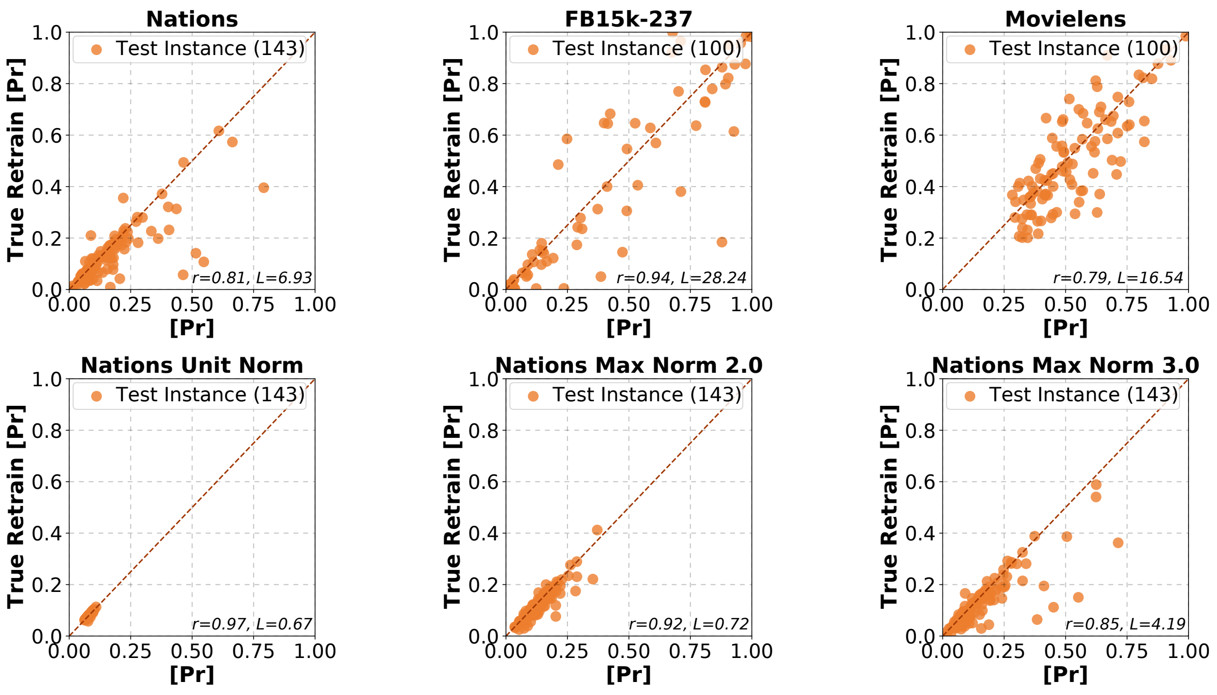

Figure 2: Scatter plots illustrating the correlation between the probability values estimated by GR and after a retraining of the model, as well as Pearson correlation value *r* and Lipschitz constant *L*. The high correlations indicate that GR is able to accurately approximate the probability and, therefore, the change in probability, of a triple after retraining the model without the triple that GR deemed to cause the highest change in probability. The bottom row shows results when imposing constraints on the weights for the NATIONS dataset: the stronger the constraint (unit norm $>$ maximum norm 2.0 $>$ 3.0 $>$ None), the smaller the Lipschitz constant and the better the Pearson correlation.

	PD%				TC%			
	1	3	5	10	1	3	5	10
NH	49	50	51	60	3	13	14	20
CRIAGE	91	93	94	95	16	36	48	68
GR-O	**92**	94	**97**	97	25	<u>48</u>	<u>62</u>	68
GR	**92**	**96**	**97**	**98**	**<u>27</u>**	**<u>51</u>**	**<u>66</u>**	**73**

Table 2: Results on NATIONS, using DISTMULT with a SIGMOID activation function and for $k = \{1, 3, 5, 10\}$. Bold marks the best results; underlined results mark a statistical significance with regards to CRIAGE at $p \leq 0.01$ using an approximate randomization test; all results are statistically significant with regards to NH. CRIAGE performs worse than both GR and GR-O, especially with regards to TC. Furthermore, CRIAGE considers only training triples with the same object as an explanation and is significantly slower.

moving 1 triple from the set of adjacent triples (NH-1) is close to random, causing a drop in probability (PD%) about half of the time. In contrast, removing the 1 training instance considered most influential by GR (GR-1), gives a PD of over 90%. GR-1 leads to a top-1 change (TC) in about 40% of the cases. This suggests that there is more than one influential triple per test triple. When removing all triples identified by GR (GR-ALL) we observe PD and TC values of nearly 100%. In contrast, removing the same number of adjacent triples randomly leads to a significantly lower PC and TC. In fact, removing the highest ranked triple (GR-1) impacts the model behavior more than deleting about half the adjacent triples at random (NH-ALL) in terms of PD%.

For FB15K-237 we observe that GR is again better at identifying influential triples compared to the NH baselines. Moreover, only when the NH method deletes over half of the adjacent triples at random (NH-ALL) does it perform on par with GR-10 in terms of PD% and TC%. Crucially, deleting one instance at random (NH-1) causes a high TC% compared to the other two datasets. This suggests that the model is less stable for this dataset. As our theoretical results show, GR works better on more stable models, and this could explain why GR-ALL is further away from reaching a TC% value of 100 than on the other two datasets.

For MOVIELENS GR is again able to outperform all respective NH baselines. Furthermore, on this dataset GR-ALL is able to achieve perfect PD% and TC% values. Additionally, GR shows a substantial increase for PD% and TC% when moving from $k = 1$ to $k = 10$, suggesting that more than one training triple is required for an explanation. At the same time, NH-10 still does not perform better than random in terms of PD%. The bottom half of Table 1 reports results using COMPLEX, which show a similar pattern as DISTMULT.

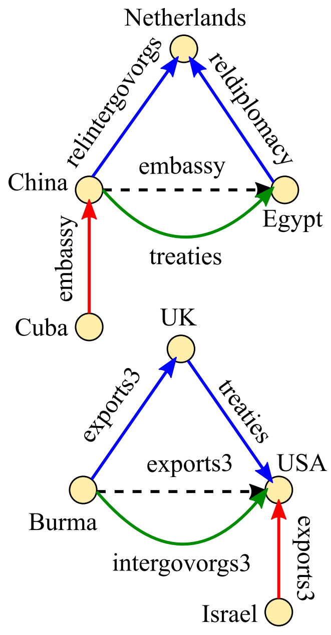

Figure 3: Some example explanations generated using GR. The dashed line indicates the test triple. Depicted are triples deemed highly influential on model behavior by GR (i) with the same head and tail as test triple (green), (ii) with the same relation type (red), and (iii) a pair of triples via a third entity (blue).

Approximation Quality. In this experiment we select, for each test triple, the training triple d' GR deemed most influential. We then retrain without d' and compare the predicted probability of the test triple with the probability (PR) after retraining the model. Figure 2 shows that GR is able to accurately estimate the probability of the test triple after retraining. For all datasets the correlation is rather high. We also confirm empirically that imposing a constraint on the weights (enforcing unit norm or a maximum norm), reduces the Lipschitz constant of the model and in turn reduces the error of GR, as evidenced by the increase in correlation. Figure 3 also shows two qualitative examples.

Comparison to CRIAGE. Like GR, CRIAGE can be used to identify training samples that have a large influence on a prediction. However, CRIAGE has three disadvantages: (1) it only considers training instances with the same object as the prediction as possible explanations; (2) it only works with a SIGMOID activation function and not for SOFTMAX; (3) retrieving explanations is time-consuming, e.g. on MOVIELENS GR can identify an explanation in 3 minutes whereas CRIAGE takes 3 hours. Consequently, we only run CRIAGE on the smaller NATIONS dataset. For a direct comparison, we introduce GR-O, which like CRIAGE only considers training instances as possible explanations if they have the same object as the prediction. Results for the top-$k \in \{1, 3, 5, 10\}$ are reported in Table 2. Both GR and GR-O outperform CRIAGE and GR performs best overall for all k and both metrics.

Conclusion

Gradient rollback (GR) is a simple yet effective method to track the influence of training samples on the parameters of a model. Due to its efficiency it is suitable for large neural networks, where each training instance touches only a moderate number of parameters. Instances of this class of models are neural matrix factorization models. To showcase the utility of GR, we first established theoretically that the resource overhead is minimal. Second, we showed that the difference of GR's influence approximation and the true influence on the model behavior is smaller than known bounds on the stability of stochastic gradient descent. This establishes a link between influence estimation and the stability of models and shows that, if a model is stable, GR's influence estimation error is small. We showed empirically that GR can successfully identify sets of training samples that cause a drop in performance if a model is retrained without this set. In the future we plan to apply GR to other models.

Acknowledgements

We thank Takuma Ebisu for his time and insightful feedback.

Ethical Impact

This paper addresses the problem of explaining and analyzing the behavior of machine learning models. More specifically, we propose a method that is tailored to a class of ML models called matrix factorization methods. These have a wide range of applications and, therefore, also the potential to provide biased and otherwise inappropriate predictions in numerous use cases. For instance, the predictions of a recommender system might be overly gender or race specific. We hope that our proposed influence estimation approach can make these models more interpretable and can lead to improvements of production systems with respect to the aforementioned problems of bias, fairness, and transparency. At the same time, it might also provide a method for an adversary to find weaknesses of the machine learning system and to influence its predictions making them more biased and less fair. In the end, we present a method that can be used in several different applications. We believe, however, that the proposed method is not inherently problematic as it is agnostic to use cases and does not introduce or discuss problematic applications.

References

Bianchi, F.; Rossiello, G.; Costabello, L.; Palmonari, M.; and Minervini, P. 2020. Knowledge Graph Embeddings and Explainable AI. In Ilaria Tiddi, Freddy Lecue, P. H., ed., *Knowledge Graphs for eXplainable AI – Foundations, Applications and Challenges. Studies on the Semantic Web*. Amsterdam: IOS Press.

Biggio, B.; Fumera, G.; and Roli, F. 2014. Security Evaluation of Pattern Classifiers under Attack. *IEEE Transactions on Knowledge and Data Engineering* 26(4): 984–996.

Bordes, A.; Usunier, N.; Garcia-Duran, A.; Weston, J.; and Yakhnenko, O. 2013. Translating Embeddings for Modeling Multi-relational Data. In *Advances in Neural Information Processing Systems 26, NeurIPS*, 2787–2795.

Bousquet, O.; and Elisseeff, A. 2002. Stability and Generalization. *Journal of Machine Learning Research* 2: 499–526.

Cai, L.; and Wang, W. Y. 2018. KBGAN: Adversarial Learning for Knowledge Graph Embeddings. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL*, 1470–1480.

Chen, L.; Li, J.; Peng, J.; Xie, T.; Cao, Z.; Xu, K.; He, X.; and Zheng, Z. 2020. A Survey of Adversarial Learning on Graphs. *arXiv* abs/2003.05730: 1–28.

Dai, H.; Li, H.; Tian, T.; Huang, X.; Wang, L.; Zhu, J.; and Song, L. 2018. Adversarial Attack on Graph Structured Data. In *Proceedings of the 35th International Conference on Machine Learning, ICML*, 1115–1124.

Dong, Y.; Su, H.; Zhu, J.; and Bao, F. 2017. Towards Interpretable Deep Neural Networks by Leveraging Adversarial Examples. *arXiv* abs/1708.05493: 1–12.

Ebrahimi, J.; Rao, A.; Lowd, D.; and Dou, D. 2018. HotFlip: White-Box Adversarial Examples for Text Classification. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL*, 31–36.

Guo, Q.; Zhuang, F.; Qin, C.; Zhu, H.; Xie, X.; Xiong, H.; and He, Q. 2020. A Survey on Knowledge Graph-Based Recommender Systems. *arXiv* abs/2003.00911: 1–17.

Guo, S.; Wang, Q.; Wang, L.; Wang, B.; and Guo, L. 2016. Jointly Embedding Knowledge Graphs and Logical Rules. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing, EMNLP*, 192–202.

Hardt, M.; Recht, B.; and Singer, Y. 2016. Train Faster, Generalize Better: Stability of Stochastic Gradient Descent. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning, ICML*, 1225—1234.

Harper, F. M.; and Konstan, J. A. 2015. The MovieLens Datasets: History and Context. *ACM Transactions on Interactive Intelligent Systems* 5(4).

Hooker, S.; Erhan, D.; Kindermans, P.-J.; and Kim, B. 2019. A Benchmark for Interpretability Methods in Deep Neural Networks. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 9737–9748.

Jacovi, A.; and Goldberg, Y. 2020. Towards Faithfully Interpretable NLP Systems: How should we define and evaluate faithfulness? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL*, 4198–4205.

Jain, P.; Rathi, S.; Chakrabarti, S.; et al. 2020. Knowledge Base Completion: Baseline strikes back (Again). *arXiv* abs/2005.00804: 1–6.

Kadlec, R.; Bajgar, O.; and Kleindienst, J. 2017. Knowledge Base Completion: Baselines Strike Back. In *Proceedings of the 2nd Workshop on Representation Learning for NLP*, 69–74.

Koh, P. W.; and Liang, P. 2017. Understanding Black-box Predictions via Influence Functions. In *Proceedings of the 34th International Conference on Machine Learning, ICML*, 1885–1894.

Kok, S.; and Domingos, P. 2007. Statistical Predicate Invention. In *Proceedings of the 24th International Conference on Machine Learning, ICML*, 433–440.

Minervini, P.; Demeester, T.; Rocktäschel, T.; and Riedel, S. 2017. Adversarial Sets for Regularising Neural Link Predictors. In *Conference on Uncertainty in Artificial Intelligence, UAI*, 9240–9251.

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*, 809–816.

Papernot, N.; McDaniel, P.; Jha, S.; Fredrikson, M.; Celik, Z. B.; and Swami, A. 2016. The Limitations of Deep Learning in Adversarial Settings. In *2016 IEEE European Symposium on Security and Privacy, EuroS&P*, 372–387.

Pezeshkpour, P.; Chen, L.; and Singh, S. 2018. Embedding Multimodal Relational Data for Knowledge Base Completion. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP*, 3208–3218.

Pezeshkpour, P.; Tian, Y.; and Singh, S. 2019. Investigating Robustness and Interpretability of Link Prediction via Adversarial Modifications. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL*, 3336–3347.

Rocktäschel, T.; Singh, S.; and Riedel, S. 2015. Injecting Logical Background Knowledge into Embeddings for Relation Extraction. In *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics, NAACL*, 1119–1129.

Ruffinelli, D.; Broscheit, S.; and Gemulla, R. 2020. You CAN Teach an Old Dog New Tricks! On Training Knowledge Graph Embeddings. In *8th International Conference on Learning Representations, ICLR*.

Toutanova, K.; Chen, D.; Pantel, P.; Poon, H.; Choudhury, P.; and Gamon, M. 2015. Representing Text for Joint Embedding of Text and Knowledge Bases. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, EMNLP*, 1499–1509.

Trouillon, T.; Welbl, J.; Riedel, S.; Gaussier, É.; and Bouchard, G. 2016. Complex Embeddings for Simple Link Prediction. In *International Conference on Machine Learning, ICML*, 2071–2080.

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*.

Ying, Z.; Bourgeois, D.; You, J.; Zitnik, M.; and Leskovec, J. 2019. GNNExplainer: Generating Explanations for Graph Neural Networks. In *Advances in Neural Information Processing Systems 32, NeurIPS*, 9240–9251.

Zügner, D.; Akbarnejad, A.; and Günnemann, S. 2018. Adversarial Attacks on Neural Networks for Graph Data. In *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, 2847–2856.