Repeating Link Prediction over Dynamic Graphs

Daniele Montesi
Department of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
montesi@kth.se

Sarunas Girdzijauskas
Department of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
sarunasg@kth.se

Vladimir Vlassov
Department of Computer Science
KTH Royal Institute of Technology
Stockholm, Sweden
vladv@kth.se

2020 IEEE International Conference on Big Data (Big Data) | 978-1-7281-6251-5/20/$31.00 ©2020 IEEE | DOI: 10.1109/BigData50022.2020.9378360

***Abstract*—Graphs are a vastly useful and widely used form of modeling and representation of systems, processes, entities, events, objects, components etc., in various domains of discourse, that reflects relations or connections of modeled entities. Graphs are vital to diverse data mining applications, as they capture relationships between data items, such as dependencies or interactions, and graph analysis can reveal valuable insights for many application domains including machine learning, anomaly detection, clustering, recommendations, social influence analysis, bioinformatics, and others. The analysis of the evolutionary behavior of dynamic graphs provides the means to continuously predict the appearance, and also, the disappearance of new graph links, i.e., to perform the Dynamic Link Prediction Task. Dynamic Link Prediction has been explored widely in the past years; however, the majority of these works focus on discovering new edges (by implicitly assuming ever growing dynamic networks). However, very few works focus on the repeating edges, i.e., links that continuously vanish and reappear in the dynamic network, but which size (in terms of number of nodes and edges) does not significantly change over long periods of time. In this work, we first study the literature for link prediction in the static settlement, then, we focus on dynamic link prediction, underlining the strengths and weaknesses of every approach studied. We discover that traditional methods do not work well with repeating links as they are unable to encode temporal patterns associated with the edges while also considering the topological graph features. We propose a novel method, Temporal Edge Embedding Neural Network (TEEN), which is based on a deep learning architecture that jointly optimizes the prediction of the correct edge labels as well as the proximity of two nodes' pairs in their latent space at every time step. Our solution benefits of node embeddings created with deep encoders from where an edge embedding is created for every time step. Our evaluation experiments on transactional graphs show that TEEN is able to outperform state-of-the-art models by over 8% on AUC and over 7% on F1-Score. We show that our approach brings significant improvements in the scenario of transactional graphs.**

***Index Terms*—Repeating Link Prediction, Dynamic Graphs, Graph Mining, Deep Learning**

I. Introduction

The Dynamic Link Prediction task regards the prediction of future connections that are going to be formed in the network through the analysis of a temporally evolving network. It has numerous real-world applications in the study of Social Networks to recommend new friends to the users [1], [2], or in the paper citation graph to predict next connections between authors [3]. Dynamic Link Prediction differs from traditional link prediction because the first uses temporal information of the evolving graph to infer what link will be forming in the future, while the latter uses the static network. Given a sequence of time periods $1, 2, \dots T-1, T$, we define Dynamic Graph an evolving network represented as a contiguous sequence of graph snapshots $G_1, G_2, \dots, G_{T-1}, G_T$.

Many types of sub-tasks can be distinguished when talking about link prediction. Dynamic Link Prediction refers to the prediction of new links forming in the network exploiting the sequence of graph snapshots described before to predict the new graph G_{T+1}. Generally, a common assumption throughout the state of the art (SOTA) methods of link prediction is that once a link is formed, it remains in the graph also in the subsequent snapshots. Thus, link prediction is considered for predicting *new links* in *growing networks* only. However, this assumption might be a limitation concerning some real-world applications where links are not fixed once formed. For instance, we can think about the temporal graph of the instant message exchange between users in social networks: representing a message as a link occurring at a specific time, we can track the users exchanging messages during a specific period of times (days, weeks, months) and predict whether this connection will occur again or if it will 'temporally' vanish in the subsequent snapshot. The same scenario is valid for the temporal graph of emails between employees or transactions networks: given the repetitive nature of the links in the graph considered, these problems cannot be modeled in constantly growing networks where links are permanently added.

Those types of dynamic graphs can be studied in other terms, for instance, the analysis is not focused on the number of nodes introduced in the graphs, nor in the new link added to the graph itself, but on the behavior that such repeating links show in their appearance/disappearance. For the use cases of above, a more appropriate task is Repeating Link Prediction. Repeating Link Prediction is one of the two sub-tasks associated to Dynamic Link Prediction: the first focuses on predicting *new links*, i.e., the edges that occur in the graph but never occurred before, the second is about the problem of interest of this paper, the *repeating links*, i.e., the edges that occurred at least once in the past and might appear in the next graph snapshot. The former sub-task relates to the analysis of ever-growing networks, while the latter sub-task relates the analysis of networks under link churn (i.e., the interruption of the relationship). The majority of the works related to generic Link Prediction focus only on the study

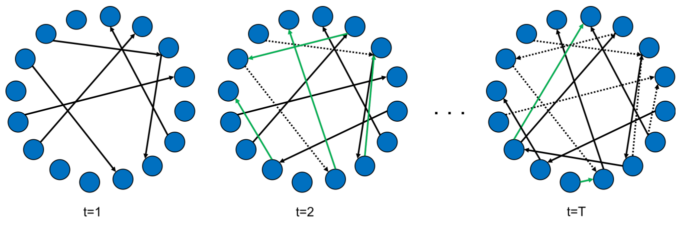

Fig. 1. Dynamic graph with repeating links (in black), and new links (in green). Dotted lines identify links that are vanishing

of *new connections* that will be forming between every pair of nodes without taking into account the prediction of edges that continuously *appear* and *vanish* at different time steps, once they have appeared once in the past. However, this ought to be a limitation for some networks. What it usually happens over a large class of real-world graphs (e.g., instant messaging networks, transaction networks, etc.) is that, as the networks evolve, the ratio between the appearance of new links decrease and the repeating links increase significantly, as shown in Figure 1). In short, link prediction on *repeating links* is a prediction that targets only on the connections existing previously in the dynamic graph, while on *new links* we consider all the non-existing ones.

It is clear that besides the two above-mentioned classes of edges are linked to the same task, they require a different methodology. One is the evaluation method: when assessing new links, all the possible non-linked nodes must be considered as potential connections and several K most likely neighbors are considered as future edges. This is the case of new friends recommendation in social networks (some related evaluation methods are explored in the paper of Yang et al [4]). When assessing repeating links, the test set is *pre-determined* and is equal to the connections present *at least once* in the past graph snapshots. Let us take a transactional dataset: a directed weighted graph containing the payments exchanged between companies. In such scenario, the test set becomes the union of all relationship occurring in the past between these firms for which we would like to predict the presence (1) or absence (0) in the next graph snapshot. Thus, the evaluation metrics adopted are the ones of a *binary classification*. Another important aspect to consider regards *time series modeling*: when considering new link prediction sub-task, the algorithm usually relies more on the overall graph behavior statically or dynamically analyzing the graph topology evolution through time. When considering repeating link prediction, the analysis may benefit from a time series approach applied exclusively on the edge occurrence at every time step. If we consider the transactional dataset example of above keeping the focus on a single edge over time, we can observe two temporal patterns: the *trend* (e.g., several transactions increase as time passes) and the *seasonality* (e.g., every year during certain months, the transaction is present, while in certain periods it vanishes). This could not be considered while evaluating new edges since it lacks a *history* to consider for that link.

While link prediction is very well studied by the literature, as of 2020, the majority of these works focus only on new links, i.e., on the ever-growing networks. Repeating links are introduced for the first time in less recent papers of Tylenda et al, in 2009, [5], also by Oyama et al [6] and Lankeshwara [7] where it is observed that repeating links are studied poorly in the literature. A more recent work is the one of Patel et al [8] which considers the task of repeating links again, without offering a specific solution for that problem. There are research efforts on focusing on the separate tasks of dissolution graph and formation graph prediction, which is partially related with the problem of repeating link prediction. Hisano [9] evaluates separately the two networks as a sequential graph transition: one for the link formation, one for the link dissolution. Still, this modelling has the problem to be focused on predicting *all* the possible edges, including the new links, rather than on the repeating ones. Especially in the prediction of the formation network, this modelling has shown limitations observed by the author of the paper itself [9].

In this paper, we target the Repeating Link Prediction problem with the workloads coming from the use case of bank transactions. In particular, we analyze transactions done between corporate clients, where the nodes are single clients, identified by their IBAN, and the edges are identified by money transactions. The analysis of repeating edges over the network of client transactions can play an important role in the bank scenario. To the best of our knowledge, this work is the first to assess the performance of a link prediction algorithm on repeating links over evolving dynamic graphs.

As baselines, we pick several important SOTA algorithms relying on deep learning: TNodeEmbed [10], DynamicGraph2Vec [11], DDNE [12], as well as static link prediction techniques: Node2Vec [13], SDNE [14] and HOPE [15]. Additionally, we implement a simple unsupervised baseline, Common Neighbors. As already explained, the scores will be calculated exclusively over repeating link, hence these methods were adapted to solve this problem.

The main contribution of this paper is the introduction of our solution for repeating the link problem: a Deep Learning model dealing with Link Prediction over Dynamic Networks modelling edge embeddings at every time step. We have shown that our solution achieves prediction scores higher on average +8% on AUC and +7% on F1-Score if compared to other SOTA techniques. During the model design, we assessed different model architectures relying on the computation of edge embeddings via node *embeddings subtraction* and *edge values concatenation*. Moreover, in this paper, we show through experiments how the introduction of an additional unsupervised component to the overall loss function improves considerably the performance of the model.

The rest of the paper is structured as follows. Section II describes the background. Section III introduces the new model proposed, namely TEEN (Temporal Edge Embedding Neural Network). Section IV presents the experimental and evaluation setup, including datasets, tools and frameworks used in this study, the splitting criteria of the data, the baselines implemented and the evaluation metrics. Section V provides the experimental results. Section VI discusses related work. We conclude and highlight our future work in Section VII.

II. Background

In the background section, we define all the necessary elements linking also some important related works. We will refer to the term graph as a mathematical model $G = (V, E)$ describing the structure of the network where V is a set of elements called *nodes* or *vertices*, E is a set of elements named *edges* or *links*. Every graph can be represented with an adjacency matrix A: a square matrix of order $|V|$ whose elements in the coordinate (i, j) indicate whether pairs of nodes i, j, present a connection or not.

A. Static Link Prediction

In the context of dynamic link prediction, some techniques used in the static settlement can be used on the dynamic one. Static Graphs are network models that are evaluated statically, i.e., considering a fixed set of vertices V and a set of present edges E. Given a network at a time t $G_t = (V^t, E^t)$, in Static Link Prediction we want to predict the new graph $G_{t+1} = (V^{t+1}, E^{t+1})$. A simple approach for static link prediction is computing a node proximity function. Given a pair of vertices (i, j), the prediction for the node neighborhood is given by the highest scored pair of vertices given the proximity function $f(i, j)$. The pairs are hence ranked following the chosen function, and the *top-K* ranked pairs are assigned as the predicted neighborhood for the node. The strategy above can be considered *unsupervised* since no training is needed. A simple proximity score is *Common Neighbors*, representing the fraction of common neighbors between 2 nodes: $F_{CN}(u, v) = \frac{|Adj(u) \cap Adj(v)|}{|Adj(u) \cup Adj(v)|}$. It is also possible to combine multiple metrics in a weighted way to perform a hybrid approach as done by Hasan et al [16]. Other approaches are taking advantage of Machine Learning, which as stated in surveys [16], [17], usually these techniques outperform the unsupervised ones considerably. When using machine learning on Link Prediction, several problems should be taken into account. One of them is *dataset imbalance*. Given the high *sparsity* of real-world graphs, it is likely that the positive class would be less represented than the positive one, with the risk that the machine learning algorithm would naturally classify all the samples in the negative class.

B. Embeddings

In recent years there was the need to represent graphs in a way such that it was possible to apply Machine Learning algorithms on them. Generally, this requires transforming each sample (a node or an edge) into a vector. Researchers focused their study to find a way to represent the graph *projecting* its nodes of a d-dimensional space R^d such that for every node v, its representation was described by a vector $\vec{v}$ of dimension d (called *embedding*). These techniques are known under the name of Graph Representation Learning (GRL) and they can ease machine learning algorithms to work with graphs serving a multitude of problems such as *Clustering, Node Classification* and *Link Prediction*. A simple GRL technique is Node2Vec. In its paper [13], Grover and Leskovec describe the goal of Node2Vec as the projection of the graph in an embedded space such that the likelihood of the adjacency set of every node is maximized.

C. Neural Networks in Dynamic Link Prediction

Dealing with dynamic link prediction is usually a more complex problem where static graph techniques are not enough to achieve good results. Even though Embeddings can be computed for static graphs, some techniques allow computing embeddings that take into account temporal patterns. Deep Learning methods are used very often for tasks concerning network analysis. Their main advantage is that the process of feature engineering is performed automatically by feature-extraction layers. Some architectures adopted for graph analysis to be mentioned are Autoencoders and Recurrent Neural Networks (RNN) described as follows.

An Autoencoder is a deep neural network made of two symmetrical components, the Encoder, and the Decoder. The first projects the input onto a lower-dimensional latent space, also called *embedding*, while the second reconstructs the input starting from this embedding, with fully connected layers of the same Encoder's shape. The objective function employed by Autoencoders is usually the Mean Squared Error (MSE). The structure is widely used in image processing to extract features from pictures and making the model more robust introducing noise, but it is also used in graphs for reconstructing their nodes' adjacencies [11], [14], [18]. What an Autoencoder tries to do computing an embedding is to perform dimensionality reduction, a task well known in machine learning and it is an unsupervised learning method.

RNNs are also widely used architectures for graph analysis able to model temporal sequences of the size to predict the future. The most popular typologies of RNNs are the *sequence-2-sequence* when a sequence of inputs is given as input and another sequence of values is returned as output, *sequence-to-vector*, if given a sequence of inputs, only a single value is returned. One main drawback in RNN is that for every input considered temporally, a layer is created. This enhanced a very well known problem in Neural Networks: *vanishing gradient*, which avoids the less recent layers to be updated by the contributions at the new ones. Several techniques can be used to solve this issue; one of these is using Long-Short Term Memory (LSTM) layers [19], which can discriminate between updates to be kept and ones to be thrown away.

III. Model

In this study, we propose a novel model, Temporal Edge Embedding Neural Network (TEEN), a Semi-supervised End-2-End neural network for binary classification. It can compute the edge latent representation at every time step considered in the dynamic graph and to use these edge embeddings to extract temporal patterns and to predict the presence of a connection between two nodes in the next graph snapshot.

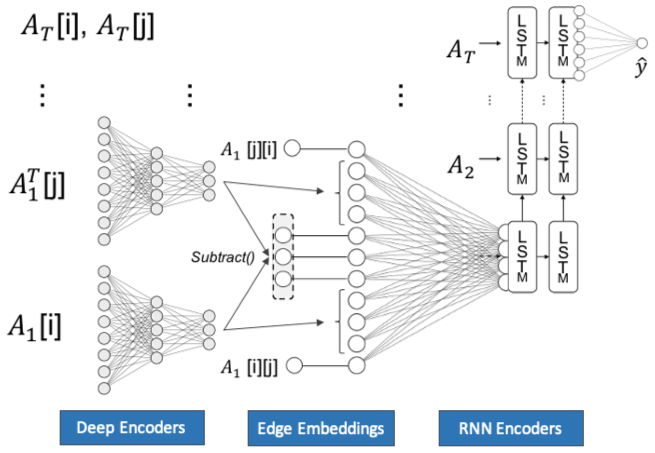

Fig. 2. TEEN Model Architecture

Aspects were taken into account while designing the models are as follows: (1) support of repeating links (2) objective function focused on label prediction (3) explicit presence of link time series captured by the model. Repeating Link Support (1) is a problem for all methods relying on static embeddings as they only exploit graph topological features instead of the temporal ones. Also, (1) is not taken into account by matrix alignment techniques [10], [17] working only for growing graphs. Secondly, having an object function focused on correct label prediction is something missing in many SOTA techniques, as these optimize the reconstruction of the complete adjacency matrix. However, this optimization works bad for repeating links and in general, it would not work well in some cases (e.g., graphs with large number of nodes or with high fraction of vanishing links). Aspect (3) refers to the need for the model of receiving an input of edge time series. Other baselines explored in this work are never given explicit information of how a certain link varies time by time, such as a vector $v = [e_ij^1, e_ij^2, \ldots, e_ij^{T-1}, e_ij^T]$ containing the time series of the link weight at every time step. Instead, these baselines need to extract automatically such information and the temporal patterns associated with it from the adjacency matrix given. We believe that, in the case of repeating link prediction, giving such time-series information explicitly is very helpful and that is why we consider this aspect when designing TEEN.

A. Model Architecture

The model is shown in Figure 2 is made by 3 deep-learning architectures: Deep Encoders, Edge Embeddings formation and RNN Encoders, to which it follows the final label prediction. Every piece of the model has its role in the architecture that helps it to make better predictions. The two of Node Encoders' receives the 2 nodes' neighborhoods (the row of the graph adjacency matrix A^t) into the model at every time step. Thus, the input of the model is of shape $(B, N \times 2, T)$, where B is the mini-batch size, and T is the number of time steps used to make the prediction. The 2 neighborhoods are projected onto a subspace with the use of T *Deep Encoders* made of 3 layers each: the first of shape $(B, D \times 4)$, the second $(B, D \times 2)$, the third (B, D), where D is the embedding dimension, and it is a hyperparameter to be tuned depending on the data. After computing the latent representation of each node, we get their difference and those are concatenated together with the single embeddings and to the two edge values: $A_t[i][j]$ for the edge $i \rightarrow j$, $A_t[j][i]$ for the inverse edge $j \rightarrow i$, both at that specific time step considered. Overall, an edge latent space is formed of dimension $D \times 3 + 2$. Note that, if the graph is weighted, the edge value becomes the weight itself. In our case, since our dataset is a weighted direct graph of transactions, we concatenate the amount of money exchanged between the parties properly standardized with the following min-max standardization:

$$x' = \frac{x - min(x)}{max(x) - min(x)} \tag{1}$$

The subtraction procedure has the goal of representing the First-Order proximity between the 2 nodes representations. The smaller the 2 embeddings difference is, the closer the 2 nodes are in the embedding space. This difference is also minimized in the model loss function Eq. (2) described in Section III-B. The idea is taken from Laplacian Eigenmaps [20] and it was also employed in another Deep Learning algorithm for link prediction [14]. To the edge embedding concatenation, it follows the RNN Encoders for each of the T concatenations capturing important temporal relationships between each time-steps. The RNN Encoder is preferred to a single RNN because it reduces further the dimension of the inputs used by the model turning out in generalization. We used 2 layers, each of size $(B, T, D \times 3 + 2)$ and (B, T, D) from which is output a single latent representation of size D of the edge at the next step $T+1$. Finally, the output predictor provides for the label prediction. Being the goal of the model to predict *one* label for the presence of a link in the model, the natural choice for that model activation function was to use a single neuron with sigmoid activation function outputting the edge presence probability. The link presence is predicted if this probability y_{ij} is at least 0.5.

B. Loss Function

The loss function to be optimized is made of 2 components: the label prediction loss L_{cr}, computed using *binary crossentropy* which maximizes the correct labeling, and an unsupervised component L_{latent} modeling the First-Order proximity between the pair of nodes at every time step, penalizing the big distances between their latent representation in case of a present connection [14]. Thus, the loss to be minimized is:

$$L_{tot} = \alpha L_{cr} + (1 - \alpha) L_{latent} \tag{2}$$

With $0 \leq \alpha \leq 1$ is an hyperparameter weighting the contribution of each component. In particular, we have:

$$L_{latent} = \frac{1}{|E|} \sum_{t=1}^{T} \sum_{i,j \in E} e_{i,j}^t \cdot ||X_i^t - X_j^t||_2^2 \tag{3}$$

TABLE I
GRAPH DATASET INFORMATION

	Dataset A	**Dataset B**	**Dataset C**
#Nodes	∼4000	∼5000	∼1000
#Edges	∼50000	∼50000	∼10000
Avg Degrees	13	12	12
Diameter	9	10	9
Clustering Coeff.	0.127	0.108	0.169
Avg Path Length	3.419	3.634	3.039
Density	0.004	0.003	0.011
Market Share	Low-Mid	Mid	Mid-High
Pos/Neg Balance	∼60%	∼10%	∼50%

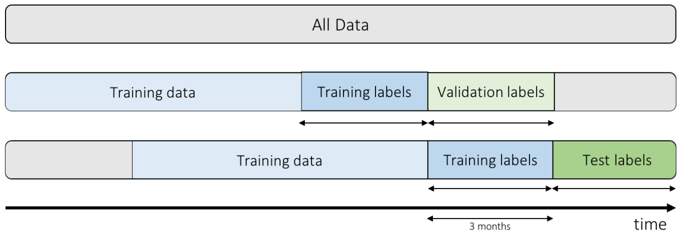

Fig. 3. Dataset split following a time-series settlement. Training data is split into T graph snaphots of time period 3 months each

$$L_{cr} = \frac{1}{|E|}\sum_{i,j \in E} y_{ij}log(p(y_{ij})) - (1-y_{ij})log(1-p(y_{ij})) \quad (4)$$

Here $E = E^1 \cup E^2 \cup E^3 \cdots \cup E^T$ i.e., the set of all existing edges in the graph G up to time T. In L_{latent}, $e^t_{i,j}$ is the label for the presence of an edge between nodes i and j at time t and its value is 1 when the edge is present else 0, X^t_i and X^t_j are the embedding vectors respectively of nodes i and j at time t. In L_{cr}, $p(y)$ is the predicted probability for the presence of the edge, $y_{i,j}$ is equal to $e^{T+1}_{i,j}$ i.e., the ground truth label for the presence of link (i,j) at future step $T+1$.

The model is trained using the traditional backpropagation procedure of neural networks. Stochastic Gradient Descent with Adam optimizer [21] is adopted. Being the model built in an *End-2-End* fashion, the gradient is propagated to all the backward layers in the network.

IV. EVALUATION

A. Dataset

We run our experiments on three transactional datasets provided by a European bank. The networks are divided into graph snapshots G_t, each represented as a Directed Weighted Graph $G = (V, E, W)$, where V is the set of nodes (clients), E is the set of edges (financial relationships) and W is the weight (amount of money exchanged) for every occurring edge. In particular, such weights are number $w_{ij} \in \mathbb{R}$. The datasets are belonging to corporate clients (i.e., companies) and their commercial relationship (supplier and buyer relationships). In the case of multiple transactions within the designed time of split for the graph snapshots, the weight becomes the sum of all these amounts. Each country has its stand-alone dataset. The business logic applied to every dataset as preprocessing is as follows. (1) all the amount of money exchanges are converted from local country currency into *EUR*; (2) transactions having amounts lower than 100 EUR each is filtered out. This action is suggested is reasonable as we are dealing with corporate transactions whose amounts are higher than retail bank transactions); (3) edges appeared only once in 4 years time window are filtered out; (4) clients having less than 3 counterparts are filtered out.

The main statistics for every table are shown in Table I. Because of compliance reasons, some of the numerical values are *approximative*. In the upper part, there is typical graph dataset information such as #Nodes, #Edges, and the density of the graph. The last two values are *market share*, which represents the share of the bank in the country from where the dataset is taken, and the balance of positive/negative class samples. The market share is an indicator for the fraction of the total edges in that country are present in our dataset, indeed, the only relationship having at least *one counterpart* as a client is registered in the databases of the bank. The total transactions between corporates in that country are not visible to a single bank's dataset. The balance of positive/negative class represents the variability of the network: the bigger is this percentage, the more are the clients maintaining stable relationships. In our datasets, such stability is around 50% except for Dataset B where it is 10% and there is a data imbalance over negative samples. This problem has been corrected using Keras *sample weights* technique [22]. Regarding the connectivity, the graph is weakly connected. In general, lots of strongly connected components are present suggesting that most of the time, the relationship is directional. For this reason, the links that will be evaluated are directional and the bad prediction for an inverse-direction link is regularly penalized as a non-existing edge.

B. Data Split

The dataset is split according to a traditional time-series problem. Firstly, the network is divided into multiple graph snapshots, each of a period of a *quarter* (3 months length). Using a quarter when analyzing bank transactions is reasonable from business sense since taxes are generally paid every 3 months. Moreover, this was a choice also performed in other papers related to link prediction over transactional data [23]. Then, we used $T = 8$ quarters for training and the next quarter for testing. As shown in figure 3, when fitting the algorithm an additional snapshot is required. When evaluating the model, a moving window approach is used such that 8 snapshots are used as input data and the next for assessing the results of the prediction. A clearer picture of this process is visible in Figure 3.

C. Evaluation Metrics

Given that the focus of this paper is on repeating link prediction task, we evaluate the model performances with binary classification methods, as explained in Section I. The metrics used for the evaluation are AUC and F1-Score. A difference must be noticed when comparing new and repeating edges as targets for link prediction. When evaluating *new links*, the samples to assess can be any missing link in the previous step. Given the sparsity of the networks evaluated, target edges to be evaluated can be up to $\Theta(|V|^2)$. Moreover, of those candidate pairs of nodes, only a few percentages would develop a connection and it enhances the well-known imbalanced dataset problem. In this case, usually, it is selected only a relevant subset of data such that there is a balance between the number of positive and negative class samples [17]. When evaluating on repeating links, the abovementioned problem does not subsist and the target pairs to be considered are determined to look at existing edges from time $t = 1$ up to timestep $t = T$. Moreover, the imbalanced dataset problem becomes an issue only if the dataset is highly variable, which is not our case except for dataset B as shown in I. For this reason, we are also going to evaluate *Prediction* and *Recall* for the positive class as metrics exclusively for Dataset B.

D. Hyperparameter Tuning

Hyperparameter tuning is a fundamental process to be managed to reach good results when using Deep Learning algorithms. Given the presence of many works related to ours, some of which containing the same architectures, we start following the suggestions present in the literature on papers [12], [18]. Then, we find the best parameters using a *grid search* method on the validation set. The final results are instead related to the test scores as explained in 3. As hyperparameters, we decided to not include the number of neurons at each layer but fix a configuration where the number of neurons is proportional to the dimension of the embedding chosen (detailed in Table II).

V. Results

In this section, we report the results obtained on the evaluation of the repeating link prediction task over the three datasets discussed in Section IV. We compare the performance of our solution model TEEN with 5 SOTA models, 3 Static LP techniques and a simple unsupervised baseline. The section also shows results in terms of the number of parameters used by the models and training and prediction time.

A. Experimental Results

The empirical results on Table II show that TEEN can outperform all the other models by a significant margin on the three datasets of study both on AUC and on F1-Score (indicated as F1 in the table). In this section, we focus more deeply on TEEN and we analyze its training process for every dataset of study in terms of hyperparameter choice and on metrics. We will complement the analysis showing some figures to understand the evolution of the metrics during the validation phase.

We plot the performance obtained over 16 epochs for the validation set in terms of AUC in figure 4. The highest AUC is obtained by dataset B with a score of 0.9252, it follows dataset A obtains an AUC of 0.8865 and dataset C with AUC

TABLE II
RESULTS OVER OUR THREE TRANSACTIONAL DATASETS

	Dataset A		**Dataset B**		**Dataset C**	
Approach	AUC	F1	AUC	F1	AUC	F1
Common Neighbors	0.521	0.523	0.513	0.518	0.543	0.544
Node2Vec	0.579	0.539	0.679	0.520	0.593	0.521
HOPE	0.626	0.561	0.615	0.502	0.621	0.555
SDNE	0.546	0.528	0.607	0.512	0.560	0.520
TNodeEmbed	0.632	0.572	0.812	0.522	0.641	0.564
DynAE	0.732	0.681	0.799	0.701.	0.716	0.678
DynRNN	0.739	0.685	0.798	0.690	0.725	0.679
DynAERNN	0.793	0.705	0.912	0.727	0.803	0.701
DDNE	0.752	0.718	0.841	0.789	0.745	0.726
TEEN	**0.874**	**0.795**	**0.963**	**0.855**	**0.859**	**0.786**

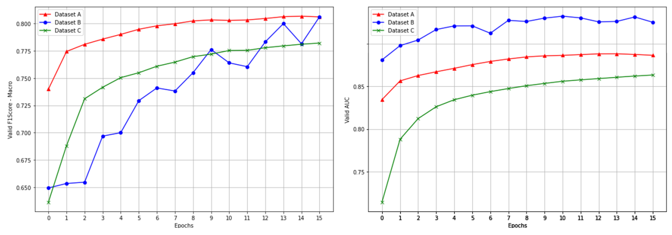

Fig. 4. AUC (on the left) F1-Score macro avg (on the right) for all datasets

of 0.8633. Regarding the F1-Score, we present the results in figure 4. Here we obtain the same score for datasets A and B (0.8058) and a slightly lower score for dataset C (0.7819). Overall, we find out that good results are reached almost immediately after a few epochs. Since we give the model the explicit time series of link occurrence at every time step, it quickly learns the easy patterns depending on that, for instance, links always present are likely to remain during the next step also. In the epochs later, the features provided by the embeddings come into play helping the model improving its performance further. We notice that AUC is constantly growing, instead, F1-Score is likely to fluctuate more. As mentioned before, this is because of the *hard scores* used to calculated this metric, which is very sensitive to small changes of direction of model output probabilities.

The fluctuation observed in the plots is motivated by the fact that the architecture struggles more in searching for equilibrium to minimize the total loss function.

As mentioned earlier in Section IV, we study precision and recall for the positive class samples on dataset B and plot their values on the first 16 epochs in figure 5. Using *class_weights*

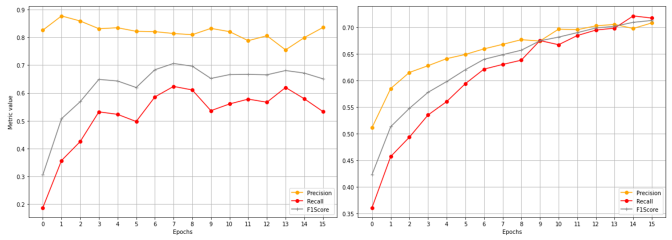

Fig. 5. Precision, Recall and F1-Score for Dataset B using class_weight (on the right) and without (on the left)

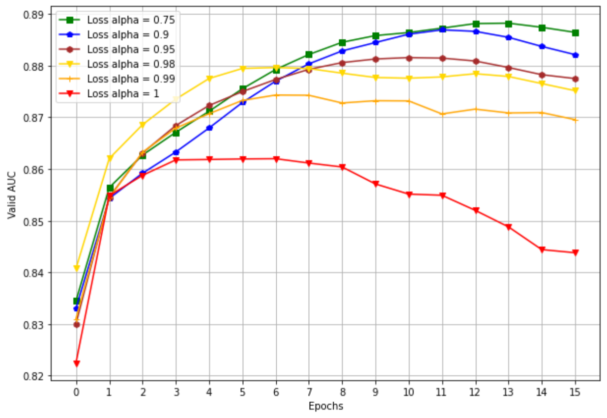

Fig. 6. AUC score at different α_{latent} values (Dataset A)

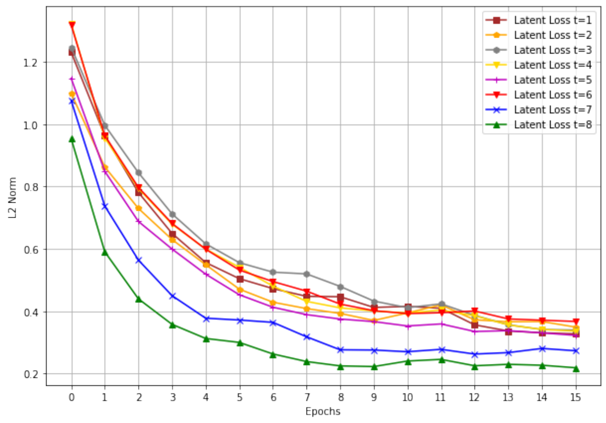

Fig. 7. Latent loss for every timestep (Dataset A)

parameter shows good results allowing us to obtain a balanced score for Precision, Recall, and F1-Score on the positive class samples.

B. *Effect of Latent Loss on Model Performance*

In this section, we investigate how the contribution of the latent loss on the total objective function impacts the results. In figure 6 we compare the AUC scores obtained on dataset A on different α values. We remind that α is the parameter controlling the balancing the contribution of the binary cross-entropy and the latent losses on the total objective function, as described in Section III-B. From the plotted graph, it is possible to notice the regularizing effect of the latent losses: the higher is the alpha, the smaller is the regularizing effect and the algorithm reaches a non-optimal peak quicker. Instead, if alpha decreases, the algorithm can reach better results with more epochs. In other words, the value $1 - \alpha$ is the regularization term. In the extreme cases of $\alpha = 1$ and $\alpha = 0.75$ we notice that the latter configuration provides an increment of 3% on both macro F1-Score and AUC. The score is calculated concerning the respective peak scores.

It is also interesting to assess the evolution of the latent losses on our algorithm. As described in Section III-B, we employ a latent loss for every time step evaluating the distance between the node pair for T times (or lookback). In our dataset, we consider the lookback of 8-time steps. In Figure 7 we plot the validation L_{latent} value for every time step, ordered from the oldest (*latent_loss_1*) to the most recent (*latent_loss_8*) and we see how they evolve over 16 epochs of training. It is possible to notice that the newer are the components, the smaller is their value, suggesting a bigger correlation between the connections present at recent timesteps with the one to be predicted. Interestingly, we see that this rule is not valid for *latent_loss_1* and *latent_loss_5*, which are closer. This is reasonable as, in our dataset, we consider units of three months each, hence, *latent_loss_1* and *latent_loss_5* corresponds to the same quarter of the one to be evaluated and the algorithm understands such correlation.

TABLE III
HYPERPARAMETERS CHOICE IN THE FINAL MODEL CONFIGURATION

	Dataset A	**Dataset B**	**Dataset C**
Learning rate	1e-4	8e-5	6e-4
Embedding Dimension	128	200	50
$Alpha_{latent}$	0.75	0.95	0.85
$Alpha_{ELU}$	1.0	1.0	1.0
Lookback	8	8	8
Batch size	100	100	100
Optimizer	Adam	Adam	Adam

C. *Effect of Embedding Dimension on Model Performance*

The embedding dimension is an important hyperparameter that affects consistently the performance of the algorithm. To decide for its best value, we have to take into account at the same time the dimension of the input (which depends on the number of nodes in the graph considered), the number of parameters introduced in the model, and the generalization properties we want the architecture to preserve. In general, the bigger is the number of embedding neurons, the easier it is to reconstruct the node neighborhood given as input from the latent space. At the same time, the bigger is the embedding, the more are the number of parameters and the worse is the generalization of newly seen inputs. We validate the dimension of the embedding chosen using grid search; finally, we arrive at the values listed in III. Overall, we notice that when embedding dimensions are increased too much, the algorithm has difficulty to reach high performances showing instability and overfitting. Instead, when it is too low, it overgeneralizes reaching the peak and dropping quickly. In figure 8 we plot the AUC score obtained on dataset A at different embedding values. The best configuration is between 80 and 100, in the middle, such that the extracted embeddings can capture the role of the nodes in the graph.

D. *Other results*

In Table IV we present the number of embedding dimensions, parameters, and hyperparameters adopted for the Deep Learning-based models adopted in the evaluation. Overall, we notice that TEEN presents a high number of parameters,

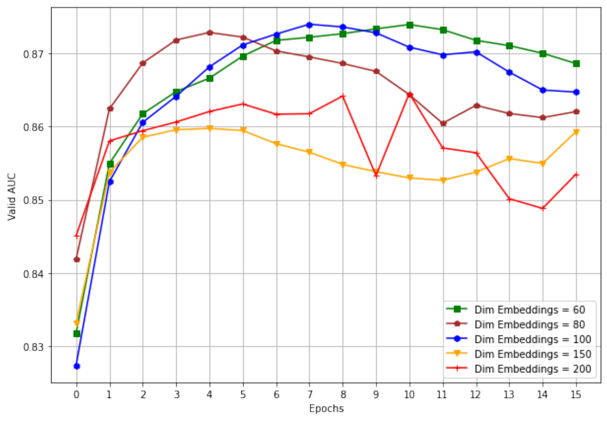

Fig. 8. AUC score for different *Embedding Dimensions* (Dataset A)

TABLE IV
COMPASION OF THE MODEL ON ADDITIONAL METRICS (DATASET B)

	Dim. Emb.	**# Train Params.**	**# Hyperparams**
TNodeEmbed	150	225,901	6
DynAE	256	4,899,896	10
DynRNN	256	**96,807,440**	10
DynAERNN	256	24,324,056	10
DDNE	150	36,428,301	9
TEEN	150	44,201,147	8

mainly due to the pair of nodes encoder structure. The most expensive model in terms of trainable parameters is DynRNN; as also stated in its paper [18], the algorithm suffers from the high number of parameters as received in input the adjacency vector of the node directly. Moreover, we observe that our model, similarly to TNodeEmbed and DDNE, uses a smaller embedding dimension as they do not need to perform the final *adjacency reconstruction* as the other methods [11]. We decided to include the Embedding dimension as it is strongly connected to the size of the architecture, thus, influencing the number of trainable parameters. A better description of how the two are influenced is explained in Section III-A. The number of hyperparameters shown here gives an intuition on how difficult it is to perform fine-tuning for each algorithm. All the models must deal with the following 6 hyperparameters: the number of epochs, learning rate, embedding dimension, lookback, batch size, and optimizer. The difference is in terms of *regularization* and encoder-decoder size. In TNodeEmbed we have the lowest number of hyperparameters due to the simple architecture; DynAE, DynRNN and DynAERNN have the maximum number hyperparameters; our model has 8 hyperparameters including the above-mentioned 6 and the following 2: α_{latent}, controlling the contribution of latent loss in the objective function and α_{elu}. On average, TEEN provides a *lower* number of hyperparameters thanks to the latent loss regularization which makes useless other types of regularization such as dropout or l1-norm and l2-norm, which were explored in TEEN but did not show any improvement on its performances.

VI. RELATED WORK

The techniques to solve the Dynamic Link Prediction problem are numerous: on a recent survey of Divakaran et al [17], the authors identify 8 different classes of algorithms. However, only some of the methods of studying these approaches are valuable to mention in this background section because some of them focus only on the new link prediction task. One of them is TNodeEmbed, [10], where the authors employ static embedding nodes in sequential temporal order having the goal of enriching the node representation through time and performing the prediction of the link. The algorithm works in the following way. Firstly, the embedding is computed for every node at every time step. Secondly, the output vectors are *aligned* in pairs of 2 consecutive embeddings applying the *Orthogonal Procrustes* technique [24], an algebraic linear transformation. Finally, the authors employ the output from the application of Orthogonal Procrustes of each node into a Recurrent Neural Network (RNN) model, in particular, a LSTM multi-layered model using embedding to infer the link prediction at the next step.

Important algorithms relevant for this paper are the ones presented in the DynamicGEM framework [11]. It comprises the implementation of three relevant deep learning techniques for dynamic graph analysis. The first architecture introduced in [18] is dynamic2vecAE, where Deep-AutoEncoders are employed on every static graph of the temporal snapshots G_t, then, every output embedding graph representation ${Y^K}_t$ is connected one sequentially such that the new task becomes not the reconstruction, but the prediction of the following step adjacency matrix. For this purpose, a new parameter l is introduced, the *loopback*, which is the number of time steps to take into account for the training process of the algorithm (how long is the chain of interconnected graph embeddings). One limitation of dynamic2vecAE is the incredibly high number of parameters employed, which is $O(nld^k)$, with $n = |V|$ the number of vertices, l the loopback and d^k the number of neurons for the embedding layer. k is the number of the layer where embedding is created. Secondly, the authors adopt a RNN to solve this issue with the model named dynamic2vecRNN. Every node representation is passed through multiple RNN and encoded in a lower-dimension space, and then decoded with other multiple LSTM gates. Even applying the RNN, still, the model has a huge number of parameters, mainly because the representation of the node neighborhood is a sparse vector of dimension n. The third architecture solves this issue: the authors introduce dynamic2vecAERNN where the inputs are now the embeddings computed with the deep-autoencoders model.

Other important works in the area of Network Representation Learning are [25] and [26]. In the first, the authors build a 4-step framework for link prediction: Graph Embedding, Manifold alignment, Trajectory Prediction, Graph Reconstruction, to which follows the evaluation on the link prediction task (both for repeating links and new links). The alignment is performed with the Orthogonal Procrustes procedure [24] also employed in TNodeEmbed [10]. In the second [26], the authors provide a distinction between the formation of new edges with the dissolution of the present ones at the previous step, identifying two distinct objective functions which are computed based on the different link formation and link dissolution prediction task.

VII. Conclusion and Future Work

In this work, we have proposed TEEN, a novel method of repeating link prediction over dynamic graphs using Deep Learning. TEEN includes a number of improvements concerning the other SOTA approaches considered. The *edge embedding* representation turned out to play an important role. Firstly, node embeddings are learned independently from each other using two deep encoders to preserve their role in the graph. Secondly, these are concatenated together with their subtraction and with the direct and inverse edge values. We introduced an edge value concatenation which makes possible to the model understanding the temporal patterns present in the time series. Another improvement our work offers is in the *latent loss*, which is an additional loss function optimizing for the proximity of the two nodes latent representations at a certain time step t. We validated the impact of latent loss on the model and obtained a notably good result, as shown in Figure 6, observing how increasing the weighting of such loss acted as a form of *regularization*. Besides very encouraging results, some limitations are present, such as the *scalability* issue of the model that was not addressed in this work, also due to the number of distinct nodes in scope were only *few thousands*. We plan to address the scalability issue in the future work, by streamlining the architecture and reducing the number of trainable parameters.

Further future work includes extending of the study on multiple datasets, multiple baselines to compare, and a deeper exploration of the hyperparameters. An interesting direction for future work would be to test our model also on new edges: we believe that the improvements over the pair proximity would be able to achieve interesting results compared to other baselines. Another promising direction for the this work is on improvement of the node embeddings' quality in the graph. In our model, we did not use any additional attributes for the nodes even though many advanced embedding techniques are using a vector of features for every node to do it [27]–[29]. Some of these features can also be extracted by the graph itself; in TEEN we tried to do it creating two different embeddings for a node: the first associated with the source node and the second for the target node, as shown in figure 2. Furthermore, we plan to explore *context sensitive* GRL and extract multiple embedding for the same node relative to the multiple roles it has. The idea, as explained in the paper of Kefato et al [30], improved the performance in many graph analytics tasks. We believe that, especially for transactional datasets, computing context-sensitive embeddings would be beneficial and would allow us to improve the *explainability* of the model, for instance, being able to distinguish the features associated to the different roles of each node.

References

[1] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," *Journal of the American society for information science and technology*, vol. 58, no. 7, pp. 1019–1031, 2007.

[2] M. Al Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," in *SDM06: workshop on link analysis, counter-terrorism and security*, vol. 30, 2006, pp. 798–805.

[3] X. Yu, Q. Gu, M. Zhou, and J. Han, "Citation prediction in heterogeneous bibliographic networks." SIAM, 2012.

[4] Y. Yang, R. N. Lichtenwalter, and N. V. Chawla, "Evaluating link prediction methods," Oct 2014. [Online]. Available: http://dx.doi.org/10.1007/s10115-014-0789-0

[5] T. Tylenda, R. Angelova, and S. Bedathur, "Towards time-aware link prediction in evolving social networks," 06 2009.

[6] S. Oyama, K. Hayashi, and H. Kashima, "Cross-temporal link prediction," in *2011 IEEE 11th International Conference on Data Mining*. IEEE, 2011, pp. 1188–1193.

[7] M. LANKESHWARA *et al.*, "Time-aware methods for link prediction in social networks," 2013.

[8] A. Patel, J. Agrawal, and S. Sharma, "Link prediction-based multi-label classification on networked data using apriori algorithm."

[9] R. Hisano, "Semi-supervised graph embedding approach to dynamic link prediction," in *International Workshop on Complex Networks*. Springer, 2018, pp. 109–121.

[10] U. Singer, I. Guy, and K. Radinsky, "Node embedding over temporal graphs," 03 2019.

[11] P. Goyal, S. R. Chhetri, N. Mehrabi, E. Ferrara, and A. Canedo, "Dynamicgem: A library for dynamic graph embedding methods," *arXiv preprint arXiv:1811.10734*, 2018.

[12] T. Li, J. Zhang, P. S. Yu, Y. Zhang, and Y. Yan, "Deep dynamic network embedding for link prediction," *IEEE Access*, vol. 6, 2018.

[13] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," vol. 2016, 07 2016, pp. 855–864.

[14] D. Wang, P. Cui, and W. Zhu, "Structural deep network embedding," 2016. [Online]. Available: https://doi.org/10.1145/2939672.2939753

[15] M. Ou, P. Cui, J. Pei, Z. Zhang, and W. Zhu, "Asymmetric transitivity preserving graph embedding," p. 1105–1114, 2016. [Online]. Available: https://doi.org/10.1145/2939672.2939751

[16] M. Hasan, V. Chaoji, S. Salem, and M. Zaki, "Link prediction using supervised learning," 01 2006.

[17] A. Divakaran and A. Mohan, "Temporal link prediction: A survey," *New Generation Computing*, 2019. [Online]. Available: https://doi.org/10.1007/s00354-019-00065-z

[18] P. Goyal, S. R. Chhetri, and A. Canedo, "dyngraph2vec: Capturing network dynamics using dynamic graph representation learning," *Knowledge-Based Systems*, 2020. [Online]. Available: http://dx.doi.org/10.1016/j.knosys.2019.06.024

[19] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, p. 1735–1780, Nov. 1997.

[20] M. Belkin and P. Niyogi, "Laplacian eigenmaps for dimensionality reduction and data representation," *Neural Comput.*, vol. 15, no. 6, 2003.

[21] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2014.

[22] F. Chollet *et al.*, "Keras," 2015.

[23] V. Shumovskaia, K. Fedyanin, I. Sukharev, D. Berestnev, and M. Panov, "Linking bank clients using graph neural networks powered by rich transactional data," 2020.

[24] C. Wang and S. Mahadevan, "Manifold alignment using procrustes analysis," p. 1120–1127, 2008. [Online]. Available: https://doi.org/10.1145/1390156.1390297

[25] C. Fang, M. Kohram, X. Meng, and A. L. Ralescu, "Graph embedding framework for link prediction and vertex behavior modeling in temporal social networks," 2011.

[26] R. Hisano, "Semi-supervised graph embedding approach to dynamic link prediction," pp. 109–121, 2018.

[27] W. Hamilton, R. Ying, and J. Leskovec, "Inductive representation learning on large graphs," 06 2017.

[28] Z. T. Kefato and S. Girdzijauskas, "Graph neighborhood attentive pooling," 2020.

[29] M. Zhang and Y. Chen, "Link prediction based on graph neural networks," *arXiv preprint arXiv:1802.09691*, 2018.

[30] Z. T. Kefato and S. Girdzijauskas, "Gossip and attend: Context-sensitive graph representation learning," 2020.