BENCHMARKING GRAPH NEURAL NETWORKS ON DYNAMIC LINK PREDICTION

Anonymous authors
Paper under double-blind review

ABSTRACT

Graph neural networks (GNNs) are rapidly becoming the dominant way to learn on graph-structured data. Link prediction is a near-universal benchmark for new GNN models. Many advanced models such as Dynamic graph neural networks (DGNNs) specifically target dynamic link prediction. However, these models, particularly DGNNs, are rarely compared to each other or existing heuristics. Different works evaluate their models in different ways, thus one cannot compare evaluation metrics directly. Motivated by this, we perform a comprehensive comparison study. We compare link prediction heuristics, GNNs, discrete DGNNs, and continuous DGNNs on dynamic link prediction. We find that simple link prediction heuristics often perform better than GNNs and DGNNs, different sliding window sizes greatly affect performance, and of all examined graph neural networks, that DGNNs consistently outperform static GNNs.

1 INTRODUCTION

Recently, there has been a drive to enable fair comparisons and benchmarks of GNNs (Errica et al., 2020; Dwivedi et al., 2020; Hu et al., 2020). The drive was in part due to a lack of common practice in the validation and testing of GNNs and concerns around replicability and reproducibility. Reproducibility is a problem for the wider field of machine learning (Lipton & Steinhardt, 2019) and even science in general (Sciences et al., 2019) making it difficult to identify actual scientific advances.

In the space of dynamic graph neural networks (DGNNs) these problems are further exacerbated by (i) the dynamic nature of the data, (ii) the lack of common terminology (Holme, 2015), (iii) the lack of established strong baselines (most works don't compare performance to other DGNNs), (iv) the divide between discrete and continuous DGNNs and (v) the wide array of experimental design choices. Among the choices are: how to represent the dynamic network (e.g. snapshot, time-windows, continuous, time-to-live of edges, etc.), which node features to include, how to split the data into train-validation-test sets, which metrics to use to evaluate the results, how to use negative sampling rate in reported metrics, and how to choose/optimize neural network parameters (e.g. learning rate, early stopping criterion, embedding space dimensions, etc.). All of this means that comparing the performance of methods by reading research papers is not possible unless they clearly state all their design choices and those design choices are identical between papers.

DGNNs are a promising avenue in modeling network dynamics because of their ability to encode both spatial patterns through GNNs and temporal patterns through time-series components (e.g. recurrent neural networks (RNN) or self-attention). However, the so far proposed DGNNs have been tested on few datasets and are rarely compared to other DGNNs. Different studies compare the methods on different datasets as there is no consensus when it comes to which datasets to use in DGNN benchmarking.

To address this problem we aim to perform a fair and comprehensive comparison of GNN methods on the dynamic link prediction task. We benchmark each GNN and DGNN using the same experiment design, including the same strategy for optimizing hyperparameters (i.e. grid search).

Previous works in the DGNN space focus on presenting a new architecture. These models are either discrete or continuous DGNNs. However, these types of models are not compared to each other. Discrete models operate on discrete network representations while continuous models operate on continuous representations. A comparison between these kinds of models is not possible unless they are evaluated on the same network representation. We present a framework that enables this comparison. The framework represents the datasets as both discrete and continuous while evaluating the predictions identically. This is achieved by transferring the continuous DGNNs to the discrete domain. In short, this is done by training them separately on the continuous representation, then training a decoder using the continuous node embeddings on the discrete representation.

Using our new framework we compare link prediction heuristics, static GNNs, discrete DGNNs, and continuous DGNNs. This is therefore not just a comparison of different models, but a comparison of different kinds of models. Our aim is to give an indication of which GNNs are best capable of encoding dynamic network topology and indeed if they are better than well-established heuristics commonly used in the network science community for link prediction. To the best of our knowledge, this is the first comparison of discrete and continuous DGNNs.

Another important question is whether DGNNs are better than traditional GNNs at encoding dynamic graph structure. There are multiple ways that dynamic networks can be represented as static networks, thus allowing GNNs to encode dynamic networks. Previous works compare DGNNs to GNNs and their results indicate that DGNNs do indeed perform better than GNNs (Pareja et al., 2020; Chen et al., 2021; Xu et al., 2020; Rossi et al., 2020). However, these works use only one of the many ways of converting dynamic networks to static ones. Without exploring these options it is still uncertain whether DGNNs tend to outperform GNNs on dynamic network encoding. To gain more insight, we use three different ways of aggregating dynamic networks to static networks, or five if counting different sliding window sizes. These are explained in Section A.4.1. To thoroughly explore these options we consider the aggregation options as a hyperparameter and include it in our grid search. We also analyze the importance of this hyperparameter in Section 4.2.

Our primary contribution is a fair comparison of graph neural networks and link prediction heuristics on dynamic link prediction. To this end, we introduce a framework that can train different types of GNNs; static GNNs, discrete DGNNs, and continuous DGNNs. To enable reproduction of our results and facilitate future work we publicly release our code and configuration files[1].

2 BACKGROUND

2.1 DYNAMIC NETWORKS

Dynamic network terminology has yet to converge. Networks, where edges and nodes may appear or disappear over time, go by many names in the literature (Holme, 2015). Here we will refer to these networks as dynamic networks. There are several kinds of dynamic networks and these may also go by different names. In this work, we adopt the dynamic network cube terminology for dynamic networks, a conceptual framework that groups dynamic networks along three dimensions and enables more precise terminology (Skarding et al., 2021).

Definition 1 (Dynamic network) a dynamic network is a graph $G = (V, E)$ where: $V = \{(v, t_s, t_e)\}$, with v being a vertex of the graph and t_s, t_e are respectively the start and end timestamps for the existence of the vertex (with $\mathrm{t_s} \leq \mathrm{t_e}$). $E = \{(u, v, t_s, t_e)\}$, with $u, v \in V$ and t_s, t_e are respectively the start and end timestamps for the existence of the edge (with $\mathrm{t_s} \leq \mathrm{t_e}$).

Temporal granularity refers to how coarse or fine-grained a network representation is (Skarding et al., 2021). In order of increasingly fine-grained, we have static, discrete, and continuous networks. Static networks are networks with no information of time.

Dynamic networks can also be distinguished by the link duration spectrum (Skarding et al., 2021). Evolving networks are characterized by links persisting for longer, while links in temporal networks are ephemeral. An interaction network is a type of temporal network where links have no duration.

Discrete representations are ordered sets of static graphs, examples of such representations include snapshots. A discrete network representation is an ordered set of graphs,

$$DG = \{G^1, G^2, \dots, G^T\}, \tag{1}$$

where T is the number of snapshots/time-windows.

[1]Code available at `https://github.com/xkcd1838/bench-DGNN`

Continuous network representations have exact temporal information. They represent the dynamic network as one graph with time stamps on the nodes and/or edges. Examples of such representations include interval graphs (Holme & Saramäki, 2012; Holme, 2015) and graph streams (Zhang, 2010). Since this work uses interaction networks, we use a contact sequence (Holme & Saramäki, 2012) to represent our continuous networks. A contact sequence is a time-ordered list of triplets, where one triplet represents an interaction between two nodes at a given time.

$$CS = \{(u_i, v_i, t_i); i = 1, 2, \ldots\}, \tag{2}$$

where u_i and v_i is the node pair and t_i is the time the nodes interacted.

The distinction between discrete and continuous networks is important because a model, e.g. a DGNN, is made to encode one of the representation forms. A discrete DGNN can only encode networks represented as discrete networks and a continuous DGNN can only encode networks represented as continuous networks.

2.2 Graph neural networks & Link prediction heuristics

Graph neural networks (GNN) have seen a surge in popularity in recent years. GNNs are representation learning models which aim to store a latent representation of a graph structure. GNN models use message passing to aggregate features of neighboring nodes together (Wu et al., 2020). A common output of a GNN layer is node embeddings. Most GNNs can only encode static networks, in this work we refer to these models as static GNNs.

Dynamic graph neural networks (DGNN) is a subclass of GNNs capable of encoding dynamic networks (Skarding et al., 2021)[2]. The two main types of DGNNs are discrete DGNNs and continuous DGNNs which are capable of encoding discrete and continuous networks respectively. Due to the difference in network representation, their architectures are radically different.

Discrete DGNN architectures usually consist of GNN layers and time series layers, with the time series layers being RNN layers or attention layers. Continuous DGNNs on the other hand have more varied architectures. The key challenge for the continuous models is to encode the inter-event time between node interactions. For this, some models (Ma et al., 2020; Kumar et al., 2019) use RNNs, such as a time-aware LSTM (Baytas et al., 2017), other models use temporal point processes (Trivedi et al., 2017, 2019; Knyazev et al., 2021) and the most recently emerged approaches use time embeddings (Kazemi et al., 2019). There are currently only two time embedding based models, TGAT (Xu et al., 2020) and TGN (Rossi et al., 2020). Discrete DGNNs iterate over the data snapshot by snapshot, while continuous DGNNs iterate over the data edge by edge.

Link prediction heuristics are simple methods based on the idea that two nodes are more likely to form links if they have common neighbors. The simplest implementation of this idea is the common neighbor heuristic (Liben-Nowell & Kleinberg, 2007) where the similarity score is given by

$$|\Gamma(u) \cap \Gamma(v)| \tag{3}$$

Where $\Gamma(u)$ is the neighbors of node u. There is a rich literature on these methods in the network science community (Liben-Nowell & Kleinberg, 2007; Martínez et al., 2016).

2.3 Dynamic link prediction

Traditionally, link prediction is the task of predicting links in static networks. Where there is no distinction between predicting missing links and future links. When predicting links in dynamic networks, this distinction is important. Missing link prediction can be referred to as *interpolation* and future link prediction can be referred to as *extrapolation* (Kazemi et al., 2020). In this work, we compare methods on the future link prediction (extrapolation) task on discrete networks.

We predict which links will appear in the next time-window. From the perspective of discrete methods, this is a very natural prediction task. The methods read in time-windows (snapshots) which

[2]There also exist GNN architectures for static networks with dynamic node and/or edge labels. These so-called spatio-temporal graph neural networks are discussed in some GNN surveys (Wu et al., 2020; Zhou et al., 2020)

Table 1: Dataset statistics. Snap density is the average density of the snapshots, Snap size, is the size of the snapshots (time-windows) in days. Num snap is the number of snapshots in the network. Total time is the time period (in days) that the dataset covers.

Dataset	Nodes	Edges	Unique edges	Density	Snap density	Cont. edges	Snap size	Num snap	Total time	Splits
Enron	151	5,780	1,569	0.13854	0.01379	50,572	30	37	1,137	80-10-10
UC Irvine messages	1,899	22,497	13,838	0.00769	0.00014	59,835	2.2	88	193	71-10-19
Bitcoin-OTC	5,881	23,686	21,492	0.00124	0.00001	35,592	14	135	1,903	70-10-20
Autonomous-systems	7,716	583,946	7,796	0.00026	0.00020	2,335,784	1	99	99	70-10-20
Wikipedia	9,227	39,804	18,257	0.00043	0.00003	157,474	1	30	30	70-15-15
Reddit	10,984	307,593	78,516	0.00130	0.00017	672,447	1	30	30	70-15-15

are in chronological order and predict which links appear in the next snapshot. Adapting static and continuous models for dynamic link prediction is more involved. We cover the details on how static, discrete, and continuous models are adapted to dynamic link prediction in Section A.4.

Link prediction can be seen as a special case of dynamic link prediction, where there are only three snapshots; the train, validation, and test snapshots. In that sense, dynamic link prediction is simply an extension of link prediction with multiple snapshots in the train, validation, and test sets.

3 Experimental setup

3.1 Datasets

Table 1 shows statistics of the datasets. We select five continuous interaction networks and one discrete evolving network (Autonomous) as datasets. We chose interaction networks as they allow us to easily convert to more coarse-grained temporal granularities such as discrete networks. Sparser snapshots indicate a greater imbalance between links and non-links, thus making the classification problem harder. Details on the datasets are found in the appendix, Section A.1.

We prepare two versions of each dataset, a directed continuous interaction network and an undirected discrete network. Continuous models encode the continuous network. The static and discrete models encode the discrete network. In the conversion from continuous to discrete, reciprocal edges are added to make the discrete networks undirected. The number of times an edge occurs in a snapshot is added as a weight to the snapshot's edge.

All results are reported predictions on the discrete networks. For continuous models, this is achieved by splitting the continuous parts of the continuous networks into snapshots corresponding to the snapshots in the discrete network. We then let the continuous models encode the continuous network before the target snapshot and then try to predict the link occurrence in the discrete network.

3.2 Methods

For each of the three network categories, static, discrete, and continuous, we select at least two GNNs to benchmark. We select GCN (Kipf & Welling, 2017) and GAT (Veličković et al., 2018) as static models as they are known to be fairly universal and representative. To represent discrete models we select EGCN-H, EGCN-O and GC-LSTM. The EGCN (Pareja et al., 2020) models differ from other DGNN models as they use RNNs to evolve the GCN weights rather than evolve node embeddings. GC-LSTM is selected as it is an integrated DGNN and the architecture was used by Chen et al. (2021) specifically for dynamic link prediction. The GC-LSTM encoder integrates an LSTM and a spectral GCN (Defferrard et al., 2016), it is very similar to the first DGNNs introduced by Seo et al. (2018). To represent continuous models we select two time embedding based continuous DGNNs, TGAT (Xu et al., 2020) and TGN (Rossi et al., 2020). Implementation details are given in the appendix, Section A.3.

To represent link prediction heuristics we select three well established methods (Liben-Nowell & Kleinberg, 2007; Martínez et al., 2016), Common Neighbors (CN), Adamic-Adar (AA) (Adamic & Adar, 2003) and Jaccard. As well as two modern heuristics; Newton's gravitational law (Newton) (Wahid-Ul-Ashraf et al., 2017) and Common Neighbor and Centrality based Parameterized Algorithm (CCPA) (Ahmad et al., 2020). The modern methods use the shortest path between nodes to get scores for missing links beyond common neighbors, whereas the other methods give a similarity score of 0 if there are no common neighbors.

3.3 Task & metrics

The dynamic link prediction task is to give a probability score for each node pair in the network that a link between the two nodes will be created in the next snapshot. The prediction problem is extremely unbalanced; in every selected network (except for Enron) we see more than 10,000 non-links for every link. This leads Yang et al. (2015) to recommend using precision-recall curves when evaluating link prediction. We use mean average precision (mAP) , which is equivalent to the area under the precision-recall curve. We also report the Area under the receiver operating characteristic curve (AUC) as this is commonly done for link prediction methods and mean reciprocal rank (MRR).

The link prediction task becomes harder the more imbalanced the classes are. The class imbalance can be measured through network density. Since we test on snapshots, the mean snapshot density in Table 1 indicates how imbalanced the prediction task on each dataset is. The mAP score was selected in part because it is sensitive to this increased difficulty. We, therefore, expect mAP scores for networks with lower snapshot density to be lower.

This extreme class imbalance makes it tempting to use negative sampling to balance the datasets more when reporting the metrics. However, negative sampling has been shown to lead to a wrong ordering of predictors (Yang et al., 2015). It also greatly affects the mAP metric, so scores with different sampling ratios are not comparable. Examples of this are the TGAT (Xu et al., 2020) and TGN (Rossi et al., 2020) papers which report sampled scores on a 1 to 1 ratio.

3.4 Evaluation scheme

We perform a chronological train-validation-test split. The snapshot size and split sizes are shown in Table 1. All models use the same snapshot sizes and the same train-validation-test splits. We report the results of the test results of the best performing (highest mAP) validation run.

We ensure that all models are trained in the same way by using a common framework that supports models of the three different temporal granularities. Our framework is an extension of the framework used by EvolveGCN (Pareja et al., 2020). Major differences with that framework include adding; (i) two additional training pipelines, one for static and one for continuous models (we use the already existing pipeline for the discrete models), (ii) grid search functionality, (iii) enabling the use of continuous embeddings on discrete network representations, and (iv) link prediction heuristics. The evaluation on the validation and test set is however identical, thus our results are comparable to Pareja et al. (2020). Details on the three training pipelines are found in, Section A.4.

All GNN models use the same decoder (a two level feed forward network) and binary-cross entropy loss. Models are trained by predicting the next snapshot in the training set. For each epoch, the framework iterates through the dynamic network, snapshot by snapshot. For each iteration, the framework feeds the GNN with the network before the current time step and evaluates the predictions of the GNN against the next snapshot. Previous snapshots may be aggregated, but the next snapshot is always of the same size to ensure that the results are comparable. The heuristics don't require any training and are thus run directly on the test set.

The loss function is weighted, giving non-existing links a weight of 0.1 and links a weight of 0.9. Neural network weights are initialized using uniform random initialization. Static and discrete models use a negative sampling ratio of 1 to 100 during training, and continuous models a ratio of 1 to 1. Importantly, we do not use negative sampling when reporting validation and test scores. We use the one-hot encoded node degree as initial node features for static and discrete models. This was shown by Errica et al. (2020) to improve the performance of GNNs on graph classification. This is a key design decision, as it dictates the size of the node embedding. The continuous models are limited to use the same dimensions on node and edge features. Following the original works, we use a node and edge feature size of 172. The node features are randomly initialized and for datasets without edge features (every dataset except Wikipedia and Reddit) we randomly initialize the edge features.

Table 2: Dynamic link prediction mAP scores. The scores reported are the mean and standard deviation (in parenthesis) of mAP scores averaged across four runs with different random seeds. The heuristics are deterministic and thus they all have a standard deviation of 0. The best performances are highlighted in **bold**, and the highest scoring GNN performances are $\underline{\text{underlined}}$. Cells marked by † were not run (see Section A.4.3). Random is completely random predictions. R-embed is predictions when the decoder is given random embeddings.

Models	Enron	UC	Bitcoin	Autonomous	Wikipedia	Reddit
Random	0.003(0.000)	$2 \cdot 10^{-5}$(0.000)	$2 \cdot 10^{-6}$(0.000)	$2 \cdot 10^{-4}$(0.000)	$3 \cdot 10^{-5}$(0.000)	$2 \cdot 10^{-4}$(0.000)
R-embed	0.003(0.000)	$3 \cdot 10^{-5}$(0.000)	$3 \cdot 10^{-6}$(0.000)	0.002(0.001)	$3 \cdot 10^{-5}$(0.000)	$2 \cdot 10^{-4}$(0.000)
CN	0.064	0.007	0.002	**0.616**	0.033	0.144
AA	0.070	0.007	0.004	**0.616**	0.033	0.144
Jaccard	0.060	0.006	0.002	0.130	0.016	0.012
Newton	0.063	0.008	**0.008**	**0.616**	**0.036**	**0.151**
CCPA	0.064	0.008	0.003	**0.616**	0.034	0.144
GCN	0.322(0.022)	0.021(0.004)	0.002(0.002)	0.032(0.006)	$3 \cdot 10^{-5}$(0.000)	0.038(0.007)
GAT	$\underline{\mathbf{0.347}}$(0.020)	0.018(0.007)	0.001(0.001)	0.030(0.017)	$4 \cdot 10^{-4}$(0.001)	0.006(0.005)
EGCN-H	0.329(0.048)	0.014(0.003)	0.001(0.001)	0.182(0.082)	0.004(0.001)	0.038(0.010)
EGCN-O	0.338(0.031)	0.025(0.001)	$\underline{0.003}$(0.001)	0.165(0.033)	0.005(0.001)	0.039(0.005)
GC-LSTM	0.270(0.023)	$\underline{\mathbf{0.047}}$(0.003)	0.002(0.001)	$\underline{0.406}$(0.024)	$\underline{0.007}$(0.001)	0.097(0.007)
TGAT	0.058(0.004)	0.001(0.001)	$2 \cdot 10^{-4}$(0.000)	†	0.006(0.000)	$\underline{0.108}$(0.007)
TGN	0.055(0.009)	0.007(0.001)	$1 \cdot 10^{-4}$(0.000)	†	0.004(0.000)	0.042(0.009)

We search for good hyperparameters by performing a grid search. While being very time-consuming, this allows us to analyze the impact that different time-windows have on training. We then run each model four times with the best found hyperparameters with different seeds.

If the hyperparameter is not included in the grid search, we use the values used by the original authors. Some of the parameters optimized differ between the different types of models. We search the learning rate on all models as it is recommended by Goodfellow et al. (2016) as an important hyperparameter to optimize. For static GNNs and discrete DGNNs we chose to search the hidden layer size to regulate the number of parameters of the model as too few might lead to underfitting and too many might lead to overfitting. For further details on the hyperparameters and the grid search see Section A.5.

4 Results and Discussion

4.1 Model comparison

Table 2 shows the mAP scores from our experiment. We also report the AUC scores (Table 5) and MRR scores (Table 6). All results are the average scores taken from four runs of the best parameter setting found by the grid search. The selected parameter settings are reported in Table 9.

How do link prediction heuristics and GNNs compare? On most datasets the heuristic baselines outperformed the GNNs in terms of mAP. The heuristics performed better than the GNNs on larger datasets, while the GNNs performed better on the smaller datasets. Among the GNNs, the discrete DGNNs performed consistently better, with continuous DGNNs performing poorly, except for on the Wikipedia and Reddit datasets where the continuous DGNNs performed relatively well. The MRR scores are quite similar to the mAP scores, however, the AUC scores rank the models completely differently. The GNNs consistently have higher AUCs than heuristics and the continuous DGNNs have the highest AUC scores for most datasets. In other words, individually, the mAP scores and AUC scores tell two almost contradicting stories.

An explanation for this disparity lies in the extreme class imbalance inherent to link prediction (Yang et al., 2015). The AUC score relies on the false positive rate. Due to the large number of non-existing links, the false positive rate may stay relatively low despite the precision is also being low. In fact, the mAP and AUC are shown to be approximately the same, except for the precision of the highest ranked links (Su et al., 2015). Our results thus indicate that the link prediction heuristics are better at ranking links initially, but later links are ranked better by the GNNs.

For applications focused on the highest ranked links, which is common in information retrieval and recommender systems, our results indicate that link prediction heuristics will typically outperform

GNNs in the absence of informative node or edge features. For a comprehensive prediction of the network, e.g. predicting future snapshots, our results indicate that the heuristics and GNNs might be complementary. Incorporating heuristics into GNNs may yield improved performance. One example of a combined prediction is done by Sankar et al. (2020).

How do different GNNs compare? Among the GNN models, DGNNs performed better than static GNNs. In terms of mAP, the discrete models performed well, and in terms of AUC, the continuous models did well. With the exception of Enron, a DGNN always outperformed the static GNNs across all reported metrics. Particularly GC-LSTM performs comparatively well. Static GNNs are on some datasets, e.g. Wikipedia, closer to the performance of random predictions.

Even in terms of mAP, the continuous models do well on Wikipedia and Reddit where they have informative edge features. However, their performance is lacking on other datasets. This implies that they may rely on edge features for good performance; we explore this further in Section 4.4. Their performance in terms of AUC is generally good. This performance gap, and as explained earlier, difficulty with ranking initial links, may be caused by the choice of negative sampling when training the encoders. During training only one randomly chosen negative sample is used per link. Increasing the quality or quantity of negative samples will possibly improve their performance.

Why did GC-LSTM outperform the other DGNNs? GC-LSTM is a fairly simple DGNN architecture where each snapshot in the snapshot window is first encoded by a GNN and the resulting node embeddings are then encoded by an LSTM. The EGCN models focus on evolving the GNN weights rather than the node embeddings. The EGCN-O model does not take node embeddings as an input, while EGCN-H only takes the top-k node embeddings as input. We hypothesize that the performance difference is caused by the EGCN models deemphasizing node embeddings.

Another difference in our comparison of GC-LSTM and the EGCN models is the GNNs and their implementation. GC-LSTM uses a GNN from PyTorch geometric (Fey & Lenssen, 2019), while the EGCN models use the original author's GCN (Pareja et al., 2020) implementation[3].

Why are the mAP scores so different between datasets? The scores vary notably between datasets. Datasets with low snapshot density have lower mAP scores than those with high snapshot density. This reflects the increased difficulty of the classification problem which comes with increased class imbalance (Yang et al., 2015).

The Autonomous dataset does however not appear to follow this pattern. The methods can perform well despite the network being rather sparse. We speculate that this might be due to Autonomous being an evolving network rather than an interaction network (Section 2.1). In slowly evolving networks there is less change between snapshots, and methods only need to predict gradual changes to the network as fewer links disappear. It is shown that network characteristics, such as the clustering coefficient and average shortest path, influence heuristic performance (Gao et al., 2015). It is plausible that link duration influences performance as well, however, since this study includes only one evolving dataset, further work is needed to confirm this.

4.2 Snapshot training window analysis

Figure 1 shows the mAP scores found during the grid search on the static and discrete models. One datapoint is one parameter setting in the search. A sliding time-window of size 5 or 10 consistently produces the best results, particularly for the discrete models. This indicates that it is beneficial to use a sliding window when training DGNNs. Most models are spread across a large spectrum of scores, implying that optimizing the hyperparameters is essential for obtaining a representative score for both GNNs and DGNNs.

4.3 Using a parameter budget

The grid search explored different layer sizes. However, this did not take into account the total number of learnable parameters. The DGNNs, therefore, ended up with much more learnable parameters than the static GNNs. To explore whether the discrete models can to fit the data better simply due to

[3]We attempted to use the PyTorch Geometric Temporal implementation of the EGCN models (Rozemberczki et al., 2021), but the original author's code outperformed the PyTorch Geometric Temporal implementation. We believe this was due to a bug in PyTorch Geometric Temporal that has since been fixed.

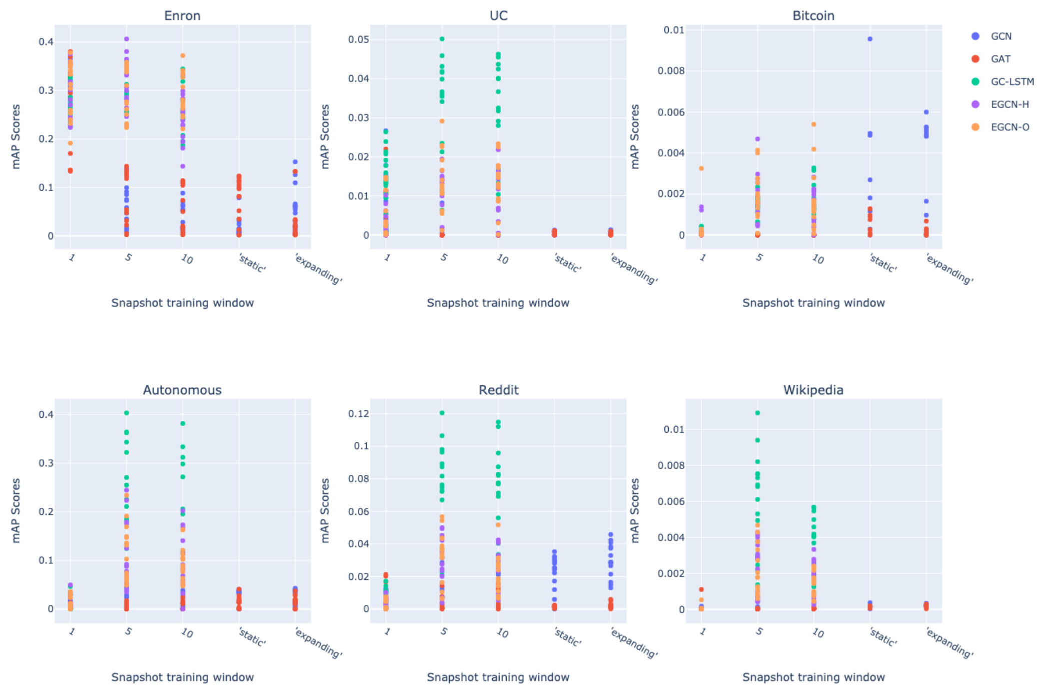

Figure 1: Grid search mAP scores against the size of the snapshot training window. The snapshot training window is explained in Section A.4.1. Each color represents the five GNNs. All figures show different y-axes for the separate datasets due to large differences in scores between datasets.

Table 3: mAP scores of GCN and GC-LSTM compared on a parameter budget. GCN+P is the GCN model with the same number of learnable parameters as the GC-LSTM.

Models	Enron	UC	Bitcoin	Autonomous	Wikipedia	Reddit
GCN	$\mathbf{0.322(0.022)}$	$0.021(0.004)$	$0.002(0.002)$	$0.032(0.006)$	$3 \cdot 10^{-5}(0.000)$	$0.038(0.007)$
GCN+P	$0.302(0.037)$	$0.010(0.010)$	$\mathbf{0.005(0.000)}$	$0.090(0.087)$	$2 \cdot 10^{-4}(0.000)$	$0.040(0.002)$
GC-LSTM	$0.270(0.023)$	$\mathbf{0.047(0.003)}$	$0.002(0.001)$	$\mathbf{0.406(0.024)}$	$\mathbf{0.007(0.001)}$	$\mathbf{0.097(0.007)}$

having more parameters, we run GCN with the same number of parameters as the best performing GC-LSTM setting. For the exact change in layer size and the resulting total number of parameters, see Table 10. We perform a full grid search to locate good values for learning rate and snapshot training window given these new layer sizes.

The mAP scores for the GCN with more parameters are compared to the original scores of GCN and GC-LSTM in Table 3. In general, the additional parameters enabled the GCN to achieve a marginally higher score on the larger datasets, but not enough to outperform GC-LSTM, except for on the bitcoin dataset.

4.4 Exploring the performance of continuous DGNNs

In Section 4.1 we hypothesized that the continuous DGNNs relied on informative edge features to achieve good results on the Wikipedia and Reddit datasets. To check this hypothesis, we run TGAT and TGN on these datasets with randomized edge features. The results, shown in Table 4, show a substantial decrease in performance when edge features are randomized, particularly on the Wikipedia dataset. The performance on Reddit has decreased, but not as drastically. This shows that the edge features are an important, but not always critical, asset to aid encoding.

Overall, the continuous DGNNs performed relatively poorly compared to discrete DGNNs in terms of mAP. We suspect this is caused by: (i) Reliance on edge features (ii) transferring from a contin-

Table 4: Continuous DGNNs with randomized edge features. '-RE' indicate random edge features.

Models	Wikipedia	Wikipedia-RE	% diff	Reddit	Reddit-RE	% diff
TGAT	$0.006(0.000)$	$0.002(0.002)$	-61%	$0.108(0.007)$	$0.088(0.005)$	-18%
TGN	$0.004(0.000)$	$0.001(0.000)$	-83%	$0.042(0.009)$	$0.023(0.008)$	-29%

uous setting to a discrete setting takes the embeddings to some extent out of their element, and (iii) the hyperparameters of the continuous DGNNs were optimized by the original authors on Wikipedia and Reddit. The performance of the continuous DGNNs can probably be significantly improved by exploring other ways to apply continuous DGNNs to the discrete representation and by further optimizing the hyperparameters.

5 Conclusions and future work

We introduce a framework that enables the comparison of discrete and continuous DGNNs. We use this framework to perform a comprehensive comparison of link prediction heuristics and three types of GNNs on the dynamic link prediction task.

Link prediction heuristics performed better in terms of mAP and DGNNs performed better in terms of AUC. We believe the mAP metric is more indicative of link prediction performance (Yang et al., 2015). For applications concerned with not only the highest ranked link but all links, heuristics and GNNs appear to be complementary.

Despite heuristics being simple, they prove to be strong baselines. Future work on GNNs and DGNNs which are applied to link prediction should be compared to at least one link prediction heuristic. However, while heuristics currently outperform GNNs in terms of mAP, the heuristics cannot leverage informative node or edge features, nor do they leverage temporal patterns. Therefore, GNNs and DGNNs have a lot of potential to improve their performance beyond the heuristics.

We find that the snapshot training window greatly affects performance. Future work should explore multiple snapshot training window sizes. Despite searching multiple different static network representations, the discrete DGNNs consistently outperformed static GNNs. Our results also indicate that network characteristics, such as the link duration (temporal vs evolving networks) influence prediction performance. Continuous DGNNs performed well when they had informative edge features, but not without them.

Exciting directions for future work include: (i) Incorporating link prediction heuristics and recent advances in GNNs into DGNNs, (ii) exploring effective ways of training continuous DGNNs on discrete networks, (iii) exploring the influence of dynamic network characteristics on link prediction performance, and (iv) expanding this benchmark to include additional methods and datasets. Interesting additional models to compare include other discrete DGNNs (Sankar et al., 2020), approaches using auto-encoder loss functions (Chen et al., 2019), generative adversarial network based approaches (Lei et al., 2019; Xiong et al., 2019) and temporal point process based continuous DGNNs (Trivedi et al., 2019; Knyazev et al., 2021).

Improving the performance of both discrete and continuous DGNNs remains an exciting research avenue. We consider this work a first step towards improving dynamic link predictions and an important foundation for future work.

References

Lada A. Adamic and Eytan Adar. Friends and Neighbors on the Web. *Social Networks*, 25(3): 211–230, 2003. doi: 10.1016/s0378-8733(03)00009-1. URL `https://doi.org/10.1016/s0378-8733(03)00009-1`.

Iftikhar Ahmad, Muhammad Usman Akhtar, Salma Noor, and Ambreen Shahnaz. Missing link prediction using common neighbor and centrality based parameterized algorithm. *Scientific reports*, 10(1):1–9, 2020.

Inci M. Baytas, Cao Xiao, Xi Zhang, Fei Wang, Anil K. Jain, and Jiayu Zhou. Patient Subtyping via Time-Aware LSTM Networks. In *Proceedings of the 23rd ACM SIGKDD International*

Conference on Knowledge Discovery and Data Mining - KDD '17, pp. 65–74, Halifax, NS, Canada, 2017. ACM Press. ISBN 978-1-4503-4887-4. doi: 10.1145/3097983.3097997. URL `http://dl.acm.org/citation.cfm?doid=3097983.3097997`.

Jinyin Chen, Jian Zhang, Xuanheng Xu, Chengbo Fu, Dan Zhang, Qingpeng Zhang, and Qi Xuan. E-LSTM-D: A Deep Learning Framework for Dynamic Network Link Prediction. *arXiv preprint arXiv:1902.08329*, 2019.

Jinyin Chen, Xueke Wang, and Xuanheng Xu. GC-LSTM: graph convolution embedded LSTM for dynamic network link prediction. *Applied Intelligence*, pp. 1–16, 2021. Publisher: Springer.

Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems 29*, pp. 3844–3852. Curran Associates, Inc., 2016.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Federico Errica, Marco Podda, Davide Bacciu, and Alessio Micheli. A fair comparison of graph neural networks for graph classification. In *International conference on learning representations*, 2020. URL `https://openreview.net/forum?id=HygDF6NFPB`.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR workshop on representation learning on graphs and manifolds*, 2019.

Fei Gao, Katarzyna Musial, Colin Cooper, and Sophia Tsoka. Link prediction methods and their accuracy for different social networks and network metrics. *Scientific programming*, 2015, 2015. Publisher: Hindawi.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT Press, 2016. http://www.deeplearningbook.org.

Petter Holme. Modern temporal network theory: a colloquium. *The European Physical Journal B*, 88(9):1–30, 2015. Publisher: Springer.

Petter Holme and Jari Saramäki. Temporal networks. *Physics reports*, 519(3):97–125, 2012.

Weihua Hu, Matthias Fey, Marinka Zitnik, Yuxiao Dong, Hongyu Ren, Bowen Liu, Michele Catasta, and Jure Leskovec. Open graph benchmark: Datasets for machine learning on graphs. *arXiv preprint arXiv:2005.00687*, 2020.

Seyed Mehran Kazemi, Rishab Goel, Sepehr Eghbali, Janahan Ramanan, Jaspreet Sahota, Sanjay Thakur, Stella Wu, Cathal Smyth, Pascal Poupart, and Marcus Brubaker. Time2vec: Learning a vector representation of time. *arXiv preprint arXiv:1907.05321*, 2019.

Seyed Mehran Kazemi, Rishab Goel, Kshitij Jain, Ivan Kobyzev, Akshay Sethi, Peter Forsyth, and Pascal Poupart. Representation learning for dynamic graphs: A survey. *Journal of Machine Learning Research*, 21(70):1–73, 2020.

Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *5th international conference on learning representations, ICLR 2017, toulon, france, april 24-26, 2017, conference track proceedings*. OpenReview.net, 2017. URL `https://openreview.net/forum?id=SJU4ayYgl`.

Boris Knyazev, Carolyn Augusta, and Graham W Taylor. Learning temporal attention in dynamic graphs with bilinear interactions. *Plos one*, 16(3):e0247936, 2021. publisher: Public Library of Science San Francisco, CA USA.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1269–1278, 2019.

Kai Lei, Meng Qin, Bo Bai, Gong Zhang, and Min Yang. GCN-GAN: A non-linear temporal link prediction model for weighted dynamic networks. In *IEEE INFOCOM 2019-IEEE conference on computer communications*, pp. 388–396, 2019.

David Liben-Nowell and Jon Kleinberg. The Link-Prediction Problem for Social Networks. *Journal of the American Society for Information Science and Technology*, 58(7):1019–1031, 2007. doi: 10.1002/asi.20591. URL `https://doi.org/10.1002/asi.20591`.

Zachary C. Lipton and Jacob Steinhardt. Troubling trends in machine learning scholarship: Some ML papers suffer from flaws that could mislead the public and stymie future research. *Queue*, 17(1):45–77, February 2019. ISSN 1542-7730. doi: 10.1145/3317287.3328534. URL `https://doi.org/10.1145/3317287.3328534`. publisher: Association for Computing Machinery.

Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. Streaming graph neural networks. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*, SIGIR '20, pp. 719–728, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 978-1-4503-8016-4. doi: 10.1145/3397271.3401092. URL `https://doi-org.ezproxy.lib.uts.edu.au/10.1145/3397271.3401092`. numPages: 10 place: Virtual Event, China.

Víctor Martínez, Fernando Berzal, and Juan-Carlos Cubero. A Survey of Link Prediction in Complex Networks. *ACM Comput. Surv.*, 49(4):69:1–69:33, December 2016. ISSN 0360-0300. doi: 10.1145/3012704. URL `http://doi.acm.org/10.1145/3012704`.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5363–5370, 2020.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Benedek Rozemberczki, Paul Scherer, Yixuan He, George Panagopoulos, Maria Astefanoaei, Oliver Kiss, Ferenc Beres, Nicolas Collignon, and Rik Sarkar. PyTorch geometric temporal: Spatiotemporal signal processing with neural machine learning models. *arXiv preprint arXiv:2104.07788*, 2021.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. DySAT: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th international conference on web search and data mining*, WSDM '20, pp. 519–527, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 978-1-4503-6822-3. doi: 10.1145/3336191.3371845. URL `https://doi.org/10.1145/3336191.3371845`. numPages: 9 place: Houston, TX, USA.

National Academies of Sciences, Engineering, Medicine, and others. *Reproducibility and replicability in science*. National Academies Press, 2019.

Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured Sequence Modeling with Graph Convolutional Recurrent Networks. In Long Cheng, Andrew Chi Sing Leung, and Seiichi Ozawa (eds.), *Neural Information Processing*, Lecture Notes in Computer Science, pp. 362–373. Springer International Publishing, 2018. ISBN 978-3-030-04167-0.

Joakim Skarding, Bogdan Gabrys, and Katarzyna Musial. Foundations and modelling of dynamic networks using dynamic graph neural networks: A survey. *IEEE Access*, 2021. Publisher: IEEE.

Wanhua Su, Yan Yuan, and Mu Zhu. A relationship between the average precision and the area under the ROC curve. In *Proceedings of the 2015 international conference on the theory of information retrieval*, pp. 349–352, 2015.

Rakshit Trivedi, Hanjun Dai, Yichen Wang, and Le Song. Know-Evolve: Deep Temporal Reasoning for Dynamic Knowledge Graphs. *arXiv:1705.05742 [cs]*, June 2017. URL `http://arxiv.org/abs/1705.05742`. arXiv: 1705.05742.

Rakshit Trivedi, Mehrdad Farajtabar, Prasenjeet Biswal, and Hongyuan Zha. DyRep: learning representations over dynamic graphs. In *International conference on learning representations*, 2019. URL `https://openreview.net/forum?id=HyePrhR5KX`.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. In *International conference on learning representations*, 2018. URL `https://openreview.net/forum?id=rJXMpikCZ`.

Akanda Wahid-Ul-Ashraf, Marcin Budka, and Katarzyna Musial-Gabrys. Newton's gravitational law for link prediction in social networks. In *International conference on complex networks and their applications*, pp. 93–104, 2017. tex.organization: Springer.

Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. A comprehensive survey on graph neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, pp. 1–21, 2020. tex.ids: wuComprehensiveSurveyGraph2019.

Yun Xiong, Yao Zhang, Hanjie Fu, Wei Wang, Yangyong Zhu, and S Yu Philip. DynGraphGAN: Dynamic graph embedding via generative adversarial networks. In *International conference on database systems for advanced applications*, pp. 536–552, 2019. tex.organization: Springer.

da Xu, chuanwei ruan, evren korpeoglu, sushant kumar, and kannan achan. Inductive representation learning on temporal graphs. In *International conference on learning representations*, 2020. URL `https://openreview.net/forum?id=rJeW1yHYwH`.

Yang Yang, Ryan N Lichtenwalter, and Nitesh V Chawla. Evaluating link prediction methods. *Knowledge and Information Systems*, 45(3):751–782, 2015. Publisher: Springer.

Jian Zhang. A survey on streaming algorithms for massive graphs. In *Managing and Mining Graph Data*, pp. 393–420. Springer, 2010.

Muhan Zhang and Yixin Chen. Link prediction based on graph neural networks. In *Advances in neural information processing systems*, pp. 5165–5175, 2018.

Muhan Zhang, Pan Li, Yinglong Xia, Kai Wang, and Long Jin. Revisiting graph neural networks for link prediction. *arXiv preprint arXiv:2010.16103*, 2020.

Jie Zhou, Ganqu Cui, Shengding Hu, Zhengyan Zhang, Cheng Yang, Zhiyuan Liu, Lifeng Wang, Changcheng Li, and Maosong Sun. Graph neural networks: A review of methods and applications. *AI Open*, 1:57–81, 2020. Publisher: Elsevier.

A Appendix

A.1 Dataset details

Enron [4], is an email communication network, where a link is an email sent between two people. Enron is a tiny network spatially, but medium-sized temporally with a reasonable number of continuous links and covering a time span of over 3 years. Due to the small number of nodes and a comparatively large number of edges, it is much denser than the other networks.

UC Irvine messages[5], shortened to UC, is an online forum network from the University of California, Irvine. Two students are connected if they interact on the same forum post. Thus this was originally a bipartite network but it has been projected to have nodes of only one type. The odd choice of snapshot size is adapted from EvolveGCN (Pareja et al., 2020) which observes that a smaller snapshot size yields some snapshots without any edges.

Bitcoin-OTC[6], shortened to Bitcoin, is a who-trust-whom network of people trading on the Bitcoin OTC platform. A link is an evaluation by one user of another. The bitcoin network is medium-sized in terms of nodes, however, most of its edges are unique edges which indicate that very few edges

[4]`http://networkrepository.com/ia-enron-employees.php`

[5]`http://konect.cc/networks/opsahl-ucforum/`

[6]`https://snap.stanford.edu/data/soc-sign-bitcoin-otc.html`

Table 5: Dynamic link prediction AUC scores. Formatted identically to Table 2

Models	Enron	UC	Bitcoin	Autonomous	Wikipedia	Reddit
Random	0.521(0.048)	0.497(0.013)	0.487(0.008)	0.500(0.000)	0.501(0.001)	0.499(0.001)
R-embed	0.443(0.084)	0.514(0.023)	0.497(0.007)	0.516(0.009)	0.505(0.003)	0.504(0.008)
CN	0.713	0.734	0.638	0.996	0.749	0.844
AA	0.714	0.734	0.638	0.996	0.749	0.844
Jaccard	0.712	0.734	0.523	**0.999**	0.749	0.906
Newton	0.727	0.810	0.526	0.998	0.829	0.967
CCPA	0.728	0.810	0.783	0.998	0.826	0.944
GCN	$\underline{\mathbf{0.911}}$(0.011)	0.646(0.006)	0.777(0.242)	0.454(0.127)	0.412(0.019)	0.974(0.001)
GAT	0.903(0.037)	0.618(0.053)	0.861(0.019)	0.562(0.030)	0.275(0.009)	0.646(0.076)
EGCN-H	0.872(0.071)	0.662(0.048)	0.765(0.018)	0.892(0.050)	0.705(0.003)	0.951(0.009)
EGCN-O	0.888(0.042)	0.585(0.014)	0.712(0.069)	0.904(0.036)	0.708(0.006)	0.951(0.011)
GC-LSTM	0.904(0.037)	0.555(0.004)	0.741(0.031)	$\underline{0.944}$(0.007)	0.706(0.006)	0.968(0.001)
TGAT	0.483(0.010)	0.700(0.283)	$\underline{\mathbf{0.920}}$(0.002)	†	0.948(0.001)	$\underline{\mathbf{0.985}}$(0.000)
TGN	0.865(0.007)	$\underline{\mathbf{0.947}}$(0.001)	0.916(0.001)	†	$\underline{\mathbf{0.965}}$(0.002)	0.969(0.001)

Table 6: Dynamic link prediction MRR scores. Formatted identically to Table 2

Models	Enron	UC	Bitcoin	Autonomous	Wikipedia	Reddit
Random	0.038(0.010)	0.004(0.001)	0.001(0.000)	0.001(0.000)	0.001(0.000)	0.001(0.000)
R-embed	0.030(0.014)	0.004(0.001)	0.002(0.001)	0.031(0.017)	0.001(0.000)	0.216(0.017)
CN	0.337	0.261	0.046	**0.625**	0.303	**0.444**
AA	0.338	0.262	0.046	**0.625**	0.303	0.442
Jaccard	0.327	0.250	0.029	0.411	0.257	0.229
Newton	0.291	**0.267**	0.029	0.615	**0.308**	0.432
CCPA	0.337	**0.267**	0.047	0.623	0.303	0.441
GCN	$\underline{\mathbf{0.404}}$(0.027)	0.141(0.021)	0.071(0.012)	0.0184(0.003)	0.002(0.002)	0.216(0.017)
GAT	0.369(0.093)	0.097(0.025)	0.013(0.002)	0.265(0.027)	0.013(0.018)	0.079(0.026)
EGCN-H	0.339(0.052)	0.110(0.110)	0.062(0.015)	0.340(0.095)	0.053(0.004)	0.204(0.015)
EGCN-O	0.318(0.029)	0.140(0.006)	0.075(0.016)	0.383(0.050)	0.007(0.010)	0.194(0.007)
GC-LSTM	0.332(0.069)	$\underline{0.162}$(0.004)	$\underline{\mathbf{0.079}}$(0.007)	$\underline{0.587}$(0.018)	0.098(0.005)	$\underline{0.344}$(0.007)
TGAT	0.117(0.023)	0.041(0.018)	0.059(0.001)	†	$\underline{0.127}$(0.004)	0.327(0.002)
TGN	0.187(0.010)	0.100(0.021)	0.033(0.005)	†	0.095(0.004)	0.223(0.012)

are reoccurring. Lack of reoccurring edges causes each snapshot to be much sparser than most of the other datasets.

Autonomous-systems[7], shortened to Autonomous, is an internet router communication network. A link is a router exchanging traffic flow with a peer. This network is already aggregated as a discrete network. We follow Pareja et al. (2020) in selecting the first 99 days and using that as our dataset. This is by far the dataset with the most edges.

Wikipedia[8], a bipartite Wikipedia page editing network. Nodes are either a Wikipedia user or a Wikipedia page. A link is a user editing a Wikipedia page. Wikipedia also has few reoccurring edges and similarly to Bitcoin, has then comparably sparse snapshots.

Reddit[9], a bipartite Reddit posting network. Nodes are either a Reddit user or a subreddit. A link is a user posting on a subreddit. Reddit is the largest network spatially as it has the largest number of nodes and unique edges.

We use a slightly larger train split on Enron as it is a tiny dataset and we had difficulties with the models learning anything if the training set was too small. The UC, Bitcoin, and Autonomous data-splits follow Pareja et al. (2020), and the Wikipedia and Reddit splits follow Xu et al. (2020). We use the same splits so we can compare our results to the previous works.

A.2 AUC AND MRR SCORES

Table 5 and Table 6 present the AUC and the MRR scores respectively. These results are discussed in Section 4.1. The scores are from the same experiments as the mAP scores in Table 2.

[7] `http://snap.stanford.edu/data/as-733.html`
[8] `http://snap.stanford.edu/jodie/wikipedia.csv`
[9] `http://snap.stanford.edu/jodie/reddit.csv`

Table 7: Overview of compared methods.

Models	Temporal granularity	Method/architecture type
CN (Liben-Nowell & Kleinberg, 2007)	Static	Heuristic
AA (Adamic & Adar, 2003)	Static	Heuristic
Jaccard (Liben-Nowell & Kleinberg, 2007)	Static	Heuristic
Newton (Wahid-Ul-Ashraf et al., 2017)	Static	Heuristic
CCPA (Ahmad et al., 2020)	Static	Heuristic
GCN (Kipf & Welling, 2017)	Static	Spectral GNN (Wu et al., 2020)
GAT (Veličković et al., 2018)	Static	Spatial GNN (Wu et al., 2020)
EGCN-H (Pareja et al., 2020)	Discrete	Integrated DGNN (Skarding et al., 2021)
EGCN-O (Pareja et al., 2020)	Discrete	Integrated DGNN (Skarding et al., 2021)
GC-LSTM (Chen et al., 2021)	Discrete	Integrated DGNN (Skarding et al., 2021)
TGAT (Xu et al., 2020)	Continuous	Time embedding based (Skarding et al., 2021)
TGN (Rossi et al., 2020)	Continuous	Time embedding based (Skarding et al., 2021)

A.3 Implementation details of models

Table 7 is an overview of the methods, their temporal granularity, and method type (architecture type in the case of GNNs). The architecture types are categories of GNNs identified by surveys (Wu et al., 2020; Skarding et al., 2021).

We use either standardized implementations or the original authors' code. For GCN and GAT, we use the PyTorch Geometric implementation (Fey & Lenssen, 2019)[10], for GC-LSTM we use the PyTorch Geometric Temporal implementation (Rozemberczki et al., 2021)[11] and we use the original authors' code for the EGCN models[12], TGAT[13] and TGN[14].

Unless otherwise stated, we do not modify any of the tested models. Some minor modifications were made to GC-LSTM, TGAT, and TGN to enable the comparison. The PyTorch Geometric Temporal implementation of GC-LSTM (Rozemberczki et al., 2021) was modified to enable sliding windows in the same manner as originally used by EvolveGCN (Pareja et al., 2020). For TGAT and TGN we leave the training unchanged, but add the functionality to extract node embeddings to enable comparison to the discrete models.

We attempted to add the static GNN, SEAL (Zhang & Chen, 2018) to our benchmark as it is a promising GNN (Zhang et al., 2020) specifically targeted at link prediction. However, the model requires preprocessing 2-hop subgraphs for every node pair. We found this to not scale well. This is particularly due to us not using negative sampling (sampling of non-links) during validation and testing. Simply storing these preprocessed subgraphs for one of our larger datasets would require several TB of disk space.

A.4 Training pipelines

While the evaluation is identical for all methods, the training procedure is different between the different kinds of methods. This is a necessity due to the different network representations that the methods operate on.

A.4.1 Static

Static GNNs encode a graph. Since our training set includes multiple snapshots, we convert these snapshots into one graph to enable the training of the GNN. We can aggregate an arbitrary number of snapshots into one snapshot by including a link in the output snapshot if it occurs in any of the input snapshots, thus turning a discrete network into a static one. We do this in three different ways and consider these approaches a hyperparameter.

The most straightforward way to train a GNN on a dynamic network is to combine all the snapshots in the training set into one big snapshot. We call this approach 'static'. This is the approach taken by

[10]`https://github.com/rusty1s/pytorch_geometric`

[11]`https://github.com/benedekrozemberczki/pytorch_geometric_temporal`

[12]`https://github.com/IBM/EvolveGCN`

[13]`https://github.com/StatsDLMathsRecomSys/Inductive-representation-learning-on-temporal-grap`

[14]`https://github.com/twitter-research/tgn`

traditional link prediction and continuous DGNN works (Xu et al., 2020; Rossi et al., 2020). This is the only approach not to train in a ”roll forward” manner.

It is presumably beneficial to exploit the temporal information in the training set and roll forward during training. One way to do this is to only encode the previous snapshot when attempting to predict the next snapshot. This does not require any snapshot aggregation. This can be seen as a sliding window of size 1 and is the approach used by Pareja et al. (2020) for static GNNs.

Complex networks tend to be rather sparse. It might therefore be beneficial to use a sliding window. We explore sliding windows of size 5 and 10. Size 10 is the default for EGCN, we chose to additionally use size 5 to investigate whether the size of the sliding window is influential. For the static models, these ”sliding snapshot windows” are aggregated into one snapshot. For even sparser networks it may be beneficial to represent the dynamic network as an evolving network. For this, we use an expanding window. We refer to this option as ’expanding’.

A.4.2 DISCRETE

Most discrete models inherently support multiple snapshots, but the number of snapshots cannot vary. It is therefore necessary to use a sliding window that feeds a consistent number of snapshots to the model. We use sliding window sizes of 1, 5, and 10. While this is comparable to the sliding window of the static models, it is also different since the snapshots in these sliding windows are not aggregated together.

A.4.3 CONTINUOUS

Continuous models have no notion of snapshots, and we are unaware of anyone training continuous models on discrete networks. As this is a comparative study we aim to train the models the same way they were originally trained, yet also in a way that allows us to compare the results fairly. Our solution is to train in two steps. Firstly we train the encoder, secondly the decoder.

Continuous models are trained edge-by-edge. Like other time-series models the edges are batched. This hinders us from training the continuous models end-to-end with our decoder (recall that we want to use the same decoder for all models) since the decoder backpropagates on each snapshot.

It is theoretically possible to train the continuous models end-to-end with our decoder by changing the edge batch size from snapshot to snapshot. This will however lead the number of edges in the batches to change from batch to batch. Whether the radical change in batch size throughout training is a viable way to train is unknown. However, we deem it as too different from the original way these networks were trained to include this approach in our study. We encourage future work to explore end-to-end training on snapshots for continuous models.

We opt to train the continuous models in two steps. First, we train identically to how the models were originally trained, with a constant batch size (essentially pretending snapshots do not exist), using contrastive learning, and a negative sampling rate of 1 to 1. We then extract the node embeddings and train our decoder separately with the encoder (the continuous DGNN) frozen, this time with a batch size matching the number of edges in the snapshot. Training with the same decoder allows us to use the same class weights and negative sampling rate (1 to 100) as the static and discrete models.

To speed up decoder training, we cache the node embeddings produced by the frozen encoder in the first epoch of the decoder training. This speeds up training by a factor of at least $10x$.

Despite the effort to optimize training speed, we opted to not test the continuous models on the Autonomous dataset. A single (encoder) epoch took on average 16 hours to run (wall-time) on our hardware (see Section A.6 for the hardware). The discrete evolving network could be preprocessed into a more suitable format for continuous models, but doing so is non-trivial and we consider that outside the scope of this work.

A.5 HYPERPARAMETERS

For the static GNNs we search the parameters; learning rate, snapshot training window, and hidden layer size. The search for parameters on discrete DGNNs is identical to the static except for the snapshot training window which only searches sliding windows. The grid search is slightly modified

Table 8: Hyperparameters searched by the grid search.

Hyperparameter	Methods	Values
	Heuristic	1, 5, 10, static
Snapshot training window	Static	1, 5, 10, expanding, static
	Discrete	1, 5, 10
Existing edge treatment	Heuristic	Default, score, adaptive
Learning rate	Static, discrete, continuous	0.0001, 0.005, 0.001, 0.05, 0.01
Hidden layer size	Static, discrete	50, 100, 200 (10, 20, 30 on Enron)
Decoder learning rate	Continuous	0.0001, 0.005, 0.001, 0.05, 0.01
Decoder weight decay	Continuous	0, 0.0001, 0.01

Table 9: Hyperparameters selected by the grid search

Models	Hyperparameter	Enron	UC	Bitcoin	Autonomous	Wikipedia	Reddit
CN	Snapshot training window	1	10	10	1	5	5
	Existing edge treatment	default	default	adaptive	default	default	default
AA	Snapshot training window	1	10	10	1	5	5
	Existing edge treatment	default	default	adaptive	default	default	default
Jaccard	Snapshot training window	1	10	1	expanding	5	expanding
	Existing edge treatment	default	default	adaptive	default	default	default
Newton	Snapshot training window	1	10	1	1	5	5
	Existing edge treatment	default	default	adaptive	default	default	default
CCPA	Snapshot training window	1	10	10	1	5	5
	Existing edge treatment	default	default	adaptive	default	default	default
GCN	Snapshot training window	1	1	static	5	5	expanding
	Learning rate	0.005	0.005	0.05	0.0001	0.05	0.01
	Hidden layer size	10	50	200	50	200	50
GAT	Snapshot training window	1	1	static	10	1	1
	Learning rate	0.001	0.005	0.001	0.005	0.005	0.01
	Hidden layer size	30	50	50	50	50	100
EGCN-H	Snapshot training window	5	10	5	5	5	5
	Learning rate	0.01	0.01	0.01	0.01	0.01	0.01
	Hidden layer size	10	50	100	200	100	50
EGCN-O	Snapshot training window	1	5	10	5	5	5
	Learning rate	0.01	0.001	0.01	0.005	0.005	0.01
	Hidden layer size	10	200	200	200	200	100
GC-LSTM	Snapshot training window	5	5	10	10	5	5
	Learning rate	0.005	0.001	0.005	0.01	0.001	0.01
	Hidden layer size	10	50	50	200	50	200
TGAT	Learning rate	0.001	0.0001	0.001	†	0.0001	0.0001
	Decoder learning rate	0.05	0.001	0.0001	†	0.001	0.005
	Decoder weight decay	0.01	0.0001	0	†	0.0001	0
TGN	Learning rate	0.0001	0.0001	0.001	†	0.0001	0.0001
	Decoder learning rate	0.0001	0.001	0.005	†	0.001	0.001
	Decoder weight decay	0.01	0.0001	0.0001	†	0.0001	0

on the Enron dataset where we search smaller layer sizes due to the dataset being so small. The parameters searched in the grid search are shown in Table 8, the selected hyperparameters are shown in Table 9.

Continuous models are trained with the original hyperparameters and we perform a grid search on the second training stage where we train the decoder. The parameters optimized are; learning rate, decoder learning rate, and decoder weight decay.

By default, link prediction heuristics don't predict scores for already existing edges. We choose to explore three options. (i) Calculating a score as if the edge didn't exist, (ii) Giving the existing links the same score based on the historical probability of a link persisting between snapshots, and (iii) the default option, simply assuming existing links will persist. We also search the snapshot training window as shown in Table 8.

Table 10: Layer size and total number of learnable parameters for the equal parameter budget runs in Section 4.3. The number of learnable parameters of the encoder is in parenthesis.

Models	Enron		UC		Bitcoin		Autonomous		Wikipedia		Reddit	
GCN	10	(880)	50	(15400)	200	(199600)	50	(39850)	200	(131400)	50	(130400)
GCN+P	70	(10360)	320	(184960)	420	(511560)	1200	(2336400)	360	(294120)	1600	(6652800)
GC-LSTM	10	(10470)	50	(184350)	50	(508350)	200	(227100)	50	(303750)	200	(6617400)

Some hyperparameters are common for all GNNs. The maximum number of epochs is 500. We use early stopping with an early stop patience of 100. The TGAT and TGN encoders are trained as in their original works, with a maximum of 50 epochs (epochs on continuous models are significantly longer than on static or discrete models. Training using a high number of epochs is therefore impractical). We evaluate the models on the validation set every 5 epochs.

Other hyperparameters are specific for models; we use the original authors' parameters and we keep them the same across all datasets. For GAT we use 8 attention heads and a dropout value of 0.5. GC-LSTM uses a spectral GCN (Defferrard et al., 2016) which approximates the graph convolution and takes a hyperparameter K. It indicates the number of hops included in the neighborhood convolution. We use $K = 3$. For continuous models we use a batch size of 200, 2 attention heads, and a dropout of 0.1. For TGN we activate memory as that is its major feature distinguishing it from TGAT.

A.6 Runtime & hardware

The grid search consisted of searching $3,690$ different parameter settings, each setting took on average roughly 4 hours to complete. The grid search took roughly $13,900$ hours. Running the best found hyperparameters with four different seeds is 140 different runs. These runs took on average roughly 7 hours to complete, those runs took roughly $1,500$ hours. In total that is $15,400$ hours, or 642 days.

The runs were computed in parallel on HPC clusters. The clusters varied slightly in their architecture, but a typical node had a processor equivalent to an Intel Xeon Gold 6126 and a GPU equivalent to an NVIDIA Quadro P6000. A singularity container was used to ensure a consistent runtime environment.