Efficient and Effective Implicit Dynamic Graph Neural Network

Yongjian Zhong
Department of Computer Science
University of Iowa
Iowa City, IA, USA
yongjian-zhong@uiowa.edu

Hieu Vu
Department of Computer Science
University of Iowa
Iowa City, IA, USA
hieu-vu@uiowa.edu

Tianbao Yang
Department of Computer Science and Engineering
Texas A&M University
College Station, TX, USA
tianbao-yang@tamu.edu

Bijaya Adhikari
Department of Computer Science
University of Iowa
Iowa City, IA, USA
bijaya-adhikari@uiowa.edu

ABSTRACT

Implicit graph neural networks have gained popularity in recent years as they capture long-range dependencies while improving predictive performance in static graphs. Despite the tussle between performance degradation due to the oversmoothing of learned embeddings and long-range dependency being more pronounced in dynamic graphs, as features are aggregated both across neighborhood and time, no prior work has proposed an implicit graph neural model in a dynamic setting.

In this paper, we present Implicit Dynamic Graph Neural Network (IDGNN) a novel implicit neural network for dynamic graphs which is the first of its kind. A key characteristic of IDGNN is that it demonstrably is well-posed, i.e., it is theoretically guaranteed to have a fixed-point representation. We then demonstrate that the standard iterative algorithm often used to train implicit models is computationally expensive in our dynamic setting as it involves computing gradients, which themselves have to be estimated in an iterative manner. To overcome this, we pose an equivalent bilevel optimization problem and propose an efficient single-loop training algorithm that avoids iterative computation by maintaining moving averages of key components of the gradients. We conduct extensive experiments on real-world datasets on both classification and regression tasks to demonstrate the superiority of our approach over the state-of-the-art baselines. We also demonstrate that our bi-level optimization framework maintains the performance of the expensive iterative algorithm while obtaining up to **1600x** speed-up.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**.

KEYWORDS

Dynamic Graphs, Implicit Graph Neural Networks, Graph Convolutional Networks

KDD '24, August 25–29, 2024, Barcelona, Spain

ACM ISBN 979-8-4007-0490-1/24/08
https://doi.org/10.1145/3637528.3672026

ACM Reference Format:
Yongjian Zhong, Hieu Vu, Tianbao Yang, and Bijaya Adhikari. 2024. Efficient and Effective Implicit Dynamic Graph Neural Network. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining (KDD '24), August 25–29, 2024, Barcelona, Spain.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3637528.3672026

1 INTRODUCTION

Graph Convolution Network (GCN) [12] and its subsequent variants [16, 32] have achieved the state-of-the-art performance in predictive tasks in various applications including molecular prediction [23], recommendation [17], and hyperspectral image classification [9]. GCNs have also been extended to the dynamic setting, where the graph changes over time. Even in the dynamic setting, GCNs have achieved state-of-the-art results for numerous tasks including rumor detection [31] and traffic prediction [13].

Despite numerous advantages, a major limitation of existing GCNs is that they can only aggregate information up to k-hops, where k is the depth of the graph convolution operation. Hence, standard graph neural networks [12, 32] cannot capture long-range dependencies beyond a radius imposed by the number of convolution operations used. Trivial solutions like setting k to a large number fail in overcoming this issue as empirical evidence [15] suggests that deepening the layers of GCN, even beyond a few (2-4) layers, can lead to a notable decline in their performance. This is because the stacked GCN layers gradually smooth out the node-level features which eventually results in non-discriminative embeddings (aka oversmoothing). This creates a dilemma where, on the one hand, we would like to capture dependencies between nodes that are far away in the network by stacking multiple layers of GCN together. On the other hand, we also would like to maintain the predictive performance by only using a few layers. To tackle this dilemma in the static setting, Gu et al. [7] proposed an implicit graph neural network (IGNN), which iterates the graph convolution operator until the learned node representations converge to a *fixed-point* representation. Since there is no a priori limitation on the number of iterations, the fixed-point representation potentially contains information from all neighbors in the graph. Evidence shows that it is able to capture long-range dependency while maintaining predictive performance. Following this, other recent works [18, 22] also have addressed the problem in the static setting.

In the case of dynamic graphs, where graphs evolve over time, dynamic graph neural networks aggregate information over the current graph topology and historical graphs to learn meaningful representations [31, 36]. Note the architecture of the graph neural networks within each time stamp in dynamic graph neural networks is similar to existing static GCNs. Hence, the information aggregated for each node in a given time stamp is still limited to a radius of $k-hops$ within the time stamp. Increasing the depth of the GCN operator in each time stamp to ensure long-range dependency exacerbates the performance degradation of dynamic graph neural networks as these models convolve both over the time stamps and within the time stamps, hence oversmoothing the features even faster. Therefore, capturing long-range dependency while improving (or even maintaining) the performance is a big challenge for dynamic graph neural networks. Despite its importance, very few prior works have studied this phenomenon in dynamic graphs: Yang et al. [36] propose an L2 feature normalization process to alleviate the smoothing of features in dynamic graphs and Wang et al. [33] mitigate the oversmoothing problem by emphasizing the importance of low-order neighbors via a node-wise encoder. However, these approaches either rescale features or forget neighborhood information, both of which are not ideal.

To address the challenges mentioned above, we propose IDGNN, an implicit neural network for dynamic graphs derived from the first principles. In designing IDGNN, we encountered multiple challenges including *i)* uncertainty on whether fixed-point (converged) representations exist for implicit neural models defined over dynamic graphs; and *ii)* efficiently training a model to find these fixed-point representations. In this paper, we overcome the first challenge by providing theoretical guarantees on the existence of the fixed-point representations on a single dynamic graph by leveraging a periodic model and generalizing this result to a set of dynamic graphs. For the second challenge, we notice that the stochastic gradient descent via implicit differentiation (often used by other implicit models [7, 14]) is too inefficient in our problem setting. As such, we reformulate our problem as an equivalent bilevel optimization problem and design an efficient optimization strategy. The key contributions of the paper are as follows:

- We propose a novel dynamic graph neural network IDGNN, which ensures long-range dependency while providing theoretical guarantees on the existence of fixed-point representations. IDGNN is the first approach to leverage implicit graph neural network framework for dynamic graphs.
- We present a bilevel optimization formulation of our problem and propose a novel stochastic optimization algorithm to efficiently train our model. Our experiments show that the proposed optimization algorithm is faster than the naive gradient descent by up to 1600 times.
- We conduct comprehensive comparisons with existing methods to demonstrate that our method captures the long-range dependency and outperforms the state-of-the-art dynamic graph neural models on both classification and regression tasks.

2 RELATED WORK

Dynamic Graph Representation Learning: GNN has been successful for static graphs, leading to the development of GNN-based algorithms for dynamic graphs [11]. DyGNN [19] comprises two components: propagation and update, which enable information aggregation and propagation for new interactions. EvolveGCN [21] uses an RNN to update GCN parameters and capture dynamic graph properties. Sankar et. al. [29] propose a Dynamic Self-Attention Network (DySAT) with structural and temporal blocks to capture graph information. TGN [26] models edge streaming to learn node embeddings using an LSTM for event memory. TGAT [35] considers the time ordering of node neighbors. Gao et al. [5] explores the expressiveness of temporal GNN models and introduces a time-then-graph framework for dynamic graph learning, leveraging expressive sequence representations like RNN and transformers.

Implicit Graph Models: The implicit models or deep equilibrium models define their output using fixed-point equations. [1]. propose an equilibrium model for sequence data based on the fixed-point solution of an equilibrium equation. El et al. [4] introduce a general implicit deep learning framework and discuss the well-posedness of implicit models. Gu et al. [7] demonstrate the potential of implicit models in graph representation learning, specifically with their implicit model called IGNN, which leverages a few layers of graph convolution network (GCN) to discover long-range dependencies. Park et al. [22] introduce the equilibrium GNN-based model with a linear transition map, and they ensure the transition map is contracting such that the fixed point exists and is unique. Liu et al. [18] propose an infinite-depth GNN that captures long-range dependencies in the graph while avoiding iterative solvers by deriving a closed-form solution. Chen et al. [2] employ the diffusion equation as the equilibrium equation and solve a convex optimization problem to find the fixed point in their model.

Implicit Models Training: Efficiently training implicit models has always been a key challenge. Normally, the gradient of implicit models is obtained by solving an equilibrium equation using fixed-point iteration or reversing the Jacobian matrix [7]. However, training these models via implicit deferential introduces additional computational overhead. Geng et al. [6] propose a phantom gradient to accelerate the training of implicit models based on the damped unrolling and Neumann series. Li et al. [14] leverage stochastic proximal gradient descent and its variance-reduced version to accelerate the training.

3 METHODOLOGY

3.1 Preliminaries

Dynamic Graphs: We are given a set of N dynamic graphs $\{\mathcal{G}_i\}_{i=1}^{N}$. Each dynamic graph $\mathcal{G}_i = \{G_i^1, ..., G_i^t, ..., G_i^T\}$ is a collection of T snapshots. Let $\mathcal{V}$ denote the union set of nodes that appear in any dynamic graph and $n := |\mathcal{V}|$ be the total number of nodes. Without loss of generality, each snapshot can be represented as $G_i^t = \{\mathcal{V}, \mathcal{E}_i^t, X_i^t\}$ since we can assume each graph is built on the union set $\mathcal{V}$ and treat the absent nodes as isolated. $\mathcal{E}_i^t$ is the set of edges at time t in $\mathcal{G}$. $X_i^t \in \mathbb{R}^{l \times n}$ represents the node attribute matrix, where l is the dimension of the node attributes. Let A_i^t be the adjacency matrix of G_i^t.

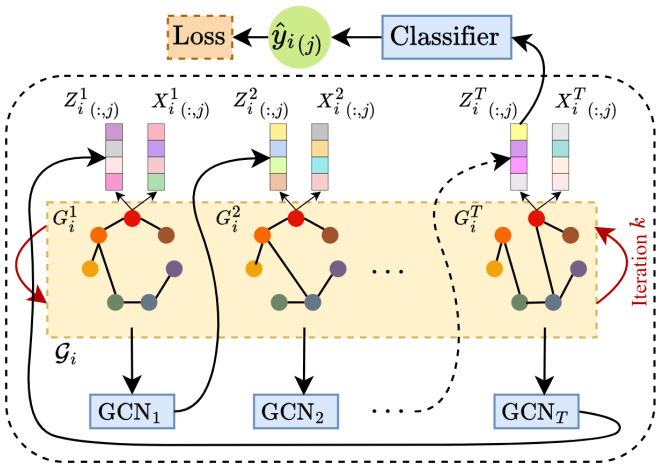


Figure 1: Model overview. This figure indicates the forward process of our model.

In this paper, we focus on node-level tasks (e.g. classification and regression) for dynamic graphs. We consider the dataset as $\{(\mathcal{G}_i, \boldsymbol{y}_i)\}_{i=1}^N$, where $\mathcal{G}_i$ is the i-th dynamic graph, and $\boldsymbol{y}_i \in \mathbb{R}^n$ is the node-level labels assigned to the nodes in the last snapshot of dynamic graph $\mathcal{G}_i$.

Permutation: Here, we define a permutation function to simplify the notation in the rest of the paper. Let $\tau(t) := T - [(T - t + 1) \bmod T]$, where T refers to the number of snapshots. This function maps $t \to t-1$ for $t \in [2, T]$ and maps $t \to T$ for $t = 1$.

Implicit Models: In general, implicit models [4, 6, 7] have the form $Z = f(Z, X)$, where f is a function that can be parameterized by a neural network, X is the data and Z is the learned representation. We can obtain the fixed-point representation via iteration $Z^* = \lim_{k\to\infty} Z_{k+1} = \lim_{k\to\infty} f(Z_k, X) = f(Z^*, X)$.

Thus, the key to designing an implicit model for dynamic graphs is to provide a function f that leads to converged representations.

3.2 Implicit Model for Dynamic Graphs

We first consider a single dynamic graph $\mathcal{G} = \{G^1, ..., G^T\}$ with T snapshots and discover its well-posedness condition in this section. We then generalize the well-posedness conclusion for a set of dynamic graphs later. All the proofs are provided in the Appendix. Now, let us consider the following stacked GCN model:

$$\begin{aligned} Z_{k+1}^1 &= \sigma(W^1 Z_k^{\tau(1)} A^1 + V X^1) \\ Z_{k+1}^2 &= \sigma(W^2 Z_k^{\tau(2)} A^2 + V X^2) \\ &\cdots \\ Z_{k+1}^T &= \sigma(W^T Z_k^{\tau(T)} A^T + V X^T) \end{aligned} \tag{1}$$

In the model presented above, the embeddings Z_{k+1}^2 of the nodes in the second time stamp in the $(k+1)$-th layer depend on the embeddings Z_k^1 of nodes in the first time stamp learned in the k-th layer and the feature of the nodes in the second time stamp X^2. This design enables us to propagate information between time stamps when stacking layers. The parameters for the t-th layer of the model are denoted as $W^t \in \mathbb{R}^{d\times d}$ and $V \in \mathbb{R}^{d\times l}$ with V being a shared weight across all layers. Note that the proposed model and the corresponding theory still hold when V is not shared. We opt for a shared V for simplicity (thorough empirical discussion on this choice is presented in the Experiment section). Following the principle of the implicit model [1, 4, 7], we apply our model iteratively infinite times. If the process converges, we consider the converged result $\{Z_\infty^1, \ldots, Z_\infty^T\}$ as the final embeddings. Consequently, the final embeddings have to satisfy the system of equations in (1) and can be considered a fixed-point solution to (1). However, at this point, it is not clear whether the fixed-point solution always exists for arbitrary graph $\mathcal{G}$.

Well-posedness is a property that an implicit function, such as in (1), possesses a unique fixed point solution. While Gu et al. [7] demonstrated the well-posedness of a single-layer implicit GCN on a single static graph, the question remains open for dynamic graphs. To establish the well-posedness property for our model, we first introduce its vectorized version as follows.

$$\begin{aligned} z_{k+1}^1 &= \sigma(M^1 z_k^{\tau(1)} + \mathbf{vec}(VX^1)) \\ z_{k+1}^2 &= \sigma(M^2 z_k^{\tau(2)} + \mathbf{vec}(VX^2)) \\ &\cdots \\ z_{k+1}^T &= \sigma(M^T z_k^{\tau(T)} + \mathbf{vec}(VX^T)) \end{aligned} \tag{2}$$

where $z = \mathbf{vec}(Z)$ is column-wise vectorization of Z, and $M^i = (A^i)^\top \otimes W^i$ where $\otimes$ is the Kronecker product. Note that Equations (2) can also be expressed in a single matrix form. This transformation involves sequentially connecting the shared nodes between the graphs. Thus, the formula (2) can be reformulated as follows:

$$\begin{bmatrix} z^1 \\ z^2 \\ z^3 \\ \vdots \\ z^T \end{bmatrix} = \sigma\left(\begin{bmatrix} 0 & 0 & \cdots & 0 & M^1 \\ M^2 & 0 & \cdots & 0 & 0 \\ 0 & M^3 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & M^T & 0 \end{bmatrix} \begin{bmatrix} z^1 \\ z^2 \\ z^3 \\ \vdots \\ z^T \end{bmatrix} + \begin{bmatrix} \mathbf{vec}(VX^1) \\ \mathbf{vec}(VX^2) \\ \mathbf{vec}(VX^3) \\ \vdots \\ \mathbf{vec}(VX^T) \end{bmatrix} \right) \tag{3}$$

Here, we omit the subscript for simplicity. Equation (3) represents a single equilibrium form of Equation (2). It can also be viewed as the time-expanded static version [] of our original dynamic graph $\mathcal{G}$. Based on the Banach fixed-point theorem [27], the Equation (3) admits a unique fixed-point if the right-hand side is a contractive mapping w.r.t. z. Therefore, we express the well-posedness condition for our model as follows,

Theorem 3.1. *For any element-wise non-expansive function $\sigma(\cdot)$, the coupled equilibrium equations in (2) have a unique fixed point solution if $\|\mathcal{M}\|_{op} < 1$, where $\mathcal{M}$ define as*

$$\begin{bmatrix} 0 & \cdots & 0 & M^1 \\ M^2 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & \cdots & M^T & 0 \end{bmatrix}$$

and $\|\mathcal{M}\|_{op}$ is the operator norm of $\mathcal{M}$, which is the largest absolute eigenvalue. Furthermore, this is equivalent to $\|M^t\|_{op} < 1$ for any $t = 1, ..., T$.

In order to maintain $\|M^t\|_{op} < 1, \forall t = 1, ..., T$, it is necessary to ensure that the condition $\lambda_{\text{pr}}(|W^t|)\lambda_{\text{pr}}(A^t) < 1$ is satisfied, where $\lambda_{\text{pr}}(\cdot)$ represents the Perron-Frobenius eigenvalue. However, satisfying the condition is challenging in general as it is hard to track the eigenvalue as the underlying matrix changes. To overcome this challenge, we impose a more stringent requirement on W which is more easily enforceable by leveraging a convex projection. We formally state this in the following theorem.

Theorem 3.2. *Let σ be an element-wise non-expansive function. If the coupled equilibrium equations satisfy the well-posedness condition, namely $\|\mathcal{M}^t\|_{op} \leq \|W^t\|_{op}\|A^t\|_{op} < 1, \forall t = 1, ..., T$, then there exists rescale coupled equilibrium equations, which satisfy the condition $\|W^t\|_\infty\|A^t\|_{op} < 1, \forall t = 1, ..., T$, and the solutions of these two equations are equivalent.*

Proof. Suppose $\{W^t\}$ satisfy $\|W^t\|_{op}\|A^t\|_{op} < 1$ for all t, then the following equation has a unique fixed point.

$$\begin{bmatrix} z^1 \\ z^2 \\ z^3 \\ \vdots \\ z^T \end{bmatrix} = \sigma\left(\begin{bmatrix} 0 & 0 & \cdots & 0 & M^1 \\ M^2 & 0 & \cdots & 0 & 0 \\ 0 & M^3 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & M^T & 0 \end{bmatrix}\begin{bmatrix} z^1 \\ z^2 \\ z^3 \\ \vdots \\ z^T \end{bmatrix} + \begin{bmatrix} \mathbf{vec}(VX^1) \\ \mathbf{vec}(VX^2) \\ \mathbf{vec}(VX^3) \\ \vdots \\ \mathbf{vec}(VX^T) \end{bmatrix}\right)$$

and this condition implies $\|\mathcal{M}\|_{op} \leq 1$. Based on Theorem 4.3, there exists a set of diagonal matrices $\{S^t\}$ such that

$$\hat{W}^t = S^t W^t (S^t)^{-1}, \hat{V} = S^t V$$

Then the fixed-point of following equations

$$\begin{bmatrix} \hat{z}^1 \\ \hat{z}^2 \\ \hat{z}^3 \\ \vdots \\ \hat{z}^T \end{bmatrix} = \sigma\left(\begin{bmatrix} 0 & 0 & \cdots & 0 & \hat{M}^1 \\ \hat{M}^2 & 0 & \cdots & 0 & 0 \\ 0 & \hat{M}^3 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & \hat{M}^T & 0 \end{bmatrix}\begin{bmatrix} \hat{z}^1 \\ \hat{z}^2 \\ \hat{z}^3 \\ \vdots \\ \hat{z}^T \end{bmatrix} + \begin{bmatrix} \mathbf{vec}(\hat{V}X^1) \\ \mathbf{vec}(\hat{V}X^2) \\ \mathbf{vec}(\hat{V}X^3) \\ \vdots \\ \mathbf{vec}(\hat{V}X^T) \end{bmatrix}\right)$$

satisfies the following relation

$$\begin{bmatrix} \hat{z}^1 \\ \vdots \\ \hat{z}^T \end{bmatrix} = \begin{bmatrix} (S^1)^{-1} & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & (S^T)^{-1} \end{bmatrix}\begin{bmatrix} \hat{z}^1 \\ \vdots \\ \hat{z}^T \end{bmatrix}$$

□

As previously stated, the theorem is formulated for a single dynamic graph. In the following Remark, we present a broader and more general conclusion for a set of dynamic graphs.

Remark 1. *Considering a set of dynamic graphs denoted as $\{\mathcal{G}_i\}_{i=1}^N$, achieving fixed-point representations for all dynamic graphs using our model is guaranteed if the condition $\|W^t\|_\infty\|A_i^t\|_{op} < 1$ holds for all time steps $t = 1, ..., T$ and all graph indices $i = 1, ..., N$.*

Proof. Given a set of dynamic graphs $\{\mathcal{G}_i\}_{i=1}^N$, we can construct a single dynamic graph by merging the snapshots that are from the same time stamp, then we obtain

$$\hat{\mathcal{G}} = \{[G_i^1, ..., G_N^1], ..., [G_i^T, ..., G_N^T]\} \tag{4}$$

Let $\hat{A}^t$ denote the adjacency matrix of $[G_i^t, ..., G_N^t]$. By theorem 3.2, we need to ensure $\|W^t\|_\infty\|\hat{A}^t\|_{op} < 1$. Since $\hat{A}^t$ contains N disconnected graphs, $\|\hat{A}^t\|_{op} \leq \max_i \|A_i^t\|_{op}$, which means $\|W^t\|_\infty\|A_i^t\|_{op} < 1$ needed to be satisfied for all i. Since t is arbitrary, the remark holds. □

In practice, we can track the matrices with the largest operator norms ($\max_i \|A_i^t\|_{op}$) at each time step. Focusing on these "critical" matrices and their corresponding constraints ensures our model meets the required conditions.

Incorporating Downstream Tasks: Based on the established conditions, we can obtain the fixed-point representation by iteratively applying our model. Now, we want the fixed-point representations suited for specific downstream tasks with their own optimization objective. Let us now introduce the comprehensive objective which incorporates both the application loss and the convergence requirements mentioned above. To this end, we utilize a neural network $f_\theta(.)$, parameterized by θ, to map graph embeddings to their respective targets. Let $\mathcal{W} := \{W^1, ..., W^T\}$. The comprehensive objective can now be summarized as follows:

$$\begin{aligned} \min_{\theta,\mathcal{W},V} \mathcal{L}(\theta, \mathcal{W}, V) &= \sum_{i=1}^N \ell(f_\theta(z_i^T), \boldsymbol{y}_i) \\ \text{s.t. } z_i^1 &= \sigma\left(\left((A_i^1)^\top \otimes W^1\right) z_i^{\tau(1)} + \mathbf{vec}(VX_i^1)\right) \\ &\cdots \\ z_i^T &= \sigma\left(\left((A_i^T)^\top \otimes W^T\right) z_i^{\tau(T)} + \mathbf{vec}(VX_i^T)\right), \\ \|W^t\|_\infty &\leq \frac{\kappa}{\|A_i^t\|_{op}}, \forall i = 1, ..., N, t = 1, ..., T \end{aligned} \tag{5}$$

Where ℓ is a loss function (e.g. cross entropy loss, mean square error), and κ is a positive number that is close to 1, which is added for numerical stability.

To find the fixed-point representation (i.e. forward pass), we can use fixed-point iteration or other root-finding algorithms. Backpropagation requires storing intermediate results, which is infeasible since there might be hundreds of iterations in discovering the representation. Thus, the key challenge in solving Equation (5) lies in determining how to perform backpropagation effectively, especially in the context of dynamic graphs.

4 TRAINING

In this section, we provide two algorithms to solve Equation (5). We first explore the stochastic gradient descent (SGD) method where we estimate the gradients leveraging the Implicit Function Theorem. This approach offers several advantages, such as eliminating the need to store intermediate results during the forward pass and enabling direct backpropagation through the equilibrium point. While widely used in various techniques [1, 7], this approach presents certain drawbacks when applied to our specific model, particularly regarding computational overhead. Subsequently, we introduce an efficient training algorithm for our model, which adopts a bilevel viewpoint of our problem. This novel approach allows us to overcome the limitations of the vanilla SGD method, resulting in improved computational efficiency during training.

4.1 SGD with Implicit Differentiation

The algorithm operates as follows: it first finds the fixed-point embedding through iteration and then computes the gradient based on this embedding. It then updates the weights using SGD. The main obstacle in our way is estimating the gradient.

The parameters that need to be updated are W, V (from GCN layers), and θ (from the classifier). The gradient with respect to parameter θ can be obtained as $\frac{\partial \mathcal{L}}{\partial \theta}$, which can be computed using autograd functions given the fixed point. However, computing the gradient for other parameters presents a greater challenge. Let $\frac{\partial \mathcal{L}}{\partial P_i}$ represent the gradient with respect to W_i or V_i. Then, the gradient is computed as $\frac{\partial \mathcal{L}}{\partial P_i} = \sum_{j=1}^{T} \frac{\partial \mathcal{L}}{\partial z_j} \frac{\partial z_j}{\partial P_i}$. The computation of $\frac{\partial \mathcal{L}}{\partial z_j}$ can be achieved through the autograd mechanism. However, determining $\frac{\partial z_j}{\partial P_i}$ is non-trivial due to the cyclic definition of z_j.

By the definition of our model, the learned embeddings must satisfy the following equations:

$$\begin{aligned} z^1 - \sigma(M^1 z^{\tau(1)} + \mathbf{vec}(VX^1)) &= 0 \\ z^2 - \sigma(M^2 z^{\tau(2)} + \mathbf{vec}(VX^2)) &= 0 \\ &\vdots \\ z^T - \sigma(M^T z^{\tau(T)} + \mathbf{vec}(VX^T)) &= 0 \end{aligned} \tag{6}$$

We apply column-wise vectorization on matrices W and V respectively to obtain w and v. We can then calculate the gradients $\frac{\partial z}{\partial w}$ and $\frac{\partial z}{\partial v}$ using the implicit function theorem. Here, we will show the details of the derivation.

Let $Z^t_{ij}, A^t_{ij}, X^t_{ij}$ and W^t_{ij} denote the element at the i-th row and j-th column of Z^t, A^t, X^t and W^t, respectively. Taking the derivative of element-wise form of Equation (6), $Z^t_{ij} - \sigma(\sum_l \sum_n W^t_{il} Z^{\tau(t)}_{ln} A^t_{nj} + \sum_l V_{il} X^t_{lj}) = 0$, with respect to W^a_{bc} we obtain

$$\frac{\partial Z^t_{ij}}{\partial W^a_{bc}} - \Sigma^t_{ij} \left(\sum_n \delta_{at} \delta_{bi} Z^{\tau(t)}_{cn} A^t_{nj} + \sum_l \sum_n W^t_{il} \frac{\partial Z^{\tau(t)}_{ln}}{\partial W^a_{bc}} A^t_{nj} \right) = 0$$

where δ_{at} is the indicator function which equals 1 only when $a = t$, and $\odot$ denotes element-wise multiplication. Let $\sigma'(.)$ represent the derivative $\sigma(.)$, and we define

$$\Sigma^t_{ij} := \sigma'\left(\sum_l \sum_n W^t_{il} Z^{\tau(t)}_{ln} A^t_{nj} + \sum_l V_{il} X^t_{lj}\right)$$

which means Σ^t is a matrix and has a shape like Z^t. Let $H^t := (Z^{\tau(t)} A^t)^T$, and $H^t_{\cdot c}$ denotes the c-th column of H^t. Let e_b be a column vector with value 1 at b-th entry and 0 elsewhere. Enumerating Z^t_{ij} for all i, j, we have

$$\frac{\partial z^t}{\partial W^a_{bc}} - \mathbf{vec}(\Sigma^t) \odot \left(\delta_{at} e_b \otimes H^t_{\cdot c} + M^t \frac{\partial z^{\tau(t)}}{\partial W^a_{bc}} \right) = 0$$

Therefore, for any time stamp t, the gradient of z^t with respect to the a-th layer of GCN, w^a, can be expressed as:

$$\frac{\partial z^t}{\partial w^a} - \Xi^t \odot \left(\delta_{at} H^t \otimes I + M^t \frac{\partial z^{\tau(t)}}{\partial w^a} \right) = 0 \tag{7}$$

Where Ξ^t is a matrix that has identical column vectors, and each column vector is the $\mathbf{vec}(\Sigma^t)$. Similarly, we can compute the gradient of Z^t_{ij} w.r.t. V_{bc}.

$$\frac{\partial Z^t_{ij}}{\partial V_{bc}} - \Sigma^t_{ij} \left(\delta_{bi} X^t_{cj} + W^t_{il} \frac{\partial Z^{\tau(t)}_{ln}}{\partial W^a_{bc}} A^t_{nj} \right) = 0$$

Therefore,

$$\frac{\partial z^t}{\partial v} - \Xi^t \odot \left((X^t)^\top \otimes I + M^t \frac{\partial z^{\tau(t)}}{\partial v} \right) = 0 \tag{8}$$

While these equations provide a path for gradient computation, it is essential to note that all gradients are interconnected within a system of equations. The resolution of such a system entails a substantial computational overhead.

Per-iteration Complexity of naive gradient descent: Equations (7) and (8) reveal that a set of equilibrium equations determines the gradients. Consequently, to compute the gradients, we need to solve these equations using fixed-point iteration. Each layer necessitates one round of fixed-point iteration, and in total, including V, we need to perform fixed-point iteration $T + 1$ times. The major computational overhead arises from the multiplication of M with the derivatives, resulting in a complexity of $O((nd)^2 d^2)$. Each fixed-point iteration involves T instances of such computations. Consequently, the overall runtime for each update is $O(T^2 n^2 d^4)$. Although the adjacency matrix is sparse, it only reduces the complexity to $O(T^2 n d^4)$. This limits the application of our model in large-scale dynamic graphs and hampers our ability to utilize large embeddings.

4.2 Efficient Update via Bilevel Optimization

To address the previously mentioned challenges, we turn to Bilevel Optimization [3] as a potential solution. We reformulate the problem presented in Equation (5) as the following standard bilevel optimization problem.

$$\begin{aligned} \min_{\theta, \mathcal{W}, V} \mathcal{L}(\theta, \mathcal{W}, V) &= \sum_{i=1}^{N} \ell(f_\theta(z_i^T, \boldsymbol{y}_i) \\ \text{s.t. } z_i^T &= \arg\min_z \|z - \phi(z, \mathcal{W}, V; \mathcal{G}_i)\|_2^2 \\ \|W^t\|_\infty &\le \frac{\kappa}{\|A_i^t\|_{op}}, \forall i = 1, ..., N, t = 1, ..., T \end{aligned} \tag{9}$$

Where $\phi(z, \mathcal{W}, V; \mathcal{G}_i) = \sigma(M_i^T ... \sigma(M_i^1 z + \mathbf{vec}(VX_i^1)) ... + \mathbf{vec}(VX_i^T))$. The main differences between these problems lie in the constraints. Equation (9) introduces explicit constraints solely on the last snapshot, leading to a multi-block bilevel optimization problem. This type of problem has been investigated recently by [25] and [10]. [25] focus on top-K NDCG optimization, formulating it as a compositional bilevel optimization with a multi-block structure. Their approach simplifies updates by sampling a single block batch in each iteration and only updating the sampled blocks. [10] employs a similar technique but addresses a broader range of multi-block min-max bilevel problems.

However, these state-of-the-art bilevel optimization algorithms are designed to address problems with strongly convex lower-level

Algorithm 1 Bilevel Optimization for IGDNN

Require: $\mathcal{D} = \{(\mathcal{G}_i, \boldsymbol{y}_i)\}_{i=1}^N, \eta_1, \eta_2, \gamma$
Ensure: ω, θ
1: Randomly initialize z_j^0 and v_j^0 for $j = 1, ..., N$
2: **for** $k = 0, 1, ..., K$ **do**
3: Sample a batch data $\mathcal{B}$
4: $\hat{z}_j^{k+1} = \begin{cases} (I - \eta_1)\hat{z}_j^k + \eta_1 \phi(\hat{z}_j^k, \omega^t; \mathcal{G}_i) & j \in \mathcal{B} \\ \hat{z}_j^t & \text{o.w.} \end{cases}$
5: $\hat{v}_j^{k+1} = \begin{cases} (I - \eta_2 \nabla_{zz}^2 g(\hat{z}_j^k, \omega^k))\hat{v}_j^k + \eta_2 \nabla_z \ell_j(\hat{z}_j^k, \omega^k) & j \in \mathcal{B} \\ \hat{v}_j^t & \text{o.w.} \end{cases}$
6: Update gradient estimator
$$\Delta^{k+1} = \frac{1}{|\mathcal{B}|} \sum_{j \in \mathcal{B}} \left[\nabla_\omega \ell_j(\hat{z}_j^k, \omega^k) - \nabla_{\omega z}^2 g_j(\hat{z}_j^k, \omega^k)\hat{v}_j^k \right]$$
7: $m^{k+1} = (1 - \gamma)m^k + \gamma \Delta^{k+1}$
8: $\omega^{k+1} = \Pi_\Omega \left(\omega^k - \eta_0 m^{k+1} \right)$
9: **end for**

problems, which does not hold true for our problem. It is evident that our lower-level problems in 9 are generally nonconvex with respect to z since they involve highly nonlinear neural networks. Additionally, these methods utilize stochastic gradient descent on the lower level in each iteration, which may lead to potential extra computation in estimating the gradient. Nevertheless, it is crucial to note that the optimal solution to our lower-level problem corresponds to the fixed point of Eq (2). Leveraging this insight, we employ a fixed-point iteration to update the lower-level solution. We propose a single loop algorithm (see Algorithm 1) with fixed-point updates.

To better illustrate our algorithm, let $\omega = \{\mathcal{W}, V\}$, and $g_i(z, \omega)$ represents the i-th-block lower-level problem, defined as $\|z - \phi(z, \omega; \mathcal{G}_i)\|_2^2$, and let $\ell_i(z, \omega) := \ell(f_\theta(z), y_i)$. The key to solving the bilevel optimization problem in (9) is to estimate the hypergradient $\nabla \mathcal{L}(\theta, \omega)$ and backpropagate. The hypergradient of our objective in (9) with respect to ω as follows:

$$\begin{aligned} \nabla \mathcal{L}(\theta, \omega) =& \frac{1}{N} \sum_{i=1}^N \nabla \ell_i(z_i^T, \omega) \\ & - \nabla_{\omega z}^2 g_i(z_i^T, \omega) \left[\nabla_{zz}^2 g_i(z_i^T, \omega) \right]^{-1} \nabla_z \ell_i(z_i^T, \omega) \end{aligned}$$

Explicitly computing the inverted Hessian matrix $\left[\nabla_{zz}^2 g_i(z_i^T, \omega)\right]^{-1}$ in computation of our hypergradient is computationally expensive. Inspired by [10] and [25], we instead directly approximate $\left[\nabla_{zz}^2 g_i(z_i^T, \omega)\right]^{-1} \nabla_z \ell_i(z_i^T, \omega)$ using v_i for each block by moving average estimation (line 5). More specifically, we track the optimal point of $\min_v \frac{1}{2} v^\top \nabla_{zz}^2 g_i(z_i^T, \omega) v - v^\top \nabla_z \ell_i(z_i^T, \omega)$ for each block by maintaining v_i. Let $\hat{z}_i$ be a moving average approximation to the optimal lower-level solution z_i^T. We use a single fixed-point update for $\hat{z}_i$ (line 4). We do not want to update all blocks in every iteration since this is impractical when the number of blocks is large. To address this issue, we use stochastic training. For sampled blocks, we update their $\hat{z}$ and $\hat{v}$, and we compute the hypergradient (line 6). Note that the multiplication $\nabla_{\omega z}^2 g_j(\hat{z}_j^k, \omega^k)\hat{v}_j^k$ (in line 6) can also be efficiently computed using Hessian vector product [24]. .As a result, the training time for our algorithm is proportional to normal backpropagation, eliminating the need for fixed-point iterations.

Note that in cases where the lower-level problem is strongly convex, the errors introduced by these approximations are well-contained [10]. Our lower-level problem admits a unique fixed point, hence, employing fixed-point iteration becomes an efficient means of attaining the optimal lower-level solution, akin to the effectiveness of gradient descent under strong convexity. Therefore, it is justifiable to assert that our approximations are effective in this scenario, with empirical evidence robustly endorsing their practical efficacy.

Per-iteration Complexity of bilevel optimization: The main computational overheads are updating v and estimating gradient. Both steps are involved with estimating a huge Hassian matrix, but, in practice, we use Hessian vector product to avoid explicitly computing the Hessian matrix. Therefore, the dominant runtime of bilevel optimization is three times backpropagation. Each backpropagation takes $O(Tnd^2 + Tn^2d)$.

5 EXPERIMENTS

In this section, we present the performance of IDGNN in various tasks, including effectiveness in capturing long-range dependencies and avoiding oversmoothing on a synthetic dataset. We benchmark IDGNN against nine state-of-the-art baselines on multiple real-world datasets, comprising three node classification and four node regression datasets. Key dataset statistics are summarized in Table 2, with further details available in section 5.1. Due to the space constraints, specifics on experimental setup and hyperparameters are provided in Appendices A.1 and A.2 respectively. Our code is publicly available for reproducibility. [1] .

5.1 Datasets

Brain dataset is derived from a real-world fMRI brain scans dataset [2]. In this dataset, nodes represent brain tissues, and edges capture nodes' activation time similarity in the examined period. The node attributes are generated from the fMRI signals [34]. Given the temporal graph, our goal is to predict the functionality of each node, which is in one of the **10** categories.

DBLP is an extracted co-author network from DBLP website [3], where nodes are authors and edges represent co-authorship relationships. The extracted authors are from **5** research areas [34], serving as our class labels. Note attributes are word2vec representations of titles and abstracts from related papers.

Reddit dataset is extracted from a popular online platform Reddit [4] [8], where nodes represent posts, and edges are constructed between nodes having similar keywords. The node attributes are word2vec representations of the comments of a post [34], and node labels correspond to one of the **4** communities or "subreddits" to which the post belongs.

PeMS04 & PeMS08 These two datasets represent traffic sensor networks in two distinct districts of California during various months.

[1] https://github.com/yongjian16/IDGNN
[2] https://tinyurl.com/y4hhw8ro
[3] https://dblp.org/
[4] https://www.reddit.com/

Table 1: Performance for classification task (ROCAUC) and regression task (MAPE (%)). Performances on Brain10, England-COVID, PeMS04, and PeMS08 for baseline methods are taken from [5]. The best performance for each dataset is highlighted in bold, while the second-best performance is underlined.

Model	Classification			Regression					
	Brain10	DBLP5	Reddit4	England-COVID		PeMS04		PeMS08	
				Trans.	Induc.	Trans.	Induc.	Trans.	Induc.
EvolveGCN-O	0.58±0.10	0.639±0.207	0.513±0.008	4.07±0.73	3.88±0.47	3.20±0.25	2.61±0.42	2.65±0.12	2.40±0.27
EvolveGCN-H	0.60±0.11	0.510±0.013	0.508±0.008	4.14±1.14	3.50±0.42	3.34±0.14	2.84±0.31	2.81±0.28	2.81±0.23
GCN-GRU	0.87±0.07	0.878±0.017	0.513±0.010	3.56±0.26	2.97±0.34	1.60±0.14	1.28±0.04	1.40±0.26	1.07±0.03
DySAT-H	0.77±0.07	**0.917±0.007**	0.508±0.003	3.67±0.15	3.32±0.76	1.86±0.08	1.58±0.08	1.49±0.08	1.34±0.03
GCRN-M2	0.77±0.04	0.894±0.009	0.546±0.020	3.85±0.39	3.37±0.27	1.70±0.20	1.20±0.06	1.30±0.17	1.07±0.03
DCRNN	0.84±0.02	0.904±0.013	0.535±0.007	3.58±0.53	3.09±0.24	1.67±0.19	1.27±0.06	1.32±0.19	1.07±0.03
TGAT	0.80±0.03	0.895±0.003	0.510±0.011	5.44±0.46	5.13±0.26	3.11±0.50	2.25±0.27	2.66±0.27	2.34±0.19
TGN	0.91±0.03	0.887±0.004	0.521±0.010	4.15±0.81	3.17±0.23	1.79±0.21	1.19±0.07	1.49±0.26	0.99±0.06
GRU-GCN	0.91±0.03	0.906±0.008	0.525±0.006	3.41±0.28	**2.87±0.19**	1.61±0.35	1.13±0.05	1.27±0.21	0.89±0.07
IDGNN	**0.94±0.01**	0.907±0.005	**0.556±0.017**	**2.65±0.25**	3.05±0.25	**0.53±0.05**	**0.63±0.04**	**0.45±0.11**	**0.50±0.05**

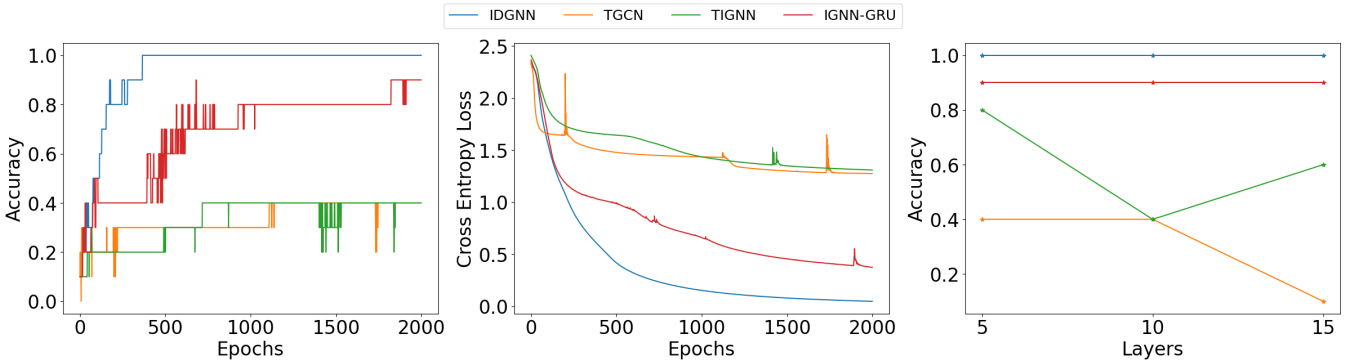


Figure 2: The left and middle are accuracy and loss curves when using 10 layers. The x-axis is epochs, and the y-axis is accuracy and cross entropy loss, respectively. The right plot represents the accuracy results of all baselines under different layer settings.

Table 2: Statistics of datasets. N: number of dynamic graphs, $|V|$: number of nodes, $\min|E|$: minimum number of edges, $\max|E|$: maximum number of edges, T: window size, d: feature dimension, y label dimension

	N	$\|V\|$	$\min\|E\|$	$\max\|E\|$	T	d	y
Brain10	1	5000	154094	167944	12	20	10
DBLP5	1	6606	2912	5002	10	100	5
Reddit4	1	8291	12886	56098	10	20	4
PeMS04	16980	307	680	680	12	5	3
PeMS08	17844	170	548	548	12	5	3
England-COVID	54	129	836	2158	7	1	1

In this context, each node symbolizes a road sensor, while the edge weights denote the geographical distance between two sensors (with uniform weights across all snapshots). Every sensor captures average traffic statistics such as flow, occupancy, and speed over a 5-minute interval. The datasets utilized in this study are identical to those in [5], where we exclusively forecast attribute values for a single future snapshot.

England-COVID This temporal graph dataset is taken from [5], which is created by aggregating information from the mobility log released by Facebook under Data For Good program [5] for England [20]. The nodes are cities, and the edges are transportation statistics between them. Given a temporal graph of length 7, which represents the a week, we need to predict the infection rate for the next day.

5.2 Regression

For node-level tasks, there are two evaluation paradigms: transductive and inductive. Transductive evaluation allows the model to access identities of the nodes and edges in testing data during training (i.e., only the labels are hidden), while inductive evaluation involves testing the model on new nodes and edges. In simpler terms, transductive evaluation separates training and testing sets based on nodes, while inductive evaluation separates them based on time stamps (i.e., models are trained on past time stamps and tested on the future). Here, we conduct experiments on both settings.

The datasets we used for the regression task are England-COVID, PeMS04, and PeMS08. We use the mean average percentage error

[5] https://dataforgood.fb.com/tools/disease-prevention-maps/

Table 3: Memory and runtime comparison results for all methods on reddit4 and DBLP5 datasets. We report the memory usage using MB and runtime using seconds per window

.

	Reddit4		DBLP5	
	Mem.	Time	Mem.	Time
EvolveGCN-O	**42**	0.0649±0.017	**52**	0.0672±0.014
EvolveGCN-H	52	0.0904±0.020	82	0.0997±0.037
GCN-GRU	221	0.0733±0.012	200	0.1142±0.045
DySAT-H	181	0.1613±0.056	165	0.1343±0.012
GCRN-M2	322	0.4345±0.080	319	0.4934±0.076
DCRNN	223	0.1697±0.019	278	0.2121±0.039
TGAT	793	0.0750±0.014	338	0.0770±0.015
TGN	450	0.0417±0.004	233	0.0454±0.012
GRU-GCN	4116	**0.0199±0.008**	580	**0.0161±0.007**
IDGNN	89	0.0291±0.007	75	0.0302±0.002

(MAPE) as our evaluation metric. The results are reported in Table 1 with mean MAPE and standard deviation. Our proposed method outperforms other methods in both transductive and inductive settings, with the exception of the inductive case in England-COVID. Our method demonstrates a significant improvement for PeMS04 and PeMS08, particularly in the transductive learning scenario. In comparison to the second-best method, our proposed model reduces the error by over 1% of MAPE, but our model on the inductive learning scenario does not enjoy such improvement. We attribute this difference to our model's tendency to separate nodes, even when they have the same labels and topology. We delve into this phenomenon in the Appendix B.

5.3 Classification

We conduct classification experiments on Brain10, DBLP5, and Reddit4 datasets. Since these datasets consist of only one dynamic graph, we focused on testing the transductive case. The evaluation was done using the Area under the ROC Curve (AUC) metric. The average prediction AUC values and their corresponding standard deviations are presented in Table 1. Our proposed model achieved the top rank in 2 out of 3 datasets and was the second best in the remaining dataset. These results demonstrate that our model successfully captures the long-range dependencies within the dynamic graphs, as reflected in the learned embeddings.

5.4 Long-range Dependency and Oversmoothing

Long-range Dependency: Here we first consutrct a toy data that aims to test the ability of all approaches to capture long-range dependencies. Our toy data consists of $\{5, 10, 15\}$ snapshots, with each snapshot being a clique of 10 nodes. Each node has 10 associated attributes. The task is to classify nodes at the last snapshot, where each node represents its own class (i.e., there are a total of 10 classes). The node attributes consist of randomly generated numbers, except for the first snapshot, which uses the one-hot representation of the class. Successful classification of this dataset requires effective information aggregation starting in the initial time stamp, propagating the class label information over time, and avoiding oversmoothing as the information is propagated. In this dataset, there are no testing nodes; all nodes are used for training.

The training results are presented on Figure 2. Our model is compared with TGCN [37]. We also propose two more modified baselines: IGNN-GRU and TIGNN, which are obtained by replacing the GCNs within GCN-GRU [30] and TGCN by IGNN. We ensure the comparison is fair by ensuring a similar number of parameters are used, and we test all models on $\{5, 10, 15\}$ layers. All methods are trained for a maximum of 2000 epochs, followed by the hyper-parameter selection approach described in the Appendix. As shown in the figure, we explored the impact of varying the number of layers on both baseline models and our proposed model. We documented the resulting accuracies accordingly. Notably, TGCN exhibits a discernible pattern of over-smoothing, evidenced by a performance decline with increasing layers. Conversely, IGNN-GRU and TIGNN do not demonstrate such susceptibility. Additionally, we observed challenges in data fitting for both methods, whereas our model consistently achieved 100% accuracy across different layer configurations. This experiment underscores the robustness of our architecture in fully unleashing the potential of implicit models in dynamic graph learning.

Moreover, we perform an alternative experiment, wherein we shift label information from snapshot 1 to snapshot 5. This adjustment ensures consistent difficulty in leveraging label information across all models. Due to space constraints, we provide the detailed results in the Appendix; however, the overall conclusion remains unaffected.

Oversmoothing: Here, we employ Dirichlet energy to assess the smoothness of the learned embeddings [28]. The Dirichlet energy measures the mean distance between nodes and is defined as follows:

$$DE(Z) = \sqrt{\frac{1}{|\mathcal{V}|} \sum_{i \in \mathcal{V}} \sum_{j \in \mathcal{N}_i} \|Z_i - Z_j\|^2}$$

Here, $\mathcal{N}_i$ denotes the neighbors of node i. Low Dirichlet energy means smooth embedding. We compare our model with two different baselines:

- Static: We create a static graph by averaging the adjacency matrices of all snapshots and using the feature matrix of the last snapshot as its feature. Subsequently, we learn the embeddings using a vanilla multi-layer GCN.
- W/o loop: This model shares the structure of IDGNN but does not enforce the learned embeddings to be a fixed point.

We evaluate these methods on the Brain10 dataset, and the results are presented in Table 4. The "static" method exhibits oversmoothing as the layer count increases, and a similar issue is observed for the "W/o loop" method, despite stacking one layer for each snapshot. These findings strongly indicate that our model effectively addresses oversmoothing in dynamic graph learning by leveraging the implicit model structure.

5.5 Efficiency

We compare runtime and performance between SGD and bilevel optimization algorithms. To this end, we compare Brain10, England-COVID, PeMS04, and PeMS08. The results are summarized on the

Table 4: Evaluating the smoothness of embeddings and the AUC on Brain10.

	Dirichlet Energy ↑	AUC ↑
Static (3 layers)	10.660	89.79
Static (4 layers)	8.781	88.67
Static (5 layers)	3.399	81.49
W/o looping	8.957	85.61
IDGNN	**35.299**	**94.87**

Table 5: Runtime and performance comparison between SGD and Bilevel Methods.

	Runtime (s/win)		Performance	
	SGD	Bilevel	SGD	Bilevel
Brain10	624	**0.390**	94.7	**94.9**
PeMS04	0.72	**0.049**	**0.63**	**0.63**
PeMS08	0.29	**0.046**	0.56	**0.50**
England-COVID	0.092	**0.030**	**2.97**	3.05

Table 5. The results are computed by averaging the runtime of a whole epoch with the number of dynamic graphs N.

These methods have similar performance, but the runtime results show that the bilevel optimization algorithm is much faster than the SGD. Especially, in the Brain10 dataset, bilevel algorithm achieves *1600* times of speedup compared with SGD. Furthermore, we notice that the ratio of runtimes in PeMS04 and PeMS08 is $\frac{0.72}{0.29} = 2.48$, and the squared ratio of their number of nodes is $(\frac{307}{170})^2 = 3.26$. This confirms our complexity result for SGD, which is quadratic regarding the number of nodes. On the other hand, the bilevel method exhibits only linear dependency. We also present the memory usage and runtimes of all methods on Reddit4 and DBLP5. The memory efficiency of implicit models comes from the fact that implicit models can use few parameters and do not need to store the intermediate results. However, we need to store intermediate results and backpropagate for our bi-level method. Due to the simple RNN-free architecture of our method, our approach is competitive in runtime and memory. We provide a memory and runtime comparison on DBLP5 and Reddit4. The results are summarized in Table 3.

5.6 Ablation Study

In this section, we delve into the various configurations of our model. Drawing from the properties mentioned earlier, our model can be represented by the formula (1). We have deliberately decided to assign different weights (W) to each timestamp while maintaining weight-tied for V. Alternatively, the model comprises T layers of GCN and one linear layer. The linear layer serves to aggregate static information, while the GCNs handle dynamic information. To validate our architecture choice, we conducted a thorough comparison of our model against other configurations:

- Share both: Both W and V are shared across layers.
- Share V: Only V is shared across layers.
- Not share: W and V are not shared.
- W/o loop: as defined in Section 5.4.

The results are presented in Table 6. As observed, the share-both model exhibits the poorest performance. We believe this is due to the limited number of free variables available for learning, which makes the training process challenging. In our approach and the share-V method, we achieve very similar results. Our model utilizes T layers of GCN for dynamic information and one linear layer for static information, while the share-V method employs one GCN and T linear layers. The not-share method achieves the best result, although the improvement is negligible. However, it increases the parameter size, resulting in significant computational overhead. Hence, we opt for the current configuration, as it delivers good performance while minimizing parameter size, especially since the number of attributes may exceed the hidden dimension. We additionally evaluate an alternative baseline, denoted as "w/o loop," wherein we eliminate the looping structure from IDGNN. The obtained results reveal that this model exhibits the lowest performance, underscoring the efficacy of our proposed approach.

Table 6: Evaluating different variants of our model. Presenting AUC results on Brain10 datasets

	Share both	IDGNN	Share V	Not share	W/o loop
AUC	94.29	94.87	94.87	94.90	85.62

6 CONCLUSIONS

In this paper, we proposed a novel implicit graph neural network for dynamic graphs. As far as we know, this is the first implicit model on dynamic graphs. We demonstrate that the implicit model we proposed has the well-posedness characteristic. We proposed a standard optimization algorithm using the Implicit Function Theorem. However, the optimization was too computationally expensive for our model. Hence, we proposed a novel bilevel optimization algorithm to train our proposed model. We conducted extensive experiments on 6 real-world datasets and one toy dataset. The regression and classification tasks show that the proposed approach outperforms all the baselines in most settings. Finally, we also demonstrated that the proposed bilevel optimization algorithm obtains significant speedup over standard optimization while maintaining the same performance. A key limitation of our proposed approach is that it can only predict the consecutive snapshot. In the future, we plan on addressing this issue and also provide a diffusion model-based training algorithm.

7 ACKNOWLEDGEMENT

We are grateful to the anonymous reviewers for their constructive comments and suggestions. This work was supported in part by the NSF Cybertraining 2320980, SCH 2306331, CAREER 1844403 and the CDC MInD Healthcare group under cooperative agreement U01-CK000594.

REFERENCES

[1] Shaojie Bai, J Zico Kolter, and Vladlen Koltun. 2019. Deep equilibrium models. *Advances in Neural Information Processing Systems* 32 (2019).

[2] Qi Chen, Yifei Wang, Yisen Wang, Jiansheng Yang, and Zhouchen Lin. 2022. Optimization-induced graph implicit nonlinear diffusion. In *International Conference on Machine Learning*. PMLR, 3648–3661.

[3] Benoît Colson, Patrice Marcotte, and Gilles Savard. 2007. An overview of bilevel optimization. *Annals of operations research* 153 (2007), 235–256.

[4] Laurent El Ghaoui, Fangda Gu, Bertrand Travacca, Armin Askari, and Alicia Tsai. 2021. Implicit deep learning. *SIAM Journal on Mathematics of Data Science* 3, 3 (2021), 930–958.

[5] Jianfei Gao and Bruno Ribeiro. 2022. On the equivalence between temporal and static equivariant graph representations. In *International Conference on Machine Learning*. PMLR, 7052–7076.

[6] Zhengyang Geng, Xin-Yu Zhang, Shaojie Bai, Yisen Wang, and Zhouchen Lin. 2021. On training implicit models. *Advances in Neural Information Processing Systems* 34 (2021), 24247–24260.

[7] Fangda Gu, Heng Chang, Wenwu Zhu, Somayeh Sojoudi, and Laurent El Ghaoui. 2020. Implicit graph neural networks. *Advances in Neural Information Processing Systems* 33 (2020), 11984–11995.

[8] William L. Hamilton, Rex Ying, and Jure Leskovec. 2018. Inductive Representation Learning on Large Graphs. arXiv:1706.02216 [cs.SI]

[9] Danfeng Hong, Lianru Gao, Jing Yao, Bing Zhang, Antonio Plaza, and Jocelyn Chanussot. 2020. Graph convolutional networks for hyperspectral image classification. *IEEE Transactions on Geoscience and Remote Sensing* 59, 7 (2020), 5966–5978.

[10] Quanqi Hu, Yongjian Zhong, and Tianbao Yang. 2022. Multi-block min-max bilevel optimization with applications in multi-task deep auc maximization. *arXiv preprint arXiv:2206.00260* (2022).

[11] Shima Khoshraftar and Aijun An. 2022. A survey on graph representation learning methods. *arXiv preprint arXiv:2204.01855* (2022).

[12] Thomas N Kipf and Max Welling. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* (2016).

[13] Fuxian Li, Jie Feng, Huan Yan, Guangyin Jin, Fan Yang, Funing Sun, Depeng Jin, and Yong Li. 2023. Dynamic graph convolutional recurrent network for traffic prediction: Benchmark and solution. *ACM Transactions on Knowledge Discovery from Data* 17, 1 (2023), 1–21.

[14] Mingjie Li, Yifei Wang, Yisen Wang, and Zhouchen Lin. 2022. Unbiased Stochastic Proximal Solver for Graph Neural Networks with Equilibrium States. In *The Eleventh International Conference on Learning Representations*.

[15] Qimai Li, Zhichao Han, and Xiao-Ming Wu. 2018. Deeper insights into graph convolutional networks for semi-supervised learning. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.

[16] Ruoyu Li, Sheng Wang, Feiyun Zhu, and Junzhou Huang. 2018. Adaptive Graph Convolutional Neural Networks. arXiv:1801.03226 [cs.LG]

[17] Jie Liao, Wei Zhou, Fengji Luo, Junhao Wen, Min Gao, Xiuhua Li, and Jun Zeng. 2022. SocialLGN: Light graph convolution network for social recommendation. *Information Sciences* 589 (2022), 595–607.

[18] Juncheng Liu, Kenji Kawaguchi, Bryan Hooi, Yiwei Wang, and Xiaokui Xiao. 2021. Eignn: Efficient infinite-depth graph neural networks. *Advances in Neural Information Processing Systems* 34 (2021), 18762–18773.

[19] Yao Ma, Ziyi Guo, Zhaocun Ren, Jiliang Tang, and Dawei Yin. 2020. Streaming graph neural networks. In *Proceedings of the 43rd international ACM SIGIR conference on research and development in information retrieval*. 719–728.

[20] George Panagopoulos, Giannis Nikolentzos, and Michalis Vazirgiannis. 2021. Transfer Graph Neural Networks for Pandemic Forecasting. In *Proceedings of the 35th AAAI Conference on Artificial Intelligence*.

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

[22] Junyoung Park, Jinhyun Choo, and Jinkyoo Park. 2021. Convergent Graph Solvers. In *International Conference on Learning Representations*.

[23] Junhui Park, Gaeun Sung, SeungHyun Lee, SeungHo Kang, and ChunKyun Park. 2022. ACGCN: graph convolutional networks for activity cliff prediction between matched molecular pairs. *Journal of Chemical Information and Modeling* 62, 10 (2022), 2341–2351.

[24] Barak A Pearlmutter. 1994. Fast exact multiplication by the Hessian. *Neural computation* 6, 1 (1994), 147–160.

[25] Zi-Hao Qiu, Quanqi Hu, Yongjian Zhong, Lijun Zhang, and Tianbao Yang. 2022. Large-scale stochastic optimization of ndcg surrogates for deep learning with provable convergence. *arXiv preprint arXiv:2202.12183* (2022).

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

[27] Halsey Lawrence Royden and Patrick Fitzpatrick. 1968. *Real analysis*. Vol. 2. Macmillan New York.

[28] T Konstantin Rusch, Michael M Bronstein, and Siddhartha Mishra. 2023. A survey on oversmoothing in graph neural networks. *arXiv preprint arXiv:2303.10993* (2023).

[29] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. 2020. 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*. 519–527.

[30] Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. 2018. Structured sequence modeling with graph convolutional recurrent networks. In *Neural Information Processing: 25th International Conference, ICONIP 2018, Siem Reap, Cambodia, December 13-16, 2018, Proceedings, Part I 25*. Springer, 362–373.

[31] Mengzhu Sun, Xi Zhang, Jiaqi Zheng, and Guixiang Ma. 2022. Ddgcn: Dual dynamic graph convolutional networks for rumor detection on social media. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 36. 4611–4619.

[32] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. arXiv:1710.10903 [stat.ML]

[33] Zehong Wang, Qi Li, and Donghua Yu. 2022. TPGNN: Learning High-order Information in Dynamic Graphs via Temporal Propagation. *arXiv preprint arXiv:2210.01171* (2022).

[34] Dongkuan Xu, Wei Cheng, Dongsheng Luo, Yameng Gu, Xinyu Liu, Jingchao Ni, Bo Zong, Haifeng Chen, and Xiang Zhang. 2019. Adaptive Neural Network for Node Classification in Dynamic Networks. *2019 IEEE International Conference on Data Mining (ICDM)* (2019), 1402–1407.

[35] Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962* (2020).

[36] Menglin Yang, Ziqiao Meng, and Irwin King. 2020. Featurenorm: L2 feature normalization for dynamic graph embedding. In *2020 IEEE International Conference on Data Mining (ICDM)*. IEEE, 731–740.

[37] Ling Zhao, Yujiao Song, Chao Zhang, Yu Liu, Pu Wang, Tao Lin, Min Deng, and Haifeng Li. 2019. T-gcn: A temporal graph convolutional network for traffic prediction. *IEEE transactions on intelligent transportation systems* 21, 9 (2019), 3848–3858.

Layers	GCN-GRU	T-GCN	IDGNN
8	0.6217	0.9934	1.1113
16	0.5204	0.8719	1.0982
32	0.0077	0.7176	1.0019

Table 7: Smoothness of embeddings. (The larger the better)

A EXPERIMENT

A.1 Experiment Setup

We follow the procedure from [5] and utilize the provided code as the code base to compare all the baselines and our method. Specifically, we first split the dataset into three portions with ratios 70%-10%-20% for training, validation, and testing, respectively. Splitting is based on nodes for transductive tasks and time for inductive tasks. We then normalize node attributes and edge attributes with the 0-1 normalization method. We train on the training portion, find the best hyperparameters using the validation set, and report the performance on the test set. We also use ROCAUC score to evaluate classification tasks and mean average percentage error (MAPE) for regression tasks.

A.2 Hyperparameter

For detailed baselines' architecture, please refer to [5]. Notice that, for all the methods and all task, we fixed the embedding size as 16, and we searched the learning rates from 0.1, 0.01, and 0.001. For Brain10, we observed that our method converged slowly, then we let it run for 1000 epochs. For DBLP5 and Reddit4, we let all the methods run 500 epochs for 5 times for each learning rate and report the performance on the test set where the performance of the validation set is the best. For regression datasets, we run 100 epochs for England-COVID and 10 epochs for PeMS04/PeMS08. The hyperparameter our model $\eta_1 \in \{0.5, 0.7, 0.9, 1\}, \eta_2 \in \{0.001, 0.01, 0.1\}$.

PROOFS

Lemma A.1. *If $\|.\|_{op}$ is the matrix operator norm on $\mathbb{R}^{n\times n}$ and $\lambda_{PF}(A)$ outputs the largest absolute eigenvalue of A, then, for any matrix $A \in \mathbb{R}^{n\times n}$,*

$$\lambda_{PF}(A) \leq \|A\|_{op}$$

Proof. Let λ be the eigenvalue of A, and let $x \neq 0$ be the corresponding eigenvector. From $Ax = \lambda x$, we have

$$AX = \lambda X$$

where each column in X is x. Further, we have

$$|\lambda|\|X\|_{op} = \|\lambda X\|_{op} = \|AX\|_{op} \leq \|A\|_{op}\|X\|_{op}$$

Since $\|X\| > 0$, taking the maximal λ gives the result. □

Lemma A.2. *The equilibrium equation $z = \sigma(Mz+b)$ has a unique fixed point solution if $|||M|||_{op} < 1$, where $\|.\|_{op}$ is the operator norm, and $\sigma(\cdot)$ is an element-wise non-expansive function.*

Proof. Based on Lemma A.1 and Theorem 4.1 in [7], this lemma is immediately followed. □

Proof of Theorem 3.1

Proof. By Lemma. A.2, the well-posedness requirement for Formula. (3) is $\|\mathcal{M}\|_{op}| \leq 1$. Since Formula. (3) and (2) are equivalent, the well-posedness requirement for Formula. (2) is also $|||\mathcal{M}|||_{op} < 1$.

Let $M := \begin{bmatrix} 0 & M_1 \\ \hat{M} & 0 \end{bmatrix}$ where $\hat{M} := \begin{bmatrix} M_2 & \dots & \\ 0 & \ddots & 0 \\ 0 & \dots & M_t \end{bmatrix}$. Let $\tilde{M} := \begin{bmatrix} \hat{M} & 0 \\ 0 & M_1 \end{bmatrix}$. Then

$$\begin{aligned} |||M|||_{op} &= |||\tilde{M}| \begin{bmatrix} 0 & I_m \\ I_n & 0 \end{bmatrix}\|_{op} \leq |||\tilde{M}|||_{op} \cdot \| \begin{bmatrix} 0 & I_m \\ I_n & 0 \end{bmatrix}\|_{op} \\ &= |||\tilde{M}|||_{op} = \max\{|||M_1|||_{op}, ..., |||M_t|||_{op}\} \end{aligned}$$

This means if all subsystems satisfy the largest eigenvalue constraint, then the coupled equilibrium equation has a fixed point solution by Lemma A.2. □

B EMBEDDING VISUALIZATION

In this section, we explore an interesting aspect of our method that can provide empirical insights into its ability to mitigate oversmoothing. We conduct experiments on a synthetic dataset that bears resemblance to toy datasets.

The dataset comprises 10 snapshots, with each snapshot representing a clique of 10 nodes. Each node is associated with 10 attributes. The nodes fall into two distinct classes, but we deliberately conceal the label information in the attributes of the first snapshot. Specifically, the first two dimensions of the attributes represent the one-hot encoding of the labels, while the remaining dimensions are set to zero. Additionally, we assign unique IDs to the nodes in sequential order. Nodes with IDs ranging from 0 to 4 belong to class 0, while those with IDs ranging from 5 to 9 belong to class 1. To assess the effectiveness of our method, we visually compare the embedding results with those obtained from TGCN.

Upon examining the visualizations, we observe that our model's embeddings exhibit gradual changes, whereas TGCN's embeddings remain consistent for nodes belonging to the same class. From a node-centric perspective, TGCN's embeddings seem reasonable. Nodes of the same class possess identical features and exhibit the same topological structure. Therefore, it is logical for them to share a common embedding. However, our embeddings tend to differentiate each individual node. We believe that this characteristic plays a role in mitigating the oversmoothing problem within our model.

Furthermore, we conducted an additional experiment to quantitatively evaluate our model's ability to tackle over-smoothing. This experiment is conducted on the binary toy dataset: the toy data we constructed consists of a dynamic graph with a maximum of 64 snapshots (adapt to layers), with each snapshot being a clique of 10 nodes. Each node has 10 associated attributes. The task is binary classification where each node's class information is hidden in its first time-stamp's attributes. Attributes of other time stamps are randomly sampled from the normal distribution. All methods are trained with a maximum of 2000 epochs and a learning rate of 0.001. Finally, we evaluate their smoothness using the Mean Average Distance (MAD). Results are summarized in Table 7.

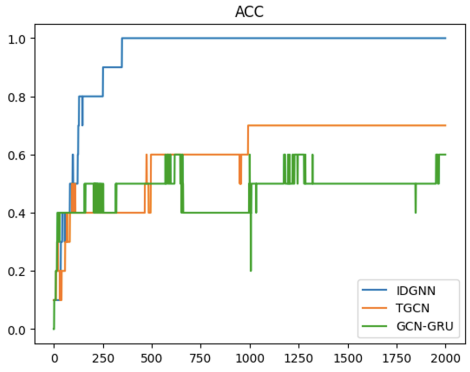
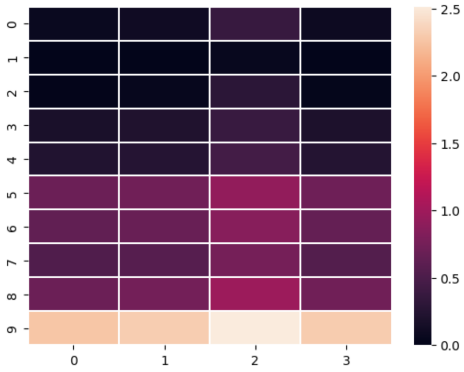
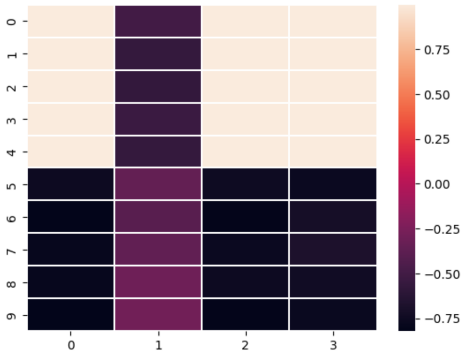


Figure 4: Additional result.



(a) Our model



(b) TGCN

Figure 3: The embedding visualization of our method and TGCN

C HESSIAN VECTOR PRODUCT

To compute the product of Hassian and a vector: Hv, and $H = \frac{\partial^2 f}{\partial x^2}$. We compute the product by $Hv = \frac{\partial (\frac{\partial f}{\partial x})^T v}{\partial x}$. In this way, we are not explicitly computing the Hessian. [6]

D ADDITIONAL RESULT ON TOY DATA

This synthetic experiment shifts label information from time stamp 1 to time stamp 5. This adjustment ensures uniform difficulty in utilizing label information across all models. Implementing this change postpones our method towards achieving 100% accuracy by approximately 50 epochs. However, even with this modification, the baselines still struggle to fit the data.

D.1 Complexity

To contrast with the emprical runtime, here we present the theoretical time complexities for our approach and the baselines. For an arbitrary temporal graph, we denote the total number of snapshots as T, the total number of nodes as n, the number of edges of time t as E_t, and E_{agg} as the number of aggregated edges, which satisfies $E_{agg} \ll \sum_t E_t$ or $E_{agg} \approx \sum_t E_t$. Some basic complexities: GRU, self-attention, and LSTM have complexity $O(T^2)$ if the input sequence has length T; GCN and SpectralGCN have complexity $O(nd^2 + Ed)$; GAT has complexity $O(nd + Ed^2)$. The complexities of all models are summarized in Table 8. Based on the complexity results, IDGNN is faster than EvolveGCN-O, EvolveGCN-H, GCN-GRU, GCRN-M2, and DCRNN due to the absence of RNN in our model.

[6] https://jax.readthedocs.io/en/latest/notebooks/autodiff_cookbook.html

EvolveGCN-O	$O(Td^2 + Tnd^2 + \sum_t E_t d)$
EvolveGCN-H	$O(Td^2 + Tnd^2 + \sum_t E_t d)$
GCN-GRU	$O(Td^2 + Tnd^2 + \sum_t E_t d)$
DySAT	$O(Td^2 + Tnd + \sum_t E_t d^2)$
GCRN-M2	$O(Td^2 + Tnd^2 + \sum_t E_t d)$
DCRNN	$O(Td^2 + Tnd^2 + \sum_t E_t d)$
TGAT	$O(Tnd + \sum_t E_t d^2)$
TGN	$O(E_{agg} d + Tnd^2 + \sum_t E_t d^2)$
GRU-GCN	$O(E_{agg} d + Tnd^2 + TE_{agg} d^2)$
IDGNN	$O(Tnd^2 + \sum_t E_t d)$

Table 8: Summary of complexities of all models