Scaling Up Dynamic Graph Representation Learning via Spiking Neural Networks

Jintang Li[1], Zhouxin Yu[1], Zulun Zhu[2], Liang Chen[1],
Zibin Zheng[1], Qi Yu[2], Sheng Tian[3], Ruofan Wu[3], Changhua Meng[3]
[1] Sun Yat-sen University
[2] Rochester Institute of Technology
[3] Ant Group
{lijt55, yuzhx6}@mail2.sysu.edu, {chenliang6, zhzibin}@mail2.sysu.edu,
zz2023@g.rit.edu, qi.yu@rit.edu, {tiansheng.tsuofan.wrf,changhua.mch}@antgroup.comr

Abstract

Recent years have seen a surge in research on dynamic graph representation learning, which aims to model temporal graphs that are dynamic and evolving constantly over time. However, current work typically models graph dynamics with recurrent neural networks (RNNs), making them suffer seriously from computation and memory overheads on large temporal graphs. So far, scalability of dynamic graph representation learning on large temporal graphs remains one of the major challenges. In this paper, we present a scalable framework, namely SpikeNet, to efficiently capture the temporal and structural patterns of temporal graphs. We explore a new direction in that we can capture the evolving dynamics of temporal graphs with spiking neural networks (SNNs) instead of RNNs. As a low-power alternative to RNNs, SNNs explicitly model graph dynamics as spike trains of neuron populations and enable spike-based propagation in an efficient way. Experiments on three large real-world temporal graph datasets demonstrate that SpikeNet outperforms strong baselines on the temporal node classification task with lower computational costs. Particularly, SpikeNet generalizes to a large temporal graph (2M nodes and 13M edges) with significantly fewer parameters and computation overheads. Our code is publicly available at https://github.com/EdisonLeeeee/SpikeNet

A graph is comprised of nodes and edges connected together to model structures and relationships of objects in various scientific and commercial fields (Hu et al. 2020; Chen et al. 2020). It is highly expressive and capable of representing complex structures that are difficult to model, with prominent examples including social networks (Liu et al. 2020), molecules graphs (Zhou et al. 2020), and transaction networks (Chen et al. 2020). In practice, graphs are often dynamic, meaning that nodes, edges, and attributes may evolve constantly over time. This type of graphs are typically referred to as *temporal graphs*, in contrast to *static graphs* where nodes and edges remain fixed over time (Kazemi et al. 2020). Figure 1 is an illustration of static and temporal graphs. The temporal graph is usually represented as a sequence of graph snapshots, while the static graph can be seen as a single (static) observation of the temporal graph.

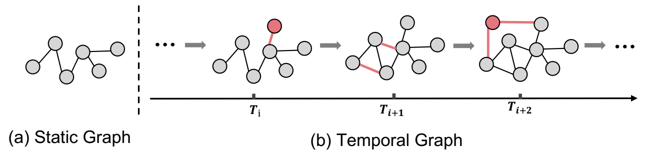


Figure 1: An illustrative example of (a) static graph and (b) temporal graph. A temporal graph is typically represented as a series of graph snapshots, with network structure constantly evolving over time.

Over the past few years, graph neural networks (GNNs) have emerged as highly successful tools for learning graph-structured data. The advances in GNNs have led to new state-of-the-art results in numerous graph-based learning tasks (Hamilton, Ying, and Leskovec 2017; Chen et al. 2021). However, GNNs have been primarily developed for dealing with static graphs while ignoring that the graph itself is dynamically evolving over time. So far, several efforts have been made to generalize current GNNs to temporal graphs by additionally considering the time dimension (Kazemi et al. 2020). The most established solution for modeling temporal graphs is by extending sequence models to graph data. In this vein, recurrent neural networks (RNNs) (Cho et al. 2014) are prominent methods that provide a natural choice to capture the temporal information for these evolving graphs (Xu et al. 2019; Pareja et al. 2020; Shi et al. 2021; Kumar, Zhang, and Leskovec 2019). In addition, self-attention (Xu et al. 2020), random walk (Sajjad, Docherty, and Tyshetskiy 2019; Wang et al. 2021; Makarov et al. 2021), and temporal point process (Lu et al. 2019; Zuo et al. 2018; Zhou et al. 2018) are some other approaches to learn structural and temporal patterns simultaneously.

While current approaches have achieved promising results, there are some fundamental challenges when switching from static to temporal graphs. (i) *Scalability*: current research is mostly dominated by RNN-based methods, which require a large number of memory cells to store temporal contextual information (Sak, Senior, and Beaufays 2014) and large amounts of training data to outperform even static methods. As a result, they suffer from high overheads and scale poorly as the number of time steps increases, especially for large graphs. (ii) *Inductive capability*: an inductive approach can generate predictions quickly for unseen nodes or links in an evolving graph (e.g., new uses and social behaviors in a social network), which is essential for temporal graphs as new arriving nodes and links need to be processed timely (Wang et al. 2021). However, the inductive learning capability on temporal graphs is less studied as opposed to that on static graphs. Overall, the above challenges limit the applications of current research on large temporal graphs and remain to be addressed.

In light of these challenges, we propose SpikeNet, an inductive framework for modeling temporal graphs with high scalability. SpikeNet incorporates the spiking neural networks (SNNs) (Gerstner and Kistler 2002) into the design of GNNs, with the aim to make better use of the spatio-temporal information while addressing the scalability issue on large temporal graphs. The key idea behind SpikeNet is the use of SNNs, a class of brain-inspired computing models that communicate with discrete spikes in an event-based manner. At each time step, SpikeNet aggregates and updates the signals (i.e., spikes) from a sampled neighborhood via an integrate-and-fire event. In this way, SpikeNet benefits from the spatio-temporal characteristics of SNNs with low computation and memory overheads. In summary, our contributions are as follows:

- SpikeNet, a spike-based inductive learning framework for efficiently capturing the structural patterns and evolving dynamics in temporal graphs.
- A temporal architectural design based on SNNs, combining analog computation with spike-based propagation for more efficient model training and inference. To our best knowledge, this is the first work to build a general SNNs-based framework for modeling temporal graphs.
- An adaptive threshold update strategy for SNNs training to fit the neuron dynamics, which increases the flexibility and expressiveness of the resulting model.
- Extensive experiments that demonstrate the outstanding performance over advanced methods in real-world datasets on the temporal node classification tasks.

Related Work

In line with the focus of our work, we provide a brief overview of the background and related work on dynamic graph representation learning and spiking neural networks.

Dynamic Graph Representation Learning

Over the past few years, learning dynamic representations for temporal graphs has attracted considerable research effort (Kazemi et al. 2020). Typically, methods developed for temporal graphs are often extensions of those for static ones, which additionally consider the temporal dimension and update schemes (Pareja et al. 2020; Xu et al. 2020).

Owning to the success of RNNs (Cho et al. 2014), one way to effectively model the temporal graph data is by employing the RNN (typically LSTM (Hochreiter and Schmidhuber 1997)) architectures. Methods whereby RNNs generalize traditional GNNs to various temporal tasks in different manners (Kumar, Zhang, and Leskovec 2019; Pareja et al. 2020; Shi et al. 2021; Rossi et al. 2020; Ma et al. 2020; Xu et al. 2019). For example, TGN (Rossi et al. 2020) and JODIE (Kumar, Zhang, and Leskovec 2019) update the memory and hidden state of the nodes by utilizing the LSTM and RNN units, and then the interaction events (node-wise or edge-wise) can be reasonably formulated. EvolveGCN (Pareja et al. 2020) proposes to use RNNs to regulate the GNN model parameters at every time step. Although the RNN-based methodology can efficiently save the history of interaction events related to the time dimension and capture graph dynamics, high dimensional vectors and numerous memory units also make the models suffer from massive computational costs and memory footprints.

Another line of research to explore the temporal graph dynamics is modeling sequential asynchronous discrete events occurring in continuous time as stochastic processes (Trivedi et al. 2019; Zuo et al. 2018; Lu et al. 2019; Zhou et al. 2018). These methods try to capture evolution patterns over historical structures by a temporal point process, in addition to attention mechanisms (Zuo et al. 2018; Lu et al. 2019; Trivedi et al. 2019) and temporal smoothness (Zhou et al. 2018) enforced on the representation of continuous snapshots. However, the temporal point process may not allow the sharp change with regard to the node insertion and deletion (evolutionary), and the estimation models are typically quadratic in the number of interaction events, which is too computationally expensive for large graphs.

Spiking Neural Networks

SNNs are a class of brain-inspired and energy-efficient networks, which have attached great importance due to the distinctive properties of low power consumption and biological plausibility (Bu et al. 2022). Different from artificial neural networks (ANNs) that make use of float values, SNNs deliver binary and asynchronous information through spikes only when the membrane potential reaches the threshold (Kim et al. 2020; Bu et al. 2022). In literature, SNNs have been reported to offer huge energy-efficiency advantage over ANNs while achieving comparative results in a wide range of vision tasks, such as image classification (Fang et al. 2020), object detection (Kim et al. 2020), and semantic segmentation (Kim, Chough, and Panda 2021)

Recently, efforts have been made towards applying SNN to graph domain (Dold and Garrido 2021; Chian et al. 2021; Zhu et al. 2022; Xu et al. 2021b). Similar to that in vision research, they transform initial node features into a series spike trains via Poisson rate coding, followed by a graph convolution layer with SNN neurons. SNNs are inherently designed for temporal data, however, prior work mainly focuses on static graphs while ignoring the dynamic nature of SNNs. In this regard, the feasibility and advantages of SNNs for representing temporal graphs remain largely unexplored. This paper presents the first research effort to exploit SNNs on temporal graphs, which opens up possibilities for exploiting graph dynamics while avoiding high overheads.

Preliminary

In this section, we begin by giving the problem definition of this work, followed by the introduction of the leaky integrate-and-fire model.

Problem Definition

Notations In this work, we consider the discrete-time scenario of dynamic graph representation learning. A temporal graph, in general, is defined as a sequence of graph snapshots at T different time steps, denoted by $\mathcal{G} = \{\mathcal{G}^1, \mathcal{G}^2, \ldots, \mathcal{G}^T\}$[1]. Each graph snapshot at time t is defined as $\mathcal{G}^t = (\mathcal{V}^t, \mathcal{E}^t)$ where $\mathcal{V}^t = \{v_1, v_2, \ldots, v_N\}$ is a set of N nodes and $\mathcal{E}^t \subseteq \mathcal{V}^t \times \mathcal{V}^t$ is a set of edges. Let $\mathcal{V}$ be the set of all nodes that appear in $\mathcal{G}$. Without loss of generality, we assume that the graph snapshot at any time is built on a common node set $\mathcal{V}$, in which the nonexistent node is treated as a dangling one with zero degree. In each time step, nodes can be paired with evolving features $X^t = \{x_v \mid \forall v \in \mathcal{V}\} \in \mathbb{R}^{N \times d}$, where d is the feature dimensionality and is commonly stable over time. The entire node features with time dimension can be denoted as $\mathcal{X} = \{X^1, X^2, \ldots, X^T\} \in \mathbb{R}^{T \times N \times d}$.

Our goal is to learn node embedding Z for all nodes that appeared in a temporal graph $\mathcal{G}$. The embedding should include nodes' (edges') evolution dynamics over time, which can be further used for downstream tasks like temporal node classification.

Leaky Integrate-and-Fire Model

In literature, the integrate-and-fire (IF) (Salinas and Sejnowski 2002) model is a mainstream computational model for neuron simulation. The IF model represents the membrane potential as a charge stored on a capacitor and abstracts the neuron dynamics of SNNs as three temporal events: (i) *Integrate*. The neuron integrates current by means of the capacitor over time, which leads to a charge accumulation; (ii) *Fire*. When the membrane potential has reached or exceeded a given threshold V_{th}, it fires (i.e., emits a spike). (iii) *Reset*. Once fired, the membrane potential will be reset back to a constant value $V_{\text{reset}} < V_{\text{th}}$ like a biological neuron (Izhikevich 2004).

However, neuron membranes are not perfect capacitors, rather they slowly leak current over time, pulling the membrane voltage back to its resting potential (Hunsberger, Eric 2018). In this regard, the LIF model additionally introduces a leaky term and describes the neuronal activity as an integration of received spike voltages as well as weak dissipation to the environment:

$$\tau_m \frac{dV}{dt} = -(V - V_{\text{reset}}) + \Delta V_m, \tag{1}$$

where ΔV_m is the pre-synaptic input to the membrane voltage; τ_m is a membrane-related hyperparameter to control how fast the membrane potential decays, which leads to the membrane potential charges and discharges exponentially in response to the inputs. To better describe the neuron behaviors and guarantee computational availability, the differential equation in Eq. (1) can also be converted to an iterative expression, as in (Fang et al. 2020; Zhu et al. 2022):

$$V^t = V^{t-1} + \frac{1}{\tau_m}\left(-(V^{t-1} - V_{\text{reset}}) + I^t\right), \tag{2}$$

[1]In this paper, we use the superscript t to denote the time index.

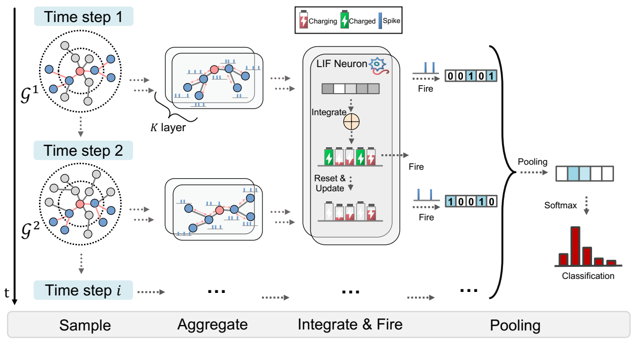


Figure 2: An overview of the SpikeNet framework.

where I^t represents the pre-synaptic input from preceding neurons at time step t.

Present Work: SpikeNet

In this section, we propose SpikeNet, a scalable framework that generalizes spiking neural networks to temporal graphs. SpikeNet is able to capture the evolving dynamics in temporal graphs, in which a variety of (unseen) nodes and edges are appeared/disappeared over time. An overview of SpikeNet is shown in Figure 2. For each arriving timestamp, SpikeNet samples a subset of nodes and learns to aggregate spike signals from a node's local neighborhood. Then, the subsequent LIF model takes the aggregated signals as inputs to capture the temporal dynamics through an integrate-and-fire mechanism. Finally, the node embedding is obtained by taking historical spikes in each time step with a spike pooling operation, which can be used for downstream learning in a supervised manner.

LIF as Temporal Architecture

We explore a new direction in that we can use SNNs (i.e., LIF) rather than RNNs as the temporal architecture to capture the dynamic patterns in a graph sequence. Recall that the LIF's neuron behaviors are characterized by a series of events: *integrate*, *fire*, and *reset*. For an arriving timestamp, each neuron in the LIF model updates the membrane potential based on its memorized state and current inputs, and then fires a spike when the membrane potential reaches a threshold V_{th}. If a spike is fired, the membrane potential is reset to a specific level V_{reset}, and the process starts again for the next round. Neurons at each layer undergo this process based on the input signals received from the preceding layer. Accordingly, a spike train is produced by each neuron for the subsequent layer.

However, prior work (Fang et al. 2020; Zhu et al. 2022) typically adopts a fixed firing threshold V_{th} for each LIF neuron across different layers, potentially limiting its flexibility and expressiveness. To address this issue, we propose an adaptive update strategy for neuron threshold, which calculates changes in neuron dynamics based on the previous threshold and incoming spikes:

$$V_{\text{th}}^t = \tau_{\text{th}} V_{\text{th}}^{t-1} + \gamma O^t, \tag{3}$$

where τ_{th} and γ are the decay factors to tune the threshold during training. O^t is the spike train that takes a binary value (0 or 1) representing whether a neuron is spiked at time t. In this way, the firing threshold can be adaptively adjusted according to the neuron dynamics. We observe that the adaptive strategy also benefits the final performance in practice. By incorporating the threshold-triggered firing mechanism, membrane reset, and adaptive threshold update rules, we propose to generalize the computation scheme of the LIF model as follows:

$$\begin{aligned} \textbf{Integrate:} \quad & V^t = f(V^{t-1}, I^t), & (4) \\ \textbf{Fire:} \quad & O^t = \Theta(V^t - V_{\text{th}}^{t-1}), & (5) \\ \textbf{Reset:} \quad & V^t = O^t V_{\text{reset}} + (1 - O^t)V^t, & (6) \\ \textbf{Update:} \quad & V_{\text{th}}^t = \tau_{\text{th}} V_{\text{th}}^{t-1} + \gamma O^t, & (7) \end{aligned}$$

where f denotes the integration behaviors of the LIF model when the neuron receives synaptic inputs from previous neurons, as in Eq. (2). The decision to fire and generate a spike in the neuron output is carried out according to the Heaviside step function (Fang et al. 2020), which is defined by $\Theta(x) = 1$ if $x \geq 0$ and 0 otherwise. For convenience, we simplify the above procedure as $O^t = \delta(I^t)$ with $\delta(\cdot)$ the LIF model when receiving an input I^t. For each time step t, the input I^t is the aggregated neighborhood information while the output O^t is a spike train for intermediate node representations. In what follows, we will introduce how SpikeNet aggregates neighborhood messages as I^t at each layer and time step t.

Temporal Neighborhood Sampling

Neighborhood sampling is very important to learn node representations for large-scale graphs (Hamilton, Ying, and Leskovec 2017). Typically, it follows the design philosophy of directly sampling a set of nodes from multiple hops instead of using full neighborhood sets to avoid the over-expansion issue. However, it was initially designed for static graphs without considering the structural evolution of temporal graphs.

In this work, we propose temporal neighborhood sampling on graph snapshots, which captures the graph dynamics from both macroscopic and microscopic perspectives. Given a node $v \in \mathcal{V}$, we sample a set of nodes by expanding the v's neighborhood in the following way:

$$\mathcal{S}^t(v) = \underbrace{\mathrm{SAMPLE}(v, \mathcal{G}^t)}_{\text{Macro-dynamics}} \cup \underbrace{\mathrm{SAMPLE}(v, \Delta\mathcal{G}^t)}_{\text{Micro-dynamics}}, \qquad (8)$$

where SAMPLE is a graph sampler that produces the required neighborhood sets by uniform sampling (Hamilton, Ying, and Leskovec 2017) or random walk (Perozzi, Al-Rfou, and Skiena 2014), to alleviate receptive field expansion and improve the computation efficiency. We denote $\mathcal{S}^t(v)$ a random sample of the node v's neighbors at time step t. $\Delta\mathcal{G}^t = \mathcal{G}^t - \mathcal{G}^{t-1}$ represents a graph snapshot with all edges established at time t; Particularly $\Delta\mathcal{G}^1 = \mathcal{G}^1$. The sampling step incorporates the *macro-dynamics* and *micro-dynamics* at each time step to learn node representations.

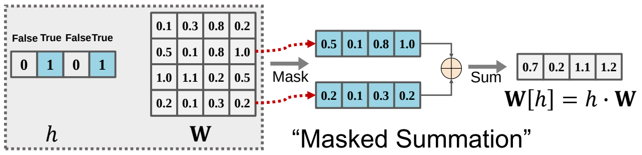


Figure 3: An example of "Masked Summation" operation $\mathbf{W}[h]$, which acts like matrix multiplication $h \cdot \mathbf{W}$ but enables more efficient computation.

Intuitively, micro-dynamics capture the fine-grained structural and temporal properties for node representation, while macro-dynamics explore the inherent evolution pattern of the graph from a global perspective.

Spike-based Neighborhood Aggregation

In the sampling step, we recursively sample the neighbors of root nodes up to a depth to perform message aggregation in a graph. At layer k, each node v is related to a sequence of local neighborhood $\{\mathcal{S}^1, \mathcal{S}^2, \ldots, \mathcal{S}^T\}$ with T time spans, to form a group of sample nodes along the time dimension. For each sampled set, SpikeNet computes the root node's hidden representation by recursively aggregating hidden node representations from bottom to top:

$$\begin{aligned} h_{\mathcal{S}}^{t,(k)} &= \{h_u^{t,(k-1)}\mathbf{W}, \forall u \in \mathcal{S}^t\}, \\ h_v^{t,(k)} &= \delta\left(\mathrm{AGG}\left(\{h_v^{t,(k-1)}\mathbf{W}\} \cup h_{\mathcal{S}}^{t,(k)}\right)\right), \end{aligned} \qquad (9)$$

where $h_v^{t,(k)}$ is the hidden representation of node v in k-th layer at time step t and particularly $h_v^{t,(0)} = x_v^t$; $\mathbf{W}$ is the learnable parameter. AGG is a differentiable aggregation function, which aggregates features of local neighborhoods and passes them to the target node v. The aggregation function should be invariant to the permutations of node orderings such as a mean, sum, or max function. $\delta(\cdot)$ is the LIF model which takes the aggregated information as input and outputs a spike train according to the neuron dynamics.

In principle, the node representations are mapped to spike trains, offering a natural transition from the graph domain to SNNs. More specifically, h_v is the pre-synaptic input denoted by a set of boolean numbers (spike or no spike), thus the matrix multiplication could be alternatively replaced by simple masking (indexing) operation combined with accumulation (since spikes are zero or one, a spike times any value x is either zero or x, avoiding the need for explicit matrix multiplication):

$$\begin{aligned} h_{\mathcal{S}}^{t,(k)} &= \{\mathbf{W}[h_u^{t,(k-1)}], \forall u \in \mathcal{S}^t\}, \\ h_v^{t,(k)} &= \delta\left(\mathrm{AGG}\left(\{\mathbf{W}[h_v^{t,(k-1)}]\} \cup h_{\mathcal{S}}^{t,(k)}\right)\right), \end{aligned} \qquad (10)$$

where $\mathbf{W}[\cdot]$ denotes the *Masked Summation* operation on $\mathbf{W}$. As shown in Figure 3, the row in a matrix $\mathbf{W}_{i,:}$ is masked (indexed) if the corresponding element $h_i = 1$(True), and then the indexed row vectors are accumulated to calculate the summations. The operation can avoid expensive matrix multiplication computations entirely, and theoretically and also in practice, speed up the model.

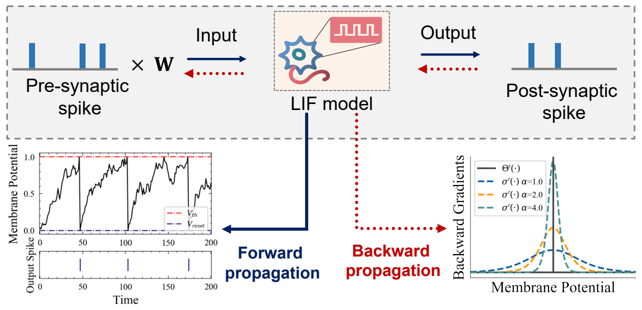


Figure 4: The illustration of forward propagation (blue arrow) and backward propagation (red dotted arrow) of SpikeNet. **Forward propagation**: the neuron outputs a spike via an integrate-and-fire mechanism, which brings the non-differentiability of the membrane potential. **Backward propagation**: a smooth Sigmoid function with different smooth factor α is adopted to approximate the backward gradients.

Spike Pooling

The pooling layer is widely used to compress a set of nodes into a compact representation (Wu et al. 2022). Inspired by the graph pooling in GNNs, we propose spike pooling as an independent operation to produce coarsened node representations over historical spikes. In this way, the final node representation is obtained by taking historical spikes at the last layer K with a pooling operation:

$$z_v = \text{Pool}\left(\{h_v^{t,(K)}, \dots, h_v^{T,(K)}\}\right), \forall v \in \mathcal{V} \tag{11}$$

where K is the number of layers and T is the number of time steps. There are several different ways to implement the Pool operator, such as Sum and Avg which take the sum or average sum over all historical spikes. However, either Sum or Avg typically treats historical spikes equally in a time window, which might hinder the model's fitting capability in temporal tasks. In this regard, we adopt a learnable linear transformation over historical spikes:

$$z_v = \text{Linear}\left(\{h_v^{t,(K)} \| \dots \| h_v^{T,(K)}\}\right), \forall v \in \mathcal{V} \tag{12}$$

where $\|$ is a concatenating operator. The linear transformation could be implemented with a single-layer feed-forward neural network, where masked summation is also available to offer better efficiency during training and inference.

Surrogate Gradient based Learning

Given the node embeddings $z_v, \forall v \in \mathcal{V}$, we apply a softmax activation to transform node embeddings into prediction results for downstream tasks, followed by a task-specific objective (e.g., cross-entropy loss) to learn useful, predictive representations.

Note that it is difficult to train SpikeNet directly via standard gradient descent due to the non-differentiability of discrete spikes and the hard threshold function. Inspired by the surrogate learning technique (Neftci, Mostafa, and Zenke 2019; Fang et al. 2020), we circumvent the non-differentiable backpropagation problem by defining a surrogate function for LIF neurons when calculating backward gradients. Here we use a smooth Sigmoid function (Zenke and Vogels 2021) $\sigma(\alpha x) = 1/(1+\exp(-\alpha x))$ to calculate surrogate gradients during backward propagation, whose derivative is formally defined as:

$$\sigma(\alpha x)' = \alpha \cdot \sigma(\alpha x) \cdot (1 - \sigma(\alpha x)), \tag{13}$$

where α is a constant factor that controls the smoothness. In general, a larger α leads to a better approximation of the hard threshold function $\Theta(\cdot)$, but causes the vanishing and exploding gradient problem, which in turn makes the training very difficult to converge. Therefore, choosing a suitable value of the smooth factor α is critical for our algorithm to achieve comparable performance.

Figure 4 illustrates the surrogate learning procedure of SpikeNet. During the forward propagation, the membrane potential increases with pre-synaptic inputs, which are obtained from pre-synaptic spikes multiplied by the corresponding weight W. When the membrane potential reaches the threshold V_{th}, the neuron generates the post-synaptic spike, and then the membrane potential discharges to a resting potential V_{reset}. During the backward propagation, a smooth Sigmoid function with different α is adopted to implement the backward gradient to take advantage of the standard backpropagation-based optimization procedures.

Time Complexity

Here we briefly discuss the time complexity of our proposed SpikeNet. For simplicity, we assume that the node feature and hidden representations are d-dimensional, and the neighborhood size is fixed as $|\mathcal{S}|$ and shared by each hop. As the LIF model and spike pooling are both simple and computationally efficient, the main bottleneck is the aggregation step involved in all T graph snapshots. Particularly, for a K layer SpikeNet, the time complexity is related to the sampled graph size and the dimension of features/hidden representations, about $\mathcal{O}(T|\mathcal{V}||\mathcal{S}|^K d^2)$. Since $|\mathcal{S}|$, K, and d are often small, the computation overhead is acceptable even for large graphs. In addition, the time complexity could be further reduced by implementing the masked summation operation in specialized chips.

	DBLP	Tmall	Patent
#Nodes	28,085	577,314	2,738,012
#Edges	236,894	4,807,545	13,960,811
#Classes	10	5	6
#Time steps	27	186	25

Table 1: Dataset Statistics.

Experiments

In this section, we conduct experiments on three large real-world graph datasets: DBLP, Tmall (Lu et al. 2019), and Patent (Hall, Jaffe, and Trajtenberg 2001). The datasets statistics are listed in Table 1. Limited by space, we present

Dataset	Metrics	Tr.ratio	DeepWalk	Node2Vec	HTNE	M^2DNE	DyTriad	MPNN	JODIE	EvolveGCN	TGAT	SpikeNet
DBLP	Macro-F1	40%	67.08	66.07	67.68	69.02	60.45	64.19±0.4	66.73±1.0	67.22±0.3	**71.18**±0.4	70.88±0.4
		60%	67.17	66.81	68.24	69.48	64.77	63.91±0.3	67.32±1.1	69.78±0.8	71.74±0.5	**71.98**±0.3
		80%	67.12	66.93	68.36	69.75	66.42	65.05±0.5	67.53±1.3	71.20±0.7	72.15±0.3	**74.65**±0.5
	Micro-F1	40%	66.53	66.80	68.53	69.23	65.13	65.72±0.4	68.44±0.6	69.12±0.8	71.10±0.2	**71.98**±0.5
		60%	66.89	67.37	68.57	69.47	66.80	66.79±0.6	68.51±0.8	70.43±0.6	71.85±0.4	**72.35**±0.8
		80%	66.38	67.31	68.79	69.71	66.95	67.74±0.3	68.80±0.9	71.32±0.5	73.12±0.3	**74.86**±0.5
Tmall	Macro-F1	40%	49.09	54.37	54.81	57.75	44.98	47.71±0.8	52.62±0.8	53.02±0.7	56.90±0.6	**58.84**±0.4
		60%	49.29	54.55	54.89	57.99	48.97	47.78±0.7	54.02±0.6	54.99±0.7	57.61±0.7	**61.13**±0.8
		80%	49.53	54.58	54.93	58.47	51.16	50.27±0.5	54.17±0.2	55.78±0.6	58.01±0.7	**62.40**±0.6
	Micro-F1	40%	57.11	60.41	62.53	**64.21**	53.24	57.82±0.7	58.36±0.5	59.96±0.7	62.05±0.5	63.52±0.7
		60%	57.34	60.56	62.59	64.38	56.88	57.66±0.5	60.28±0.3	61.19±0.6	62.92±0.4	**64.84**±0.4
		80%	57.88	60.66	62.64	64.65	60.72	58.07±0.6	60.49±0.3	61.77±0.6	63.32±0.7	**66.10**±0.3
Patent	Macro-F1	40%	72.32±0.9	69.01±0.9	-	-	-	-	77.57±0.8	79.67±0.4	81.51±0.4	**83.53**±0.6
		60%	72.25±1.2	69.08±0.9	-	-	-	-	77.69±0.6	79.76±0.5	81.56±0.6	**83.85**±0.7
		80%	72.05±1.1	68.99±1.0	-	-	-	-	77.67±0.4	80.13±0.4	81.57±0.5	**83.90**±0.6
	Micro-F1	40%	71.57±1.3	68.14±0.9	-	-	-	-	77.64±0.7	79.39±0.5	80.79±0.7	**83.48**±0.8
		60%	71.53±1.0	68.20±0.7	-	-	-	-	77.89±0.5	79.75±0.3	80.81±0.6	**83.80**±0.7
		80%	71.38±1.2	68.10±0.5	-	-	-	-	77.93±0.4	80.01±0.3	80.93±0.6	**83.88**±0.9

Table 2: Quantitative results (%) on the temporal node classification task. The results are averaged over five runs, where the best results in each row are highlighted in **boldfaced**. (Tr.ratio: trainig ratio)

the details about the datasets and experimental settings in Appendix.

Overall Performance

In our experiments, we consider the task of temporal node classification to evaluate the representation quality. In this task, the graph structure at each snapshot is entirely available for representation learning. We follow (Lu et al. 2019) and examine the performance when different sizes of training datasets are used, i.e., 40%, 60%, and 80%, including 5% for validation. For unsupervised methods, DeepWalk, Node2Vec, HTNE, M^2DNE, and DyTriad, a two-layer MLP with hidden units 128 are trained to solve the classification problem. Table 2 summarizes the performance of all the compared methods on the temporal node classification task. We repeat the experiment for five runs with different random seeds and report the average results with standard deviation in terms of both Macro-F1 and Micro-F1 scores. We reuse the results of DeepWalk, Node2Vec, HTNE, M^2DNE, and DyTriad on DBLP and Tmall already reported in (Lu et al. 2019).

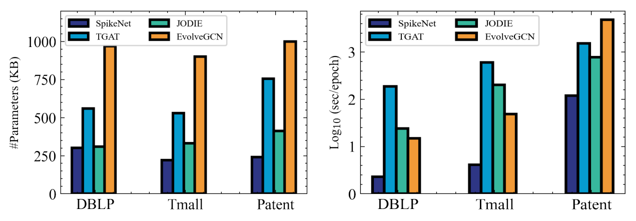
As shown in Table 2, methods developed for static graphs (i.e., DeepWalk and Node2Vec) generally underperform those for temporal graphs, which suggests that the temporal dynamics of the graph structure are indeed critical to learn a better node representation. We can also observe that the performance of unsupervised methods remains stable as the ratio of the training set increases. By contrast, MPNN, TGAT, EvolveGCN, JODIE, and SpikeNet can benefit from the increasing training data. This is due to the fact that supervised methods can incorporate supervision signals into learning and facilitate training. In this regard, supervised methods are also capable of outperforming most unsupervised ones. On the largest graph dataset Patent, HTNE, M^2DNE, and DyTriad fail to learn properly since the estimation process is time-consuming and thus too expensive to model large graphs. MPNN, an LSTM-based approach is also not available for Patent, since it requires high computation overheads to train the model in a full-batch fashion.

Although SpikeNet uses discrete spiking signals to communicate between layers rather than real-valued, it achieves state-of-the-art performance in most cases. In particular, SpikeNet slightly underperforms TGAT and M2DNE on DBLP and Tmall when the training data is small. As the training data increases, SpikeNet is able to outperform strong baselines. Notably, SpikeNet achieves about 4.7% and 4.3% performance improvement on average over the strongest baseline in Tmall and Patent as measured by Micro-F1, respectively. One potential reason could be that the surrogate learning technique requires more training data to better approximate the backward gradients. In addition, more training data can also help the threshold update strategy to adjust the neuron thresholds adaptively. The results indicate that the SNNs-based framework provides a simple yet effective way for dynamic graph representation learning.

Overhead Evaluation

In this section, we compare the overheads between our proposed SpikeNet and other baseline approaches that are most relevant to our work.

Parameter size The number of parameters of different methods is shown in Figure 5(a). It can be seen that SpikeNet is much more lightweight with only around 50% parameters as compared to TGAT, which uses a multi-head attention mechanism and requires a lot of parameters to learn. Particularly, EvolveGCN which combines GNNs with RNNs has around four times the parameter size than SpikeNet. However, more parameters would result in high memory overheads and the overfitting issue. In this regard, SpikeNet replaces the RNN cell with a parameter-free LIF model, which significantly reduces the number of parameters to be optimized almost without losing accuracy.



(a) Parameter size (b) Training time

Figure 5: Overheads of different methods in terms of parameter size and training time. Methods that cannot run on Patent are excluded for comparison.

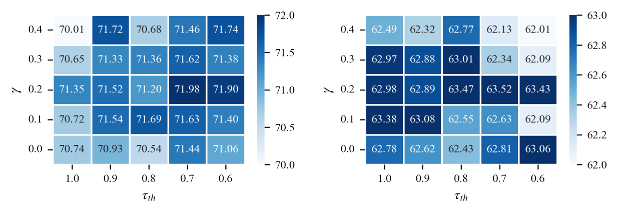
Runtime complexity The training time of each epoch for different methods is shown in Figure 5(b). The experiments were run on a Titan RTX GPU with the same batch size (except for EvolveGCN which is fully batch trained) for a fair comparison. Through the experiment, we find that SpikeNet holds obvious efficiency superiority during training. Specifically, SpikeNet can obtain orders of magnitude speedup over baselines. We believe the significant improvement can be attributed to the following two reasons.: (i) Compared to EvolveGCN, SpikeNet trains the model on a small sampled subgraph instead of the whole graph during training, so the training time is significantly smaller. (ii) Compared to JODIE and TGAT, SpikeNet benefits from the lightweight temporal architecture (i.e., LIF model), with much fewer computation overheads and parameters to be optimized. The simplification also speeds up the training. Note that, the masked summation operation between two matrices is currently not well supported in the deep learning framework, so we use matrix multiplication instead to facilitate the implementation. In the near future, SpikeNet could also benefit from neuromorphic chips and speed up the implementation.

Ablation Study

To further investigate the impact of different hyperparameters in SpikeNet, we conduct several experiments to analyze them from different perspectives.

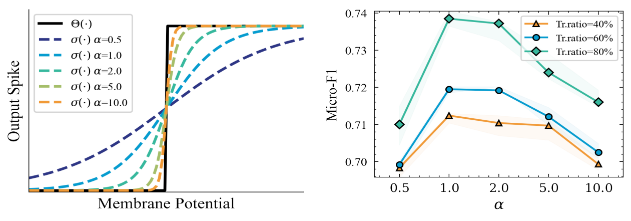
Threshold decay τ_{th} and γ As one of our main contributions, the threshold decay strategy is adopted for LIF model to stablize training. To comprehensively explore the effectiveness of threshold decay strategy, we cownduct temporal node classification on DBLP and Tmall by varying the values of τ_{th} and γ as $\{1.0, 0.9, 0.8, 0.7, 0.6\}$ and $\{0., 0.1, 0.2, 0.3, 0.4\}$, respectively. The experiments are repeated 5 times and the average results are shown in Figure 6. It is observed that the threshold decay strategy generally benefit the learning of SpikeNet on two datasets, as evidenced by decreasing τ_{th} and increasing γ simultaneously can improve the performance of SpikeNet. When $\tau_{\text{th}} = 0.7$ and $\gamma = 0.2$, SpikeNet achieves the best performance, which is much better than not using decay. Overall, the results demonstrate that the threshold decay strategy is beneficial for the model performance.

Smooth factor α Since surrogate learning technique is important for training the SNN-based model, we explore



(a) DBLP (b) Tmall

Figure 6: Ablation analysis w.r.t. τ_{th} and γ on DBLP and Tmall, respectively. Here we report Micro-F1 results with Tr.ratio=0.4 for both datasets. $\tau_{\text{th}} = 1.0$ and $\gamma = 0$ denote the standard case without thresholds decay.



(a) $\Theta(\cdot)$ and $\delta(\cdot)$ (b) Classification performance

Figure 7: Ablation analysis w.r.t. smooth factor α on DBLP.

the sensitivity of smooth factor α in SpikeNet. We vary the smooth factor α from $\{0.5, 1.0, 2.0, 5.0, 10.0\}$ to study the effects in the surrogate function $\sigma(\cdot)$. We only report the results on DBLP, because we have similar observations for other datasets. As shown in Figure 7(a), it is observed that $\sigma(\cdot)$ can better approximate the Heaviside step function $\Theta(\cdot)$ with the increase of α. However, the best performance is achieved when $\alpha = 1.0$ according to Figure 7(b). This appears to be quite straightforward since higher α will suffer from the vanishing and exploding gradient problem. Overall, SpikeNet is senstive to α and the choice of a good α is important to learn the model properly.

Conclusion and Future Work

We present SpikeNet, a scalable framework built upon SNNs to capture the evolving dynamics underlying a discrete graph sequence. To our best knowledge, SpikeNet is the first research effort to exploit SNNs on temporal graphs. The key insight behind our approach is to aggregate and update temporal information from a dynamically sampled neighborhood via an integrate-and-fire mechanism. As a result, SpikeNet enjoys the advantage of exploiting graph dynamics as well as significantly fewer parameters and overheads. SpikeNet inherently allows inductive learning, making it applicable to predict evolving dynamics on temporal graphs with unseen nodes. Experiments on three real-world datasets show that SpikeNet outperforms the recent state-of-the-art methods in most cases. Through further analysis of these results, we also show that SpikeNet provides better trade-offs between performance and computational costs particularly compared to RNN-based methods. Since our work is a time-driven model that focuses on discrete-time temporal graphs, in future work we aim to study the event-driven SNNs to capture structural and dynamic properties on continuous-time temporal graphs at a fine-grained level.

References

Bu, T.; Ding, J.; Yu, Z.; and Huang, T. 2022. Optimized Potential Initialization for Low-Latency Spiking Neural Networks. In *AAAI*, 11–20. AAAI Press.

Chen, L.; Li, J.; Peng, Q.; Liu, Y.; Zheng, Z.; and Yang, C. 2021. Understanding Structural Vulnerability in Graph Convolutional Networks. In Zhou, Z.-H., ed., *IJCAI*, 2249–2255. IJCAI. Main Track.

Chen, L.; Peng, J.; Liu, Y.; Li, J.; Xie, F.; and Zheng, Z. 2020. Phishing scams detection in ethereum transaction network. *ACM Transactions on Internet Technology (TOIT)*, 21(1): 1–16.

Chian, V. C.; Hildebrandt, M.; Runkler, T. A.; and Dold, D. 2021. Learning through structure: towards deep neuromorphic knowledge graph embeddings. *CoRR*, abs/2109.10376.

Cho, K.; Van Merriënboer, B.; Gulcehre, C.; Bahdanau, D.; Bougares, F.; Schwenk, H.; and Bengio, Y. 2014. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*.

Davies, M.; Srinivasa, N.; Lin, T.-H.; Chinya, G.; Cao, Y.; Choday, S. H.; Dimou, G.; Joshi, P.; Imam, N.; Jain, S.; et al. 2018. Loihi: A neuromorphic manycore processor with on-chip learning. *Ieee Micro*, 38(1): 82–99.

Dold, D.; and Garrido, J. S. 2021. SpikE: spike-based embeddings for multi-relational graph data. In *IJCNN*, 1–8. IEEE.

Fang, W.; Yu, Z.; Chen, Y.; Masquelier, T.; Huang, T.; and Tian, Y. 2020. Incorporating learnable membrane time constant to enhance learning of spiking neural networks. *arXiv preprint arXiv:2007.05785*.

Gerstner, W.; and Kistler, W. M. 2002. *Spiking Neuron Models: Single Neurons, Populations, Plasticity*. Cambridge University Press.

Grover, A.; and Leskovec, J. 2016. node2vec: Scalable Feature Learning for Networks. In *KDD*, 855–864. ACM.

Hall, B. H.; Jaffe, A. B.; and Trajtenberg, M. 2001. The NBER patent citation data file: Lessons, insights and methodological tools.

Hamilton, W. L.; Ying, Z.; and Leskovec, J. 2017. Inductive Representation Learning on Large Graphs. In *NIPS*, 1024–1034.

Hochreiter, S.; and Schmidhuber, J. 1997. Long short-term memory. *Neural computation*, 9(8): 1735–1780.

Hu, W.; Fey, M.; Zitnik, M.; Dong, Y.; Ren, H.; Liu, B.; Catasta, M.; and Leskovec, J. 2020. Open Graph Benchmark: Datasets for Machine Learning on Graphs. In *NeurIPS*.

Hunsberger, Eric. 2018. Spiking Deep Neural Networks: Engineered and Biological Approaches to Object Recognition.

Izhikevich, E. M. 2004. Which model to use for cortical spiking neurons? *IEEE Trans. Neural Networks*, 15(5): 1063–1070.

Kazemi, S. M.; Goel, R.; Jain, K.; Kobyzev, I.; Sethi, A.; Forsyth, P.; and Poupart, P. 2020. Representation Learning for Dynamic Graphs: A Survey. *J. Mach. Learn. Res.*, 21: 70:1–70:73.

Kim, S.; Park, S.; Na, B.; and Yoon, S. 2020. Spiking-YOLO: spiking neural network for energy-efficient object detection. In *AAAI*, 11270–11277.

Kim, Y.; Chough, J.; and Panda, P. 2021. Beyond Classification: Directly Training Spiking Neural Networks for Semantic Segmentation. *arXiv preprint arXiv:2110.07742*.

Kumar, S.; Zhang, X.; and Leskovec, J. 2019. Predicting Dynamic Embedding Trajectory in Temporal Interaction Networks. In *KDD*, KDD '19, 1269–1278. New York, NY, USA: ACM.

Li, J.; Xu, K.; Chen, L.; Zheng, Z.; and Liu, X. 2021. Graph-Gallery: A Platform for Fast Benchmarking and Easy Development of Graph Neural Networks Based Intelligent Software. In *2021 IEEE/ACM 43rd International Conference on Software Engineering: Companion Proceedings (ICSE-Companion)*, 13–16. Los Alamitos, CA, USA: IEEE Computer Society.

Liu, Y.; Liang, C.; He, X.; Peng, J.; Zheng, Z.; and Tang, J. 2020. Modelling high-order social relations for item recommendation. *IEEE Transactions on Knowledge and Data Engineering*.

Loshchilov, I.; and Hutter, F. 2019. Decoupled Weight Decay Regularization. In *ICLR (Poster)*. OpenReview.net.

Lu, Y.; Wang, X.; Shi, C.; Yu, P. S.; and Ye, Y. 2019. Temporal Network Embedding with Micro- and Macro-dynamics. In *CIKM*, 469–478. ACM.

Ma, Y.; Guo, Z.; Ren, Z.; Tang, J.; and Yin, D. 2020. Streaming Graph Neural Networks. In *SIGIR*, 719–728. ACM.

Makarov, I.; Savchenko, A. V.; Korovko, A.; Sherstyuk, L.; Severin, N.; Mikheev, A.; and Babaev, D. 2021. Temporal Graph Network Embedding with Causal Anonymous Walks Representations. *CoRR*, abs/2108.08754.

Neftci, E. O.; Mostafa, H.; and Zenke, F. 2019. Surrogate Gradient Learning in Spiking Neural Networks. *CoRR*, abs/1901.09948.

Panagopoulos, G.; Nikolentzos, G.; and Vazirgiannis, M. 2021. Transfer Graph Neural Networks for Pandemic Forecasting. In *AAAI*, 4838–4845. AAAI Press.

Pareja, A.; Domeniconi, G.; Chen, J.; Ma, T.; Suzumura, T.; Kanezashi, H.; Kaler, T.; Schardl, T. B.; and Leiserson, C. E. 2020. EvolveGCN: Evolving Graph Convolutional Networks for Dynamic Graphs. In *AAAI*, 5363–5370. AAAI Press.

Paszke, A.; Gross, S.; Massa, F.; Lerer, A.; Bradbury, J.; Chanan, G.; Killeen, T.; Lin, Z.; Gimelshein, N.; Antiga, L.; Desmaison, A.; Köpf, A.; Yang, E. Z.; DeVito, Z.; Raison, M.; Tejani, A.; Chilamkurthy, S.; Steiner, B.; Fang, L.; Bai, J.; and Chintala, S. 2019. PyTorch: An Imperative Style, High-Performance Deep Learning Library. In *NeurIPS*, 8024–8035.

Patel, K.; Hunsberger, E.; Batir, S.; and Eliasmith, C. 2021. A spiking neural network for image segmentation. *arXiv preprint arXiv:2106.08921*.

Perozzi, B.; Al-Rfou, R.; and Skiena, S. 2014. DeepWalk: online learning of social representations. In *KDD*, 701–710. ACM.

Rossi, E.; Chamberlain, B.; Frasca, F.; Eynard, D.; Monti, F.; and Bronstein, M. M. 2020. Temporal Graph Networks for Deep Learning on Dynamic Graphs. *CoRR*, abs/2006.10637.

Sajjad, H. P.; Docherty, A.; and Tyshetskiy, Y. 2019. Efficient Representation Learning Using Random Walks for Dynamic Graphs. *CoRR*, abs/1901.01346.

Sak, H.; Senior, A. W.; and Beaufays, F. 2014. Long Short-Term Memory Based Recurrent Neural Network Architectures for Large Vocabulary Speech Recognition. *CoRR*, abs/1402.1128.

Salinas, E.; and Sejnowski, T. J. 2002. Integrate-and-fire neurons driven by correlated stochastic input. *Neural computation*, 14(9): 2111–2155.

Shi, M.; Huang, Y.; Zhu, X.; Tang, Y.; Zhuang, Y.; and Liu, J. 2021. GAEN: Graph Attention Evolving Networks. In *IJCAI*, 1541–1547. ijcai.org.

Trivedi, R.; Farajtabar, M.; Biswal, P.; and Zha, H. 2019. DyRep: Learning Representations over Dynamic Graphs. In *ICLR (Poster)*. OpenReview.net.

Vanarse, A.; Osseiran, A.; Rassau, A.; and van der Made, P. 2019. A hardware-deployable neuromorphic solution for encoding and classification of electronic nose data. *Sensors*, 19(22): 4831.

Wang, Y.; Chang, Y.; Liu, Y.; Leskovec, J.; and Li, P. 2021. Inductive Representation Learning in Temporal Networks via Causal Anonymous Walks. In *ICLR*. OpenReview.net.

Wu, J.; Chen, X.; Xu, K.; and Li, S. 2022. Structural Entropy Guided Graph Hierarchical Pooling. In *ICML*, volume 162 of *Proceedings of Machine Learning Research*, 24017–24030. PMLR.

Xu, D.; Cheng, W.; Luo, D.; Liu, X.; and Zhang, X. 2019. Spatio-Temporal Attentive RNN for Node Classification in Temporal Attributed Graphs. In *IJCAI*, 3947–3953. ijcai.org.

Xu, D.; Liang, J.; Cheng, W.; Wei, H.; Chen, H.; and Zhang, X. 2021a. Transformer-Style Relational Reasoning with Dynamic Memory Updating for Temporal Network Modeling. In *AAAI*, 4546–4554. AAAI Press.

Xu, D.; Ruan, C.; Körpeoglu, E.; Kumar, S.; and Achan, K. 2020. Inductive representation learning on temporal graphs. In *ICLR*. OpenReview.net.

Xu, M.; Wu, Y.; Deng, L.; Liu, F.; Li, G.; and Pei, J. 2021b. Exploiting Spiking Dynamics with Spatial-temporal Feature Normalization in Graph Learning. In *IJCAI*, 3207–3213. Main Track.

Zenke, F.; and Vogels, T. P. 2021. The Remarkable Robustness of Surrogate Gradient Learning for Instilling Complex Function in Spiking Neural Networks. *Neural Computation*, 33(4): 899–925.

Zhou, J.; Cui, G.; Hu, S.; Zhang, Z.; Yang, C.; Liu, Z.; Wang, L.; Li, C.; and Sun, M. 2020. Graph neural networks: A review of methods and applications. *AI Open*, 1: 57–81.

Zhou, L.; Yang, Y.; Ren, X.; Wu, F.; and Zhuang, Y. 2018. Dynamic Network Embedding by Modeling Triadic Closure Process. In *AAAI*, 571–578. AAAI Press.

Zhu, Z.; Peng, J.; Li, J.; Chen, L.; Yu, Q.; and Luo, S. 2022. Spiking Graph Convolutional Networks. In *IJCAI*, 2434–2440. IIJCAI Organization. Main Track.

Zuo, Y.; Liu, G.; Lin, H.; Guo, J.; Hu, X.; and Wu, J. 2018. Embedding Temporal Network via Neighborhood Formation. In *KDD*, 2857–2866. ACM.

Algorithm

Algorithm 1: SpikeNet embedding generation algorithm (forward propagation)

```
Input:
    Temporal graph $\mathcal{G} = \{\mathcal{G}^1, \mathcal{G}^2, \dots, \mathcal{G}^T\}$; input node features $\mathcal{X} = \{X^1, X^2, \dots, X^T\}$; depth $K$ and time span $T$; weight matrices $\mathbf{W}^{(k)}$, neighborhood sampler SAMPLE, differentiable aggregation function AGG and spike pooling function Linear; LIF models $\delta^{(k)}$ $\forall k \in \{1, \dots, K\}$;
Output:
    Node representations $z_v$ for all $v \in \mathcal{V}$;
 1: $h_v^{t,(0)} \leftarrow x_v^t, \forall v \in \mathcal{V}, t \in \{1, \dots, T\}$;
 2: for $t = 1$ to $T$ do
 3:     for $k = 1$ to $K$ do
 4:         for $v \in \mathcal{V}$ do
 5:             $\mathcal{S}^t(v) \leftarrow \mathrm{SAMPLE}(v, \mathcal{G}^t) \cup \mathrm{SAMPLE}(v, \Delta\mathcal{G}^t)$;
 6:             $h_{\mathcal{S}}^{t,(k)} \leftarrow \mathrm{AGG}\left(\{\mathbf{W}^{(k)}[h_u^{t,(k-1)}], \forall u \in \mathcal{S}^t\}\right)$;
 7:             $h_v^{t,(k)} \leftarrow \delta^{(k)}\left(\mathbf{W}^{(k)}[h_v^{t,(k-1)}] + h_{\mathcal{S}}^{t,(k)}\right)$;
 8:         end for
 9:     end for
10: end for
11: return $z_v \leftarrow \mathrm{Linear}\left(\{h_v^{t,(K)} \| \dots \| h_v^{T,(K)}\}\right)$, $\forall v \in \mathcal{V}$;
```

Algorithm 1 provides an overview of the SpikeNet Embedding generation (i.e., forward propagation), the node embedding at each layer and each time step is generated by aggregating neighborhood information, followed by a LIF model to capture the evolving dynamics. It is worth noting that the forward propagation allows the multiply-and-accumulate typically inherent in matrix multiplication to be turned into simply matrix masking (or indexing) and accumulation, i.e., masked summation, which could be implemented with more energy-efficient neuromorphic hardware such as Intel Loihi (Davies et al. 2018) or Brainchip

Akida (Vanarse et al. 2019). In this manner, our method may lead to more energy-efficient implementations of GNNs on temporal graphs and will be meaningful and influential in the future.

Experimental setup

Dataset Statistics

We adopt three temporal graph datasets with different scales: DBLP, Tmall, and Patent. We briefly give an overview of the datasets:

DBLP (Lu et al. 2019). DBLP is an academic co-author graph extracted from the bibliography website. in which each node is an author and each edge means the two authors collaborated on a paper. The authors in DBLP are labeled according to their research areas.

Tmall (Lu et al. 2019). Tmall is a bipartite graph extracted from the sales data in 2014 at Tmall.com, in which each node refers to either one user or one item and each edge refers to one purchase with a timestamp. The five most frequently purchased categories are treated as labels in experiments. Note that Tmall is a partially labeled dataset in which only items are assigned with categories. We follow the pre-processing scheme as described in (Lu et al. 2019) that selects the five most frequently purchased categories as labels in experiments.

Patent (Hall, Jaffe, and Trajtenberg 2001). Patent is a citation network of US patent ranging from year 1963 to 1999. Each patent belongs to six different types of patent categories. Patent is the largest size of graph among all three datasets.

Baseline Methods

We compare the performance of SpikeNet with the following baselines, including static and dynamic graph representation learning methods:

1. **DeepWalk** (Perozzi, Al-Rfou, and Skiena 2014) performs random walks to generate an ordered sequence of nodes from a static graph to create contexts for each node, then applies a skip-gram model to these sequences to learn representations.
2. **Node2Vec** (Grover and Leskovec 2016) extends DeepWalk with biased random walks to explore the neighborhood of a node and learn node representations on a static graph.
3. **HTNE** (Zuo et al. 2018) integrates the Hawkes process and attention mechanism into network embedding to model the neighborhood formation sequences.
4. **M^{2}DNE** (Lu et al. 2019) designs a temporal attention point process and a general dynamics equation to capture structural and temporal properties of evolving graphs.
5. **DynamicTriad** (DyTriad) (Zhou et al. 2018) models both structural information and evolution patterns based on the triadic closure process, which enables the model to capture the graph dynamics effectively.
6. **MPNN-LSTM** (MPNN) (Panagopoulos, Nikolentzos, and Vazirgiannis 2021) is a time-series version of message passing neural network (MPNN) with two-layer LSTM (Hochreiter and Schmidhuber 1997). MPNN can capture the long-range temporal dependencies in temporal graphs based on the dynamics encoded into the node representations.
7. **JODIE** (Kumar, Zhang, and Leskovec 2019) employs two RNNs to update the node embeddings at every observed interaction and uses a projection operation that predicts the future embedding trajectory in an evoloving graph.
8. **EvolveGCN** (Pareja et al. 2020) uses the RNN to evolve the GNN parameters at each time step, which effectively performs model adaptation and captures the dynamics of the graph sequence.
9. **TGAT** (Xu et al. 2020) utilizes self-attention mechanisms as basic blocks and derive a functional time encoding to effectively learn temporal and topological information, which achieves state-of-the-art results for different dynamic graph learning tasks.

For DeepWalk and Node2Vec that are originally designed for static graphs, we accumulate historical information in the graph sequence and represent the structure at the last time period, i.e., $\mathcal{G}^T$. More importantly, there are several recent works (Shi et al. 2021; Xu et al. 2019, 2021a; Rossi et al. 2020; Ma et al. 2020) that are not considered for comparison in our experiments because they require very high memory footprints for modeling the temporal graphs and cannot run on a reasonably sized GPU (24GB).

Implementation Details

Each experiment is conducted five times and average results with standard deviation are reported. The hyperparameters are tuned based on the performance of validation set. More specifically, we implement DeepWalk and Node2Vec using the open-source library GraphGallery (Li et al. 2021) with an embedding size of 128. For dynamic methods, we use the code provided by the authors and closely follow the experimental setup in (Lu et al. 2019), so as to reduce the experiment workload and make a fair comparison. For other parameters, we follow the setup in (Lu et al. 2019).

For SpikeNet, it supports batch training for both transductive and inductive tasks where the batch size is tuned in [512, 1024, 2048, 4096]. The learning rate is ranged in [0.001, 0.0003, 0.005, 0.008, 0.01] with AdamW (Loshchilov and Hutter 2019) optimizer. We use two-layer structure for SpikeNet and the aggregate function AGG is set as *mean*. We set the embedding size of 128, 512, and 512 for DBLP, Tmall, and Patent. For the temporal neighborhood sampling, we perform uniformly sampling for a node to build the samples set as in (Hamilton, Ying, and Leskovec 2017), where the number of sampled nodes in each layer is set as 5 and 3 across all datasets. In addition, half of the nodes are sampled for capturing macro-dynamics and micro-dynamics, respectively.

For the LIF model, we fix $\tau_m = 1.0$ for all datasets. We empirically set the membrane reset potential $V_{\text{reset}} = 0$, firing potential $V_{\text{th}} = 1.0$ and threshold decays $\tau_{\text{th}} = 0.7$, $\gamma = 0.2$ for all datasets. To facilitate surrogate gradient

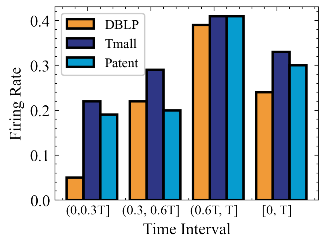


Figure 8: The average firing rate of SpikeNet at different time intervals.

learning, the smooth Sigmoid function is used as a threshold function to approximate the gradients during backpropagation with smooth factor $\alpha = 1.0$ across all datasets. All models are implemented with PyTorch (Paszke et al. 2019). All our experiments are conducted on one NVIDIA TITAN RTX GPU with 24GB memory.

Additional Experimental Results

Firing sparsity Benefiting from intrinsic neuronal dynamics and spike-based communication paradigms, SNNs can be easily applied on some specialized neuromorphic hardware, such as Intel Loihi (Davies et al. 2018) or Brainchip Akida (Vanarse et al. 2019). As for the power-efficient neuromorphic computation, the firing rate is an important property for SNNs since the energy consumption is proportional to the number of spikes (Patel et al. 2021). Reduction of firing rate is essential for energy-efficient neuromorphic chips. To this end, we hereby calculate the average firing rate of SpikeNet in the intermediate representations at a different time interval and show the results in Figure 8. Specifically, the firing rate is calculated by $\#spikes/(\#time\ steps \times \#neurons)$. A neuron with a high firing rate will lose the advantages of temporal sparsity and harm the effective representation in a temporal graph. In DBLP, SpikeNet has a lower firing rate, especially in the prior time steps. The result suggests that the evolutionary pattern of nodes and edges in DBLP in short term is not significant, which leads to a sparser response compared with Tmall and Patent. In addition, the results also show that the intermediate node representations of SpikeNet are much sparse (only 20% to 30% of elements are non-zero). In other words, we need only about 20% to 30% of the memory to store the intermediate node representations. Moreover, each dimension of the embedding vectors learned by SpikeNet is only encoded by 1 bit (binary spikes), different from the real-valued embedding vectors that are encoded by at least 32 bits. As a result, the binarized representations can also greatly reduce the memory and time cost for the downstream tasks.