Temporal Smoother: a Dynamic Graph Approach

Tommaso Castellani

1 Introduction

Many graph-structured datasets are not static but evolve over time as new edges and nodes emerge and as their attributes change. This dynamic structure creates significant challenges in encoding time and capturing sequential dependencies, since models must now account for both spatial and temporal dimensions.

This project has three main objectives. First, we aim to provide a formal introduction to dynamic graphs, along with a literature review that outlines the two primary approaches for modeling the temporal dimension in graphs. Second, we propose a new model based on the concept of temporal smoothing, which seeks to capture the overall temporal trend in graph evolution using attention mechanisms. Finally, we plan to test our model against benchmarks obtained on the Neurograph datasets [12] in the context of brain connectomics. This is a challenging dataset where researchers have struggled to achieve top performance in the dynamic setting; indeed, transitioning from a static to a dynamic framework has brought little or no improvement.

The main focus of the review will be on classification tasks for Snapshot Temporal Graphs, where the data are in the format of a sequence of static graphs and the goal is to classify either the individual snapshot or the sequence itself into predefined classes. In this framework, the problem can be approached in two different ways: either by treating the temporal and spatial dimensions as separate components or by modeling them together. The first class of models is the most popular, intuitively applying the same spatial models to all the snapshots of a sequence and then applying the temporal component on the results. Hence, it transforms the graph into treatable Euclidean data

and then applies standard methods for time-dependent data. The other class attempts to model the temporal and spatial dimensions together so that the spatial component for each graph is not static but rather depends on its time component.
Secondly, we present a new architecture that exploits the concept of temporal smoothing, closely related to spatial smoothing. Most GNN techniques are based on some form of message passing, where features are aggregated from neighboring nodes using functions that effectively smooth the node feature space. We extend this concept to the temporal dimension, leveraging an attention mechanism with the intuition that the model should aggregate past information with weights proportional to the difference between the current state and the previous ones—incorporating previous data only when it significantly differs from the current situation.
Finally, we will focus on dynamic graphs in brain connectomics [12], specifically examining two dynamic datasets. While GNNs have become the gold standard for static analysis in brain connectomics, adding the temporal dimension has not improved, and in some cases even decreased, the quality of the results. We replicate the baseline model that was originally applied on this dataset and we apply the Temporal Smoother Model matching the baseline performance and outperforming in part of the metrics. The code to replicate our experiments can be found here: GitHub Repository

2 Temporal Graphs

In this section, we provide an introduction and definition of Temporal Graphs. Broadly speaking, a Temporal Graph refers to a graph whose structure changes over time. Almost any part of the graph structure can change, starting with the nodes, which may exist only during certain time intervals, and their attributes, which can vary over time. The same applies to edges, which can change in terms of presence, weight, and other attributes. The definition of dynamic graph is formally consistent among the major theoretical papers on the topic [8, 17, 2, 15, 1]. Mathematically, can be exrpessed as:

Definition 1 ***Temporal Graph - TG.***
A Temporal Graph (TG) is a tuple $G_T = (V, E, V_T, E_T)$, where:

- *V and E are, respectively, the set of all possible nodes and edges appearing in a graph at any time.*
- *$V_T := \{(v, x^v, t_s, t_e) \mid v \in V, x^v \in \mathbb{R}^{d_V}, t_s \leq t_e\}$ represents nodes with their associated features and temporal intervals.*
- *$E_T := \{(e, x^e, t_s, t_e) \mid e \in E, x^e \in \mathbb{R}^{d_E}, t_s \leq t_e\}$ represents edges with their attributes and temporal intervals.*

Note that we can define the adjacency matrix A_t and the feature matrix X_t to be time-specific.
On the other hand the categorization of dynamic graph is slightly different across the literature. Most papers [17, 2, 15] categories dynamic graphs into Continuos and Descrete Time, we instead prefer another categoriazation [8], where the distinction is based on whether the data is represented as a sequence of static graphs or as a sequence of events that occur in an otherwise static graph in continuous time, such as edge and node additions/deletions, as well as other structural changes.
Thus, we define both types of graphs in a mathematically rigorous manner using definitions from [8].

Definition 2 (Snapshot-based Temporal Graph - STG)
Let $t_1 < t_2 < \cdots < t_n$ be the ordered set of all timestamps t_s, t_e occurring in a TG G_T. Set

$$V_i := \{(v, x^v) : (v, x^v, t_s, t_e) \in V_T, t_s \leq t_i \leq t_e\},$$

$$E_i := \{(e, x^e) : (e, x^e, t_s, t_e) \in E_T, t_s \leq t_i \leq t_e\},$$

and define the snapshots $G_i := (V_i, E_i)$, $i = 1, \ldots, n$.

Hence in this type of graph we have a sequence of graphs that are temporal snapshots of the same dynamical sequence.

Definition 3 (Event-based Temporal Graph - ETG)
Let G_T be a TG, and let ε denote one of the following events:

- ***Node insertion** $\varepsilon_V^+ := (v, t)$: the node v is added to G_T at time t, i.e., there exists $(v, x^v, t_s, t_e) \in V_T$ with $t_s = t$.*
- ***Node deletion** $\varepsilon_V^- := (v, t)$: the node v is removed from G_T at time t, i.e., there exists $(v, x^v, t_s, t_e) \in V_T$ with $t_e = t$.*

- ***Edge insertion*** $\varepsilon_E^+ := (e,t)$*: the edge* e *is added to* G_T *at time* t*, i.e., there exists* $(e, x^e, t_s, t_e) \in E_T$ *with* $t_s = t$*.*
- ***Edge deletion*** $\varepsilon_E^- := (e,t)$*: the edge* e *is removed from* G_T *at time* t*, i.e., there exists* $(e, x^e, t_s, t_e) \in E_T$ *with* $t_e = t$*.*

In this type of dynamic graph we instead have a sequence of events that happen at a specific time.
Note however that that it is always possible to transform an Event Based Problem into Snapshot problem by "discretizing" while the converse is not always true. The methods and approaches taken differ based on the type of dynamic graph that we have. For the purpose of this project I will focus on the first class of Dynamic Graphs.

3 Literature Review

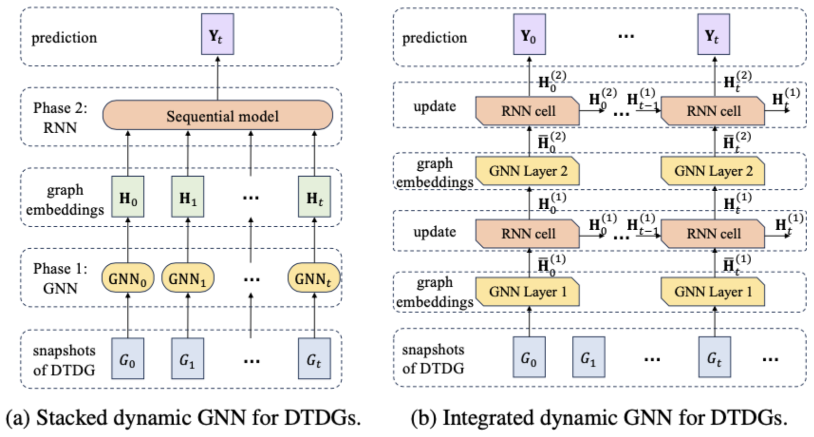

Figure 1: Different Model Architectures for Dynamic Graphs [17]

In this section, we propose a qualitative literature review on methods used to model Snapshot Temporal Graphs, with a focus on Temporal GNNs. We concentrate on methods that take as input a dynamic graph, a list of snapshots characterized by its own adjacency matrix, edge list, and nodes' feature matrix, and employ a Graph Neural Network together with a temporal unit. It is worth noting that, as widely discussed in [2, 1], a variety of methods beyond GNNs have been developed. These include autoencoders, random

walk methods, non-negative matrix factorizations, and kernel methods, each combined with a temporal unit such as ARMA/ARIMA.
Before delving into the analysis of the various methods, we will list all possible tasks and goals that can be achieved, which can be broadly divided into supervised and unsupervised approaches.
On the supervised side, we can perform:

- Classification for nodes, edges, snapshots, or whole sequences
- Regression
- Link prediction

On the unsupervised side:

- Clustering for nodes or snapshots
- Low-dimensional embedding for the nodes, snapshots, or the whole sequence

For the purpose of this project, we will restrict our scope to methods that are or can be adapted to classification at the snapshot and whole sequence levels. More formally, as defined in [8]:

Definition 4 (Temporal Graph Classification)
Let $G_{\mathcal{T}}$ be a domain of temporal graphs (TGs). The graph classification task requires learning a function

$$f_{\mathrm{GC}} : G_{\mathcal{T}} \times I^{+} \;\to\; \mathcal{C}$$

that maps a temporal graph, restricted to a time interval $[t_s, t_e] \in I^{+}$, into a class.

Note that the classification of a single snapshot and of the whole sequence is included in this definition.

3.1 Classification

Methods for snapshot temporal graphs (STG) that target graph classification can be divided into two subclasses, as suggested in [17] (see Figure 1). Namely, Stacked Dynamic GNNs where models combine spatial GNNs with separate temporal units in a modular fashion. As shown in Figure 1(a), distinct components are assigned to capture spatial features and temporal dynamics independently. Integrated Dynamic GNNs instead jointly model spatial and temporal information within a single module, as illustrated in Figure 1(b). This approach seeks to capture complex spatiotemporal dependencies, although at the cost of increased computational complexity.

3.1.1 Stacked Dynamic GNNs

This class includes several models especially earlier methods that extend static GNNs by appending a temporal unit on top. Because each snapshot is processed independently, these methods are generally less computationally expensive and easily parallelizable. Successful models in this class include:

Dynamic GCNs: [9] The Waterfall Dynamic-GCN variant handles dynamic graphs by applying the same graph convolution at each time step to capture the structural information. These spatial features are then passed to a modified LSTM that learns the temporal evolution of the node features. The convolutional filters are shared across different time steps ensuring computational efficiency and consistency.The temporal module, built on an LSTM framework, aggregates insights from each time step, capturing long-term dependencies crucial for understanding evolving patterns. This design not only handles variations in the graph structure efficiently but also reduces the number of parameters needed. Overall, the Waterfall Dynamic-GCN is well-suited for classification tasks on dynamic graphs, it produces an embedding for each snapshot.

DySAT: [13] DySAT is a dynamic graph representation model that leverages self-attention to capture both the spatial representations and temporal evolution of nodes. The model first applies a structural self-attention mechanism to each graph snapshot, allowing it to aggregate information from a node's local neighborhood. These representations are then fed into a temporal self-attention module which aggregates a node's state over multiple time steps, capturing long-term dependencies. This joint self-attentional design enables DySAT to effectively handle the complex, multi-faceted dynamics present in evolving graphs. Overall, the self-attention framework is easily scalable.

DyGESN: [10] DynGESN extends graph echo state networks to handle discrete-time dynamic graphs. Hence, The model treats a dynamic graph as a sequence of static snapshots but only the readout layer is trained. At each time step, the model updates node embeddings by aggregating information from neighboring nodes based on the current graph connectivity, while also incorporating leakage from previous states. This design allows DynGESN to produce updated graph representations in an online manner, significantly reducing both space and computation requirements compared to methods that store the entire interaction history.

3.1.2 Integrated Dynamic GNNs

Newer methods in this category jointly encode spatial and temporal information within unified modules, potentially capturing richer spatiotemporal dependencies. Although these approaches can model complex interactions more effectively, they are harder to parallelize and are computationally more expensive. Successful models in this class include:

Evolve GCN: [11] EvolveGCN is a dynamic graph neural network that adapts to changing graph structures by evolving its parameters over time. It integrates a spatial component based on a Graph Convolutional Network (GCN) with a temporal mechanism that updates the GCN weights using a recurrent unit (GRU or LSTM). Unlike static GCNs, the weights of each spatial convolution are determined by the recurrent unit based on the previous set of weights.

VGRN: [3] The core intuition behind VGRNN is represent each node as a probability distribution rather than a fixed vector. This probabilistic formulation allows the model to better reflect the variability in node behavior and the complex temporal changes in graph structure. VGRNN accepts a sequence of graph snapshots as input and produces a set of probabilistic node embeddings at every time step. These embeddings, being distributions, can be used to explain the confidence of our predictions. VGRNN combines graph convolutional operations with recurrent neural network dynamics. At each time step, it processes the current snapshot together with a hidden state that summarizes past information.

ROLAND: [16] The authors propose a method that can successfully adapt any static GNN method to accept a dynamic graph. ROLAND views the node embeddings produced at different layers of a static GNN as hierarchical node states and then updates these states over time as new graph snapshots arrive.
Regarding inputs, ROLAND accepts a sequence of graph snapshots—each snapshot representing the graph at a given time and outputs a set of dynamically updated node embeddings at each time step. These embeddings encapsulate both the static structural information and the temporal evolution of the graph. ROLAND extends any static GNN by adding an update module that recurrently updates the embedding using the past embedding at the same layer. This recurrent update mechanism is implemented with options like a moving average, MLP, or GRU.

4 Baseline Model - DynamicGNN

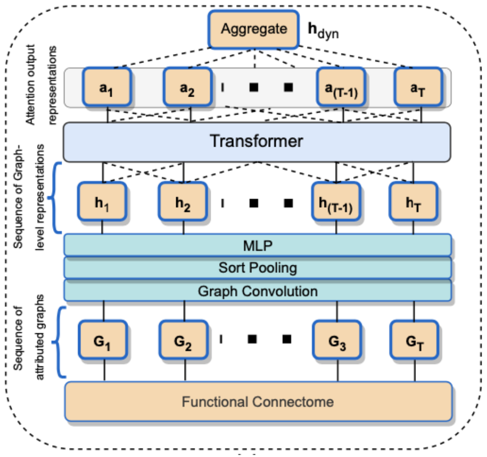

Figure 2: NeuroGraph Dynamic GNN[12]

In this section we explain the baseline model used in *NeuroGraph: Benchmarks for Graph Machine Learning in Brain Connectomics* [12]. The authors present a dynamic method that exemplifies a Stacked Dynamic GNN for snapshot temporal graphs. In this approach, the spatial and temporal units are essentially separate. In broad terms, the model first processes each static graph snapshot with the same GCN [6]. The output embeddings from all snapshots are then fed into a temporal unit based on a transformer to capture the temporal dependencies across the graph sequence.
Figure 2 illustrates the overall architecture. The spatial module extracts node or graph level features from each snapshot, while the transformer temporal module aggregates these features to produce a final dynamic representation. This design allows the spatial processing and temporal aggregation to be optimized independently.

4.0.1 Mathematical Formulation

Let the dynamic graph sequence be denoted by

$$\mathcal{G} = \{G_t = (A_t, X_t) \mid t = 1, \dots, T\},$$

where A_t is the adjacency matrix and X_t is the node feature matrix at time t.
Spatial Unit: For each snapshot G_t, a GCN is applied to extract embeddings.

$$H_t = \sigma(\mathrm{GCONV}(A_t, X_t)),$$

where σ is the activation function and $H_t \in \mathbb{R}^{n\times d}$ is the embedding matrix for snapshot t. Then, just to achieve better performance, each embedding goes through a simple MLP with dropouts and batch normalization, let all these transformations be

$$\tilde{H}_t = f(H_t),$$

Note that we have a matrix of $N_t \times d$ for each snapshot.
Temporal Unit: The sequence of embeddings $\{\tilde{H}_t\}_{t=1}^{T}$ is then processed by a transformer to capture temporal dynamics. The transformer operates on the sequence of pooled representations $\{\tilde{h}_t\}_{t=1}^{T}$, where each $\tilde{h}_t$ is derived from H_t:

$$h_t = P(H_t) \quad \text{for } t = 1, \dots, T.$$

These representations are then input to the transformer:

$$Z = \mathrm{Transformer}(h_1, h_2, \dots, h_T).$$

A standard transformer applies self-attention to the input sequence. In particular, for each element h_t, the self-attention mechanism computes:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^\top}{\sqrt{d_k}}\right) V,$$

where the query Q, key K, and value V matrices are linear projections of the input sequence. The final output Z represents the aggregated dynamic graph embedding. We can then use this embedding to make a dynamic graph classification by stacking an additional classification head.
The algorithm that will be applied later is the following, including a classification head.

Algorithm 1 Baseline - Dynamic Graph Classifier

```
1: for $t = 1, \ldots, T$ do
2:     $H_t \leftarrow \sigma(\text{GCONV}(A_t, X_t))$
3: end for
4: $\tilde{H}_t \leftarrow f(H_t)$    (MLP, dropout, batch norm)
5: $h_t \leftarrow P(\tilde{H}_t)$    (pooling)
6: $Z \leftarrow \text{Transformer}(h_1, h_2, \ldots, h_T)$
7: $z \leftarrow \text{MLP}(Z)$    (classification head) return $z$
```

5 Temporal Smoother Model

In this section, we develop a model inspired by the concept of spatial smoothing a fundamental idea in both GNNs and graph theory. Most GNN techniques are based on some form of message passing, where features are aggregated from neighboring nodes. For example, GCNs [6] perform a normalized linear combination of a node's features with those of its neighbors, GraphSAGE [4] aggregates neighborhood information using various functions such as mean or max pooling to support inductive learning, and GATs [14] utilize attention mechanisms to dynamically weight the contributions of each neighbor. These functions effectively smooth the feature space of the nodes. Our objective is to extend the concept of spatial smoothing to temporal smoothing. We want to integrate the temporal in the model by usign an attention mechanism that smooths the time changes by updating the embeddings computed at each step weighting the past contribution based on how it differs from the current embedding. We require the spatial convolution to be identical for each graph so we apply the same weights at each time stamp to avoid overparameterizing the model. We introduce an attention mechanism that regulates how much information is convolved from the past, similar to the approach proposed by Sankar et al. [13] in this feature. We use an attention mechanism becuse the intuition is that we want the model to aggregate information from the past with weights relative to how much the current state is different from the previous one, we want the model to incorporate previous information only if they differ from the current situation. This method is also partially inspired by ROLAND from You et al. [16]. As briefly described in Section 3, their paper proposes a state-of-the-art framework that adapts any static GNN method to process dynamic snapshots by adjusting the embeddings over time using ad hoc functions such as moving averages and gated recurrent units. But the main difference between our approaches it that in ROLAND, the embedding is computed just on the spatial neighborhood and then is adjusted based on the previous embedding while in our case the embedding is directly computed on both the temporal and spatial neighborhood. This difference is more pronounced if traditional GCNs [6] are used while it has different results if other kind of convolutions are used. Moreover, in our case an attention mechanism is used to control for the time information while in their paper GRUs and moving averages are employed.

5.1 Mathematical Framework and Computation

We propose a Temporal Smoother Model that extends standard spatial convolution to capture temporal dependencies in dynamic graphs. The model integrates both spatial and temporal mechanisms within each layer and is agnostic to the choice of the spatial convolution operator (e.g. GCNConv [6], GraphSAGE [4], GATConv [14], etc.).

Spatial Dimension: At each layer l and for each time step t, we apply a GNN operator on the graph snapshot to compute the intermediate (or "new") node embeddings. Using a generic graph convolution, we have:

$$\tilde{H}_t^{(l+1)} = \sigma\Big(\hat{A}_t \, H_t^{(l)} \, W_t^{(l)}\Big),$$

where $\hat{A}_t$ is the (possibly normalized) adjacency matrix at time t, $H_t^{(l)}$ denotes the node embeddings at layer l and time t, $W_t^{(l)}$ are learnable weights, and σ is an activation function.

Temporal Dimension: For each node i, given the newly computed embedding $\tilde{H}_{i,t}^{(l+1)}$ at time t and layer $l+1$ and the corresponding embedding from the previous time step $H_{i,t-1}^{(l+1)}$, we form a short sequence by stacking these two vectors:

$$x_i = \begin{bmatrix} H_{i,t-1}^{(l+1)} \\ \tilde{H}_{i,t}^{(l+1)} \end{bmatrix} \in \mathbb{R}^{2\times d},$$

where d is the embedding dimensionality at layer $l+1$.
We then apply a transformer with an integrated pooling mechanism to x_i to obtain a single fused representation:

$$Z_i = \text{Transformer}(x_i) \in \mathbb{R}^d.$$

In this design, the transformer computes attention over the two time-step embeddings and internally aggregates them (via pooling) to output one vector per node.
In matrix notation for all N nodes, we stack the embeddings as

$$X_t^{(l+1)} = \begin{bmatrix} H_{t-1}^{(l+1)} \\ \tilde{H}_t^{(l+1)} \end{bmatrix} \in \mathbb{R}^{2\times N\times d},$$

and applying the transformer with pooling gives the final smoothed embeddings:

$$H_t^{(l+1)} = \text{Transformer}(X_t^{(l+1)}) \in \mathbb{R}^{N\times d}.$$

An attention score for each node is computed as:

$$\text{Attention}(Q_l, K_l, V_l) = \text{softmax}\left(\frac{Q_l K_l^\top}{\sqrt{d_k}}\right) V_l,$$

which is encapsulated within the transformer operation. Note that attention weights are layer specific but not time spcific.
Hence, the final output is ad adjusted node embedding for each snapshot. Each embedding can be used for snapshot classification and other downstream tasks while the last embedding of the last snapshot, given that it

incorporates long term time dynamics can be used for classification of the whole sequence.

Algorithm 2 Temporal Smoother

```
 1: for $t = 0$and $l = 1, \dots, L$ do
 2:     $H_0^{l+1} \leftarrow \sigma\big(\mathrm{GCONV}(A_0, H_0^l)\big)$
 3: end for
 4: for $t = 1, \dots, T$ and $l = 1, \dots, L$ do
 5:     $\tilde{H}_t^{l+1} \leftarrow \sigma\big(\mathrm{GCONV}(A_t, H_t^l)\big)$
 6:     for each node $i$ do
 7:         $x_i \leftarrow \begin{bmatrix} H_{i,t-1}^l \\ \tilde{H}_{i,t}^l \end{bmatrix}$
 8:         $Z_i \leftarrow \mathrm{Transformer}(x_i)$
 9:     end for
10:     $H_t^l \leftarrow \{Z_i\}_{i=1}^N$
11: end for
12: $z \leftarrow \mathrm{MLP}\big(\mathrm{Pool}(H_T^L)\big)$ return $z$
```

6 Data

Our data comes from the domain of brain connectomics, specifically from NeuroGraph [12], a benchmark widely used for machine learning methods in this context. This dataset is derived from the Human Connectome Project, which leverages fMRI data to create functional connectivity (FC) matrices that define brain networks.

As a brief introduction, the authors begin with a time series of functional MRI (fMRI) data, recorded either at rest or while subjects perform various activities. This data captures fluctuations in the blood-oxygen-level-dependent (BOLD) signal, which reflects neural activity in different brain regions over time. The BOLD signal is extracted from predefined regions of interest (ROIs), which are anatomical or functional subdivisions of the brain. The time series for each ROI represents the average BOLD signal intensity across all voxels within that region.

To compute the FC matrix, the temporal correlation between the BOLD signals of different ROIs is calculated. This is typically done using Pearson's correlation coefficient, which measures the linear dependency between pairs of

time series. The resulting matrix is symmetric, with each element indicating the strength of functional connectivity between two brain regions.
The dataset provides both static and dynamic representations of brain connectivity. For this project, we focus on dynamic datasets, specifically two resting-state datasets:

- **HCP-Gender**: Aims to classify each dynamic graph sequence as Male or Female.
- **HCP-Age**: Aims to classify each dynamic graph sequence into predefined age groups.

The characteristics of these datasets are summarized in Table 1.

Dataset	$\|G\|$	$\|N\|_{\text{avg}}$	$\|E\|_{\text{avg}}$	$d_{\max}$	d_{avg}	K_{avg}	Node Feat.	#Classes
DynHCP-Gender	1080	100	874.88	53	8.749	0.439	100	2
DynHCP-Age	1067	100	875.42	53	8.754	0.439	100	3

Table 1: Dataset Statistics

Applying Graph Neural Networks (GNNs) to brain connectomics data has become the dominant paradigm in this field, as they have consistently outperformed other methods, at least in the static setting [7]. However, in the dynamic setting, results have not matched those of static models, as widely discussed in [5]. Incorporating the temporal dimension has often led to comparable or even lower performance than treating the same dataset as a collection of independent static graphs.
We find it counterintuitive that adding a temporal dimension does not lead to improved predictive performance, given the inherently dynamic nature of brain activity. This apparent discrepancy is the primary motivation behind our choice of dataset and research focus. We believe there is still significant room for improvement and for conducting a meaningful comparative analysis to better understand the role of temporal information in brain connectomics.

7 Results

In this section we describe the results of our Temporal Smoother model as well as the training, tuning, and testing strategy. As extensively described in

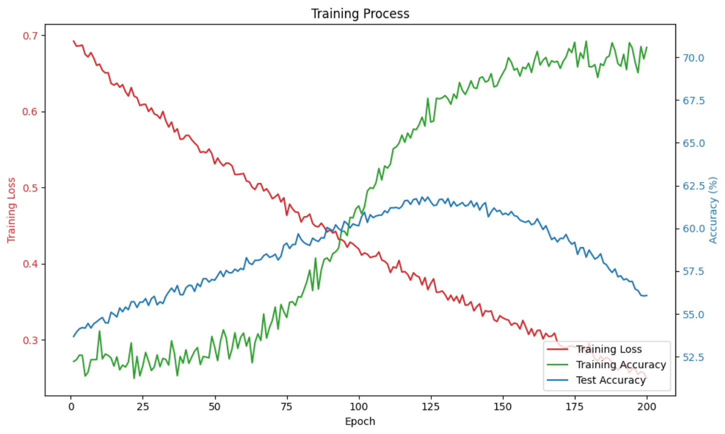

Figure 3: Training Process Temporal Smoother-GCN on DynGender

Section 6, our dataset consists of two Dynamic Graph classification problems; DynGender is a binary classification problem while DynAge is a classification into 3 classes. Both datasets consist of Dynamic Graphs, i.e., a series of snapshots of the same graph at different moments in time. Hence, the Temporal Smoother and Baseline models take as input a series of temporal graphs and, through different mechanisms extensively described above, output an embedding or a series of embeddings on which an MLP has been applied in order to output logits for the classification. We combined Temporal Smoother with different spatial convolutions, GCN[6], GraphSAGE [4] and GAT [14] to show its adaptivity. Given the logits for the classes, we compute a classic cross entropy loss function:

$$\mathcal{L} = -\sum_{c} y_c \log(\hat{y}_c)$$

and we backpropagate using the Adam optimizer. Hyperparameters have been adjusted using grid search using testing accuracy as the metric.
The behavior of the loss function, as well as the training and testing accuracy (for Gender Classification using GCN), can be observed in Figure 3. Maximum testing performance was achieved at 124 of epochs, where testing accuracy was 62.04% and training accuracy was 67.51%.
Comparing the results of Temporal Smoother, we can observe that it closely matches baseline performance in most cases and outperforms in two categories regarding the DynAge dataset. Given that that the two datasets feature 2 and 3 classes all our results are well above chance benchmark.

Dataset/Model	Baseline			TempSmooth		
	GCN	GraphSAGE	GAT	GCN	GraphSAGE	GAT
DynGender	62.04	66.20	67.13	61.73	63.42	64.32
DynAge	42.99	40.65	44.39	44.19	44.35	43.99

Table 2: Accuracy (%)

8 Conclusion

In this work, we introduced a novel framework for dynamic graph modeling in brain connectomics that leverages a temporal smoothing mechanism to fuse spatial and temporal information. By integrating a transformer-based attention module with standard graph convolutions, our approach is able to capture long-term temporal dependencies. Experimental results on NeuroGraph benchmarks indicate that our Temporal Smoother Model achieves performance competitive with, and in some cases superior to, baseline models.

Nevertheless, the model still presents significant computational challenges. Given the nature of the attention process, optimization took almost twice as long as optimizing the benchmark model for similar results, so there is still room for improvement in a full comparison. Moreover, the results achieved on the NeuroGraph database show that it is difficult to capture temporal dynamics in brain connectomics, which challenges the justification for using temporal graphs instead of static ones, which are more reliable and less computationally expensive. Moreover, given the nature of the model regularization techniques such as batch normalization and dropouts, could have been applied. Even though the model's results simply match the baseline without showing particular improvements, they support the claim that the concept of temporal smoothing should be further researched and understood for use in temporal analysis of many different combinatorial structures. Future directions could start with applying this mechanism to datasets in other domains and for different types of tasks; in this paper, we focused on sequence classification, but the model could be easily modified to address snapshot and node classification as well.

References

[1] Claudio D. T. Barros, Matheus R. F. Mendonça, Alex B. Vieira, and Artur Ziviani. A survey on embedding dynamic graphs, 2021.

[2] Shubham Gupta and Srikanta Bedathur. A survey on temporal graph representation learning and generative modeling, 2022.

[3] Ehsan Hajiramezanali, Arman Hasanzadeh, Nick Duffield, Krishna R Narayanan, Mingyuan Zhou, and Xiaoning Qian. Variational graph recurrent neural networks, 2020.

[4] William L. Hamilton, Rex Ying, and Jure Leskovec. Inductive representation learning on large graphs, 2018.

[5] Byung-Hoon Kim, Jong Chul Ye, and Jae-Jin Kim. Learning dynamic graph representation of brain connectome with spatio-temporal attention, 2021.

[6] Thomas N. Kipf and Max Welling. Semi-supervised classification with graph convolutional networks, 2017.

[7] Xiaoxiao Li, Yuan Zhou, Nicha Dvornek, Muhan Zhang, Siyuan Gao, Juntang Zhuang, Dustin Scheinost, Lawrence H. Staib, Pamela Ventola, and James S. Duncan. Braingnn: Interpretable brain graph neural network for fmri analysis. *Medical Image Analysis*, 74:102233, December 2021. Epub 2021 Sep 12.

[8] Antonio Longa, Veronica Lachi, Gabriele Santin, Monica Bianchini, Bruno Lepri, Pietro Lio, Franco Scarselli, and Andrea Passerini. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities, 2023.

[9] Franco Manessi, Alessandro Rozza, and Mario Manzo. Dynamic graph convolutional networks. *Pattern Recognition*, 97:107000, 2020.

[10] Alessio Micheli and Domenico Tortorella. Discrete-time dynamic graph echo state networks. *Neurocomputing*, 496:85–95, 2022.

[11] Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao B. Schardl, and Charles E.

Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs, 2019.

[12] Anwar Said, Roza G. Bayrak, Tyler Derr, Mudassir Shabbir, Daniel Moyer, Catie Chang, and Xenofon Koutsoukos. Neurograph: Benchmarks for graph machine learning in brain connectomics, 2024.

[13] Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dynamic graph representation learning via self-attention networks, 2019.

[14] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks, 2018.

[15] Guotong Xue, Ming Zhong, Jianxin Li, Jia Chen, Chengshuai Zhai, and Ruochen Kong. Dynamic network embedding survey, 2021.

[16] Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: Graph learning framework for dynamic graphs, 2022.

[17] Yanping Zheng, Lu Yi, and Zhewei Wei. A survey of dynamic graph neural networks, 2024.