




# Exploring Gated Graph Sequence Neural Networks for Predicting Next Process Activities

Sven Weinzierl<sup>(✉)</sup> 

Institute of Information Systems, Friedrich-Alexander-Universität Erlangen-Nürnberg,  
Fürther Straße 248, 90429 Nürnberg, Germany  
sven.weinzierl@fau.de

**Abstract.** A current trend in predictive business process monitoring is to construct predictive models using deep neural networks (DNNs), especially long short-term memory neural networks, convolutional neural networks, or multi-layer perceptron neural networks. While these DNN types typically require data defined on the Euclidean space (e.g., grids), graph neural networks (GNNs), a relatively new type of DNNs, can compute data defined on the non-Euclidean space (e.g., graphs). Because GNNs can directly compute graph-oriented data inputs, generally structured into nodes and edges, they can explicitly model event relationships. This paper investigates gated graph sequence neural networks (GGNNs) for the next activity prediction. First results with two real-life event logs show that GGNNs can outperform traditional DNNs regarding predictive quality, especially if nodes of an input graph are assumed as events, and the graph's adjacency matrix only describes event relationships of a process instance prefix.

**Keywords:** Predictive business process monitoring · Graph neural networks · Deep learning · Machine learning

## 1 Introduction

The highly volatile and uncertain digital economy increases the pressure on organizations to proactively manage their business processes [16]. Consequently, predictive business process monitoring (PBPM) is gaining momentum in business process management (BPM) [2]. It provides a set of techniques to predict properties of operational business processes such as future process behavior (e.g., next activities) or process outcomes (e.g., a process performance indicator). Predictions from these techniques enable process stakeholders to make decisions that can improve the efficiency of operational business processes. However, this assumes correct predictions, which cannot yet be fully achieved by existing PBPM techniques, especially in real business processes [11, 18, 24].

Most recent PBPM techniques construct predictive models from historical event log data using machine learning (ML) algorithms. An ML algorithm automatically discovers structures (i.e., patterns) in data and captures those within a predictive model [1]. In the context of ML, deep learning (DL) has “turned out to be very good at discovering the intricate structures in high-dimensional data and is therefore applicable to many domains” [7, p. 436]. This also applies to PBPM, where DL, especially deep

neural networks (DNNs), have shown to outperform techniques relying on traditional ML algorithms (e.g., regulated probabilistic automata [12] or extreme gradient boosting [23]).

A current trend in PBPM is to use DNNs such as long short-term memory neural networks (LSTM-NNs), convolutional neural networks (CNNs), or multi-layer perceptron neural networks (MLP-NNs) [17]. However, these DNN types typically require data defined on the Euclidean space (e.g., grids) for model training and application [25]. Therefore, a business process instance is represented by a feature vector or matrix, and features of these structures describe event relationships. Consequently, DNNs of current PBPM techniques are not explicitly aware of event relationships, i.e., they do not explicitly model them. Naturally, this makes it difficult for current DNN-based PBPM techniques to increase the predictive quality further because important information (i.e., domain knowledge) of process instances may not be considered. Consequently, they cannot identify some crucial intricate structures in the process data.

A relatively new group of DNNs are graph neural networks (GNNs) [19] that can compute data defined on the non-Euclidean space (e.g., graphs) directly [25]. Here, the structure of input graphs can be matched directly onto the topology of the GNNs, and direct inferences can be made between the network nodes and nodes of the input graphs. In PBPM, first works [10, 15, 20, 22] have proven GNNs to be useful because the control-flow of process instances can intuitively be represented as graphs, and relationships between process activities can explicitly be modeled. However, these works present either an approach designed for the process outcome prediction or apply graph convolutional neural networks (GCNs) [5] for the next activity prediction. GCNs belong to another type of GNNs computing input graphs from a spatial perspective.

This paper investigates gated graph sequence neural networks (GGNNs) [9, 19] for the next activity prediction. This type of GNNs is designed for sequential graphs and integrates a gated recurrent unit (GRU) [3] that explicitly considers the temporal aspect of sequences. Since the high expressiveness of GGNNs allows us to represent the input graphs in different ways, we investigate three forms of representing these.

The remainder of this paper is structured as follows: Sect. 2 presents preliminaries and related work on PBPM techniques using GNNs and reveals the research gap of investigating GGNNs for the next activity prediction. Section 3 describes the three investigated representation forms of input graphs and presents the developed GGNN architecture. While Sect. 5 discusses the results and limitations of this paper, Sect. 6 concludes it with a summary and points out future research.

## 2 Background

### 2.1 Preliminaries

PBPM techniques receive as input event logs. An event log is structured into traces, and in turn, a trace is structured into events.

**Definition 1 (Event, Trace, Event Log).** *An event is a tuple  $e = (c, a, t, d_1, \dots, d_n)$ , where  $c$  is the process instance or case id,  $a$  is the activity,  $t$  is the timestamp, and  $d_n$  is the  $n^{\text{th}}$  context feature assigned to the event  $e$ . A trace is a non-empty sequence*

$\sigma = \langle e_1, \dots, e_{|\sigma|} \rangle$  of events such that  $\forall i, j \in \{1, \dots, |\sigma|\}$   $e_i.c = e_j.c$ , with  $i > j$ , where  $.$  denotes the process instance id  $c$  of the event  $e_i$  or  $e_j$ . A trace referring to a process instance can also be represented as a sequence of vectors  $\langle \mathbf{x}_1, \dots, \mathbf{x}_{|\sigma|} \rangle$ , where  $\mathbf{x} \in \mathbb{R}^m$  is a vector with size  $m$ . A vector can store all event information or a part of it (e.g., information belonging to the event’s activity and its  $n^{\text{th}}$  context feature). An event log  $\mathcal{L}$  is a set of traces  $\{\sigma_1, \dots, \sigma_{|\mathcal{L}|}\}$ .

The next activities are predicted based on prefixes of a trace. Labels are the next activities and represent the learning target.

**Definition 2 (Prefix, Label).** A prefix is a non-empty sub-sequence of a trace  $\sigma = \langle \mathbf{x}_1, \dots, \mathbf{x}_k, \dots, \mathbf{x}_{|\sigma|} \rangle$  with a length  $k$ . It is defined as  $f_{pre}^{(k)}(\sigma) = \langle \mathbf{x}_1, \dots, \mathbf{x}_k \rangle$ , with  $0 < k < |\sigma|$ . For instance, possible prefixes for  $\sigma_1 = \langle \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \rangle$  are  $\langle \mathbf{x}_1 \rangle$  or  $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ . A label is an annotation for a prefix (i.e., the next activity) of a trace  $\sigma = \langle \mathbf{x}_1, \dots, \mathbf{x}_k, \dots, \mathbf{x}_{|\sigma|} \rangle$  with a length  $k$ . It is defined as  $f_l^{(k)}(\sigma) = \mathbf{a}_{k+1}$ , with  $0 < k < |\sigma|$ , where  $\mathbf{a}_{k+1}$  includes features describing the activity of the next event  $\mathbf{x}_{k+1}$ . For instance, possible labels for  $\sigma_1 = \langle \mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3 \rangle$  are  $\mathbf{a}_2$  or  $\mathbf{a}_3$  only storing information of the next events’ activities.<sup>1</sup>

## 2.2 Related Work

GNNs have been used in four PBPM works so far. Philipp et al. [15] developed DNNs with two graph convolutional [5] layers for predicting real-valued process outcomes based on finished process instances. The graph layers receive an adjacency matrix that captures activity relationships of the entire event log. A graph node is assumed as an activity. It is described by a feature representing the sum of how often the activity was performed.

Harl et al. [10] developed DNNs with a gated graph [9] layer for binary process outcome prediction based on finished process instances. The authors constructed an adjacency matrix per process instance that captures the activity relationships of the respective process instance’s activities. Even though they considered a node as an activity, such as Philipp et al. [15], they described it by the one-hot encoding of the activity and model edges between nodes to describe their relation type. Finally, they extracted relevance values per node from *softmax* attention to explain the GNN models’ predictions. Stierle et al. [20] presented an extension of Harl et al. [10]’s approach. They aggregated the relevance values across graphs to create a process model in which each activity is colored depending on its aggregated relevance value.

Venugopal et al. [22] developed DNNs with two graph convolutional layers [5] for the next activity and next timestamp prediction based on process instance prefixes. Like Philipp et al. [15], they constructed adjacency matrices based on the entire event log. Additionally, they investigated different approaches to normalize the matrices’ values. A node is assumed as an activity and described by temporal features constructed from the control-flow data of the event logs.

Against this background, we are the first to investigate the use of GGNNs for the next activity prediction that can learn sequence representations from graph data.

<sup>1</sup> Note: The functions  $f_{pre}(\cdot)$  and  $f_l(\cdot)$  can be applied to process instances  $\sigma$  with a log-oriented or vector-oriented representation.

### 3 Gated Graph Sequence NNs for Next Activity Prediction

#### 3.1 Three Representation Forms of Process Instance Graphs

GNNs, such as GGNNs, require inputs with a graph-oriented representation. Therefore, we transform prefixes created from process instances of the event log  $\mathcal{L}$  into process instance graphs. However, before we present the three investigated representation forms of process instance graphs, we define the terms *graph*, *node*, and *edge*.

**Definition 3 (Graph, Node, Edge).** A graph  $g$  is a two-element tuple  $(V, E)$ , where  $V$  is the set of nodes and  $E$  is the set of edges. A node  $v \in V$  is represented by a single value, and a set of node features can be assigned to it. An edge  $\hat{e}$  is a tuple  $(v, v') \in V \times V$ , and a set of edge features can be assigned to it.

GGNNs, as used here, cannot directly compute forward and backward propagation based on (raw) graphs. Therefore, we transform graphs into *instance graphs*.

**Definition 4 (Instance Graph, Adjacency Matrix, Node Matrix, and Edge Matrix).** An instance graph  $\psi$  is a three-element tuple  $(\mathbf{A}, \mathbf{V}, \text{and } \mathbf{E})$ .  $\mathbf{A}$  is an adjacency matrix storing which nodes of the graph are connected by an edge and lies in  $\mathbb{R}^{|V| \times |V|}$ , where  $|V|$  is the number of nodes of the graph.  $\mathbf{V}$  is a node matrix storing features that describe the graph's nodes and lies in  $\mathbb{R}^{|V| \times q}$ , where  $q$  is the number of node features.  $\mathbf{E}$  is an edge matrix storing features that describe the edges of the graph and lies in  $\mathbb{R}^{|V| \times p}$ , where  $p$  refers to the number of edge features, i.e., the source node, the target node, and features describing the edge.<sup>2</sup>

A process instance graph is an instance graph that represents a prefix of a process instance or an entire process instance. Because nodes and edges of a process instance graph can be interpreted in different ways, there are different forms of how a process instance graph can represent a prefix or an entire process instance. In this paper, we investigate three conceptually different representation forms of process instance graphs. The first representation form (*RF1*) assumes a node as an event, and the edges are prefix-based. The second representation form (*RF2*) assumes a node as an activity, and the edges are prefix-based. The third representation form (*RF3*) assumes a node as an activity, and edges are event-log-based. In the following, we present the three representation forms based on the exemplary prefix  $f_{pre}^{(4)}(\sigma_1)$ <sup>3</sup>:

$$f_{pre}^{(4)}(\sigma_1) = \langle (1, \text{A}, 2021-01-01\text{T}00:10:00, \text{Le}), \\ (1, \text{B}, 2021-01-02\text{T}00:10:00, \text{Hu}), \\ (1, \text{B}, 2021-01-03\text{T}00:10:00, \text{Hu}), \\ (1, \text{C}, 2021-01-04\text{T}00:10:00, \text{Le}) \rangle. \quad (1)$$

<sup>2</sup> Note:  $\mathbf{E}$  can also be described as a third-order tensor  $\mathbf{E}$ , that lies in  $\mathbb{R}^{|V| \times |V| \times p}$ . In this case,  $\mathbf{E}$ 's three dimensions refer to the source node, target node, and edge features.

<sup>3</sup> Note: In Eq. (1), the last element per event is a context feature indicating to the employee who performed the event's activity.

**RF1 - A Node is an Event and Edges are Prefix-Based:** The first representation form of a process instance graph assumes (time-ordered) events of a prefix as graph nodes and considers edges between the prefix’s events. The process instance graph  $\psi_{\varepsilon}^{RF1}$  for the prefix  $f_{pre}^{(4)}(\sigma_1) = \varepsilon$  is shown in Eq. (2).<sup>4</sup>

$$\psi_{\varepsilon}^{RF1} = \left( \begin{array}{c} \begin{bmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 \end{bmatrix}, \begin{bmatrix} A & 0.0 & 0.0 & 0.42 & 5.0 & Le \\ B & 1.0 & 1.0 & 0.42 & 6.0 & Hu \\ B & 1.0 & 2.0 & 0.42 & 7.0 & Hu \\ C & 1.0 & 3.0 & 0.42 & 1.0 & Le \end{bmatrix}, \begin{bmatrix} 1 & 2 & Forward \\ 2 & 3 & Repeat \\ 3 & 4 & Forward \end{bmatrix} \end{array} \right) \quad (2)$$

The first matrix of the process instance graph  $\psi_{\varepsilon}^{RF1}$  is the adjacency matrix.<sup>5</sup> It stores the connection between events. For example, the first vector  $[0.0, 1.0, 0.0, 0.0]$  indicates that an edge goes from the first event to the second event of the prefix. The second matrix is the node matrix. It stores all information for a node. In this matrix, the first feature is the activity, the next four are temporal features calculated based on the events’ timestamps, and the last feature is the resource feature. We add the four temporal features to the node features proposed by Tax et al. [21]; that are, (1) the time since the previous event in the process instance, (2) the time since the process instance started, (3) the time since midnight, and (4) the day of the week for the event. Additionally, we transform the values of the first three temporal features from seconds into days to allow smooth learning of the internal model parameters; otherwise, values of these features can be very high, and therefore, negatively affect the learning of the internal model parameters. The third matrix is the edge matrix. It stores the id of the source event, the id of the target node, and the type of edge defined by the source and target node. Regarding edge types, we differ between (1) *Repeat* (activity of a target event is equal to an activity of a source event), (2) *Backward* (activity of a target event has been observed in a previous event of the current prefix), and (3) *Forward* (activity of a target event has not been observed in previous events of the current prefix).

**RF2 - A Node is an Activity and Edges are Prefix-Based:** The second representation form of a process instance graph assumes activities of a prefix as graph nodes and considers edges between the performed activities of the prefix. The process instance graph  $\psi_{\varepsilon}^{RF2}$  for the prefix  $f_{pre}^{(4)}(\sigma_1)$  is shown in Eq. (3).

$$\psi_{\varepsilon}^{RF2} = \left( \begin{array}{c} \begin{bmatrix} 0.0 & 1.0 & 0.0 \\ 0.0 & 1.0 & 1.0 \\ 0.0 & 0.0 & 0.0 \end{bmatrix}, \begin{bmatrix} A & 1.0 & 0.0 & 0.0 & 0.42 & 5.0 & Le \\ B & 2.0 & 1.0 & 1.0 & 0.42 & 6.0 & Hu \\ C & 1.0 & 1.0 & 3.0 & 0.42 & 1.0 & Le \end{bmatrix}, \begin{bmatrix} A & B & Forward \\ B & B & Repeat \\ B & C & Forward \end{bmatrix} \end{array} \right) \quad (3)$$

Like in  $\psi_{\varepsilon}^{RF1}$ , the first matrix of the process instance graph  $\psi_{\varepsilon}^{RF2}$  is the adjacency matrix. In contrast to  $\psi_{\varepsilon}^{RF1}$ , recurring activities cannot be marked because a single node

<sup>4</sup> Note: For better readability, we show the raw values of the categorical features (i.e., activity, resource, and edge type) for the three representation forms and not the features’ one-hot encoded values.

<sup>5</sup> Note: We normalize the adjacency matrix for each prefix of each representation form by  $\mathbf{A} \leftarrow \mathbf{D}^{-1/2} \mathbf{A} \mathbf{D}^{-1/2}$ .

represents an activity. Consequently, this adjacency matrix is smaller than the adjacency matrix of  $\psi_{\mathcal{E}}^{RF1}$ . Further, as Philipp et al. [15] suggested, we store the number of how often an activity has been performed in the second column of this representation form's node matrix. The other features are equal to the first representation form. However, since activities are nodes and can be performed more than once in a sequence, we follow the work of Venugopal et al. [22] and only store the feature values of an activity's last occurrence. The edge matrix stores the source activity, the target activity, and the edge type. Concerning edges, we differ again between the types (1) *Repeat*, (2) *Backward*, and (3) *Forward*.

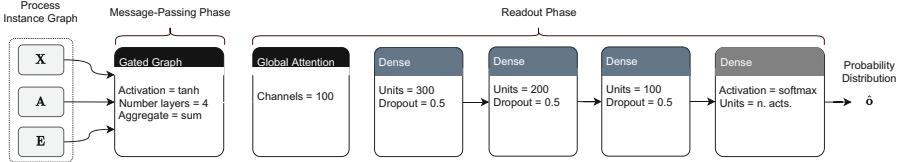
**RF3 - A Node is an Activity and Edges are Event-Log-Based:** The third representation form of a process instance graph assumes activities as graph nodes and considers edges between activities of the entire event log's process instances. The process instance graph  $\psi_{\mathcal{E}}^{RF3}$  for the prefix  $f_{pre}^{(4)}(\sigma_1)$  is shown in Eq. (4).

$$\psi_{\mathcal{E}}^{RF3} = \left( \begin{array}{c} \begin{bmatrix} 594.0 & 515.0 & 0.0 & 355.0 & 0.0 & 0.0 \\ 4.0 & 960.0 & 209.0 & 803.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 922.0 & 751.0 & 0.0 & 8.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 515.0 & 0.0 & 0.0 & 235.0 & 7.0 & 0.0 \\ 2.0 & 0.0 & 33.0 & 57.0 & 32 & 0.0 \end{bmatrix}, \begin{bmatrix} A & \dots & Le \\ B & \dots & Hu \\ C & \dots & Le \end{bmatrix}, \begin{bmatrix} A & B & Forward \\ B & B & Repeat \\ B & C & Forward \end{bmatrix} \end{array} \right) \quad (4)$$

The first matrix is the adjacency matrix, representing the edges between activities of the entire event log. To determine the edges, we create a directly-follows graph (DFG) in its native variant, counting the number of directly follows occurrences in all process instances of the event log. In contrast to the previous two representation forms, the adjacency matrix is not prefix-based but event-log-based. Therefore, each process instance graph has assigned the same adjacency matrix, representing the DFG created based on the entire event log. The node and edge matrix are equally constructed like in the second representation form.

### 3.2 Learning Gated Graph Sequence Neural Network Models

A GGNN model's architecture can be structured into two phases, a message-passing and a readout phase [4] (see Fig. 1).



**Fig. 1.** Architecture of the gated graph sequence neural network model.

The message-passing phase receives as input a set of process instance graphs  $\{(\mathbf{A}_1, \mathbf{V}_1, \mathbf{E}_1), (\mathbf{A}_{\dots}, \mathbf{V}_{\dots}, \mathbf{E}_{\dots}), (\mathbf{A}_n, \mathbf{V}_n, \mathbf{E}_n)\}$ , where  $n$  is the number of process instance

graphs. This phase is realized by a gated graph layer, calculating abstract node representations  $\mathbf{h}_v^{(t+1)}$  for each node  $v$  of a process instance graph in two steps. First, it calculates for each node  $v$  messages  $\mathbf{m}_v^{(t+1)}$ , expressing interactions between nodes, through the message function  $f_{msg}^{(t)}(\cdot)$ , as shown in Eq. (5).

$$\mathbf{m}_v^{(t+1)} = \sum_{v' \in f_{nbr}(v)} f_{msg}^{(t)}(\mathbf{h}_v^{(t)}, \mathbf{h}_{v'}^{(t)}, \mathbf{e}_{(v,v')}) = \sum_{v' \in f_{nbr}(v)} \mathbf{A}e_{(v,v')} \mathbf{h}_{v'}^{(t)}. \quad (5)$$

In Eq. (5), the function  $f_{nbr(\cdot)}$  returns for a node  $v$  its neighboring nodes  $v'$ . For each node  $v$ , the interactions to neighboring nodes are aggregated by calculating the sum of them. Consequently, the messages for each node  $v$  are  $\mathbf{m}_v^{(t+1)}$ . Second, this phase calculates the new node representation  $\mathbf{h}_v^{(t+1)}$  based on the messages  $\mathbf{m}_v^{(t+1)}$  and the previous node representation  $\mathbf{h}_v^{(t)}$  by applying the node update function  $f_{gru}(\cdot)$ , as shown in Eq. (6).

$$\mathbf{h}_v^{(t+1)} = f_{gru}(\mathbf{h}_v^{(t)}, \mathbf{m}_v^{(t+1)}). \quad (6)$$

In Eq. (6), the function  $f_{gru}(\cdot)$  is a GRU [3], with a  $\tanh$  activation. The number of (sub-)layers within the gated graph layer is four and refers to the number of iterations with the GRU. The number of GRU iterations and the number of time steps (i.e., the sequence length) determine the number of updates. In each update, the node representations for each node of a process instance graph are updated roughly simultaneously.

The readout phase receives as input the final node representations  $\mathbf{h}_v^{(T)}$  from the message-passing phase. It maps these to a probability distribution vector  $\hat{\mathbf{o}}$  through the readout function  $f_{rea}(\cdot)$ , as shown in Eq. (7). The vector  $\hat{\mathbf{o}}$  stores the probabilities of the next activities for a process instance graph. The readout function  $f_{rea}(\cdot)$  is realized by a global attention layer [9], followed by four dense layers. The first three dense layers include 300, 200, and 100 neurons, and the  $\tanh$  activation function is applied. A dropout with a ratio of 0.5 is applied to the outgoing connections of these layers to avoid overfitting. The last dense layer's number of units refers to the number of activities, and a  $\text{softmax}$  activation function is applied to calculate the vector  $\hat{\mathbf{o}}$ .

$$\hat{\mathbf{o}} = f_{rea}(\{\mathbf{h}_v^{(T)} | v \in G\}). \quad (7)$$

We update the internal parameters of the GGNN models batch-wise and over 100 epochs. Per epoch, the process instance graphs are partitioned into batches, including 32 process instance graphs. For each process instance graph of a batch, the *categorical cross-entropy* (loss function) calculates the loss between its probability distribution  $\hat{\mathbf{o}}$  and the assigned ground label  $\mathbf{y}$ . Subsequently, the cost function calculates the average of the loss function outputs over the batch's process instance graphs. Then, the gradient descent algorithm *ADAM* calculates the derivation of the cost function's output and updates the internal parameters of the model. After completing the last epoch, the internal model parameters are adjusted.

## 4 Evaluation

### 4.1 Event Logs

In our evaluation, we used the following two real-life event logs (see Table 1 for the event logs’ characteristics):

**bpi2012w**<sup>6</sup> originates from a Dutch Financial Institute. It includes three sub-processes: A (states of application), O (states of the offer belonging to the application), and W (states of work items belonging to the application). We only considered the sub-process W because work items are preformed by humans. As a context feature, we included the *resource* feature for the evaluation.

**sepsis**<sup>7</sup> stores sepsis cases. One case represents patient-related activities in a hospital. For instance, receiving treatments or taking measurements. We included the context feature *org\_group* for the evaluation.

**Table 1.** Characteristics of event logs used.

Event log	# Instances	# Inst. variants	# Events	# Act.	# Events per Inst.*	# Context attr.’s values
bpi2012w	9,658	2,263	72,413	6	[1, 74, 7]	60
sepsis	1,050	846	15,214	16	[3, 185, 14]	26

\*[min, max, mean]

### 4.2 Procedure

**Event Logs:** We only considered traces of the event logs with a size greater than two, and we created prefixes from the event log’s traces with a minimum size of two for model training and application. This is necessary because prefixes with a smaller size do not provide sufficient data for predicting the next activities [21]. Further, we added an artificial *End* event after each trace. This is a typical pre-processing step in the next activity prediction because most event logs do not include such an event, and we also want to predict the end of traces [17].

**Validation Strategy:** We performed a stratified ten-fold cross-validation with random shuffling to keep the bias and variance of the models low [6]. We used the last 20% of each training set as the validation set. We applied early stopping to the validation set after ten epochs to avoid overfitting. For a fair comparison, we performed this validation strategy for the GGNN and the baseline models.

**ML Metrics:** We calculated ML metrics in two ways. First, we calculated the weighted *Precision*, *Recall*, and *F1-Score* per iteration of the ten-fold cross-validation for both event logs. Then, we calculated the average and the standard deviation over the ten values per metric. A definition of these ML metrics can be found in other PBPM works, such as Mehdiyev et al. [12]. Second, we calculated the averaged *Recall* and *Precision* over the ten folds of the cross-validation prefix-wise for the bpi2012w event log. We

<sup>6</sup> [https://data.4tu.nl/articles/dataset/BPI\\_Challenge\\_2012/12689204/1](https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204/1).

<sup>7</sup> [https://data.4tu.nl/articles/dataset/Sepsis\\_Cases\\_-\\_Event\\_Log/12707639](https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639).



start the range of prefixes with size two, increment the prefix sizes by step size one, and end the range with size 15. This prefix range allows us to assess how well the DNN models perform in the course of process instances.

**Baselines:** We benchmarked the GNN models with an LSTM-NN, an MLP-NN, and a CNN model. For the baselines, we describe an event of a prefix by a one-hot encoded activity, four temporal features (see Sect. 3.1), and a one-hot encoded resource value. The LSTM-NN model included one hidden LSTM layer with an internal cell element size of 100. We applied a dropout with a ratio of 0.3 to this layer’s inputs to avoid overfitting. The last layer of the LSTM-NN model and the other baselines is equal to the GGNN models’ final layer, calculating a probability distribution over the next activities. The multi-layer perceptron neural network (MLP-NN) included three hidden dense layers with 100 neurons. After each hidden layer, a dropout with a ratio of 0.5 is applied to the layer’s outgoing connections to avoid overfitting. The convolutional neural network (CNN) included six hidden layers. The first five layers were one-dimensional convolutional layers with *max pooling*. For each convolutional layer, we set the number of filters to 128, the kernel size to 10, the padding to *same*, the strides to 1, and the activation function to *relu*. The last hidden layer was a dense layer with 100 neurons and a *relu* activation. Like for the GGNN models, we applied *ADAM* with standard values and *categorical cross-entropy* loss to optimize the baseline models’ internal parameters.

### 4.3 Results

$GGNN_{RF1}$  achieves the highest *Precision*, *Recall*, and *F1-Score* for both event logs (see Table 2). For the bpi2012w event log,  $GGNN_{RF2}$  outperforms  $GGNN_{RF3}$  and all baselines regarding each metric, except the *Precision* of *MLP-NN*.  $GGNN_{RF3}$  also achieves a higher *Recall*, and *F1-Score* than the baselines for bpi2012w, and its *Precision* is lower than for *MLP-NN*. For the sepsis event log,  $GGNN_{RF2}$  has a higher *Precision* than  $GGNN_{RF3}$ . While  $GGNN_{RF2}$  outperforms  $GGNN_{RF3}$ , *MLP-NN*, and *CNN* regarding *Recall*, it achieves a higher *F1-Score* than *CNN* and  $GGNN_{RF3}$ . For sepsis,  $GGNN_{RF3}$  achieves the lowest predictive quality regarding all metrics.

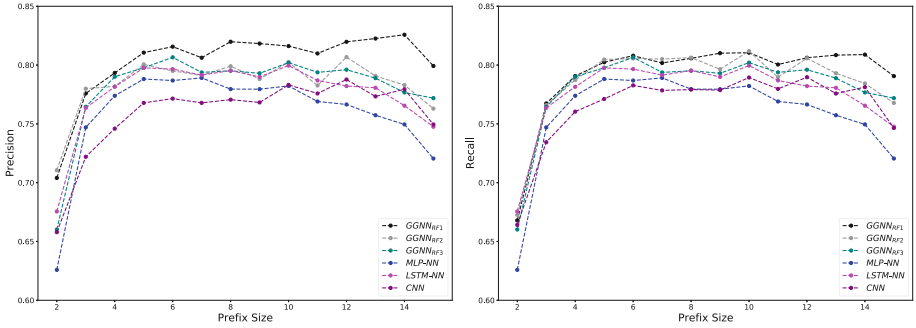
Looking at the *Precision* and *Recall* values (see Fig. 2),  $GGNN_{RF1}$  has the highest values for most of the prefix sizes. In contrast, *Precision* and *Recall* values for  $GGNN_{RF2}$  and  $GGNN_{RF3}$  are only higher than the baselines for certain prefix sizes.

## 5 Discussion

**Findings:** We derived two main findings from our results. First, *RF1*, where each event is represented as a graph node, works better for GGNNs than the other two representation forms, assuming activities as graph nodes (see Table 2 and Fig. 2). This effect can be attributed to the fact that *RF1* stores all event information, even in the case of recurrent activities in process instances. Second, GGNNs with all representation forms perform better than the baselines for bpi2012w as for sepsis (see Table 2). We suppose that GGNNs can better learn from process instance graphs with a homogeneous structure than with a diffuse structure. Consequently, GGNNs, as used in this paper, may

**Table 2.** Predictive quality of the GGNN and baseline models (average over ten folds).

Event log	DL model	Precision (w)	Recall (w)	F1-Score (w)
bpi2012w	$GGNN_{RF1}$	<b>0.7941</b> ( $\pm 0.0056$ )	<b>0.7834</b> ( $\pm 0.0046$ )	<b>0.7817</b> ( $\pm 0.0048$ )
	$GGNN_{RF2}$	0.7742 ( $\pm 0.0044$ )	0.7768 ( $\pm 0.0030$ )	0.7646 ( $\pm 0.0044$ )
	$GGNN_{RF3}$	0.7724 ( $\pm 0.0110$ )	0.7724 ( $\pm 0.0050$ )	0.7572 ( $\pm 0.0063$ )
	$LSTM-NN$	0.7610 ( $\pm 0.0065$ )	0.7707 ( $\pm 0.0052$ )	0.7559 ( $\pm 0.0066$ )
	$MLP-NN$	0.7763 ( $\pm 0.0076$ )	0.7649 ( $\pm 0.0091$ )	0.7553 ( $\pm 0.0164$ )
	$CNN$	0.7496 ( $\pm 0.0043$ )	0.7579 ( $\pm 0.0058$ )	0.7505 ( $\pm 0.0035$ )
sepsis	$GGNN_{RF1}$	<b>0.6141</b> ( $\pm 0.0113$ )	<b>0.6192</b> ( $\pm 0.0099$ )	<b>0.6050</b> ( $\pm 0.0100$ )
	$GGNN_{RF2}$	0.5851 ( $\pm 0.0193$ )	0.5956 ( $\pm 0.0134$ )	0.5816 ( $\pm 0.0130$ )
	$GGNN_{RF3}$	0.4614 ( $\pm 0.0305$ )	0.5018 ( $\pm 0.0154$ )	0.4591 ( $\pm 0.0273$ )
	$LSTM-NN$	0.6101 ( $\pm 0.0105$ )	0.6137 ( $\pm 0.0068$ )	0.6040 ( $\pm 0.0072$ )
	$MLP-NN$	0.5892 ( $\pm 0.0134$ )	0.5913 ( $\pm 0.0088$ )	0.5819 ( $\pm 0.0088$ )
	$CNN$	0.5881 ( $\pm 0.0094$ )	0.5864 ( $\pm 0.0082$ )	0.5800 ( $\pm 0.0071$ )

**Fig. 2.** Precision and Recall of the GGNN and Baseline models per prefix size for bpi2012w (Average over Ten Folds).

perform insufficiently for the sepsis event log because it includes a higher number of different activities and a lower instance per variant ratio than the bpi2012w event log.

**The Problem of Recurrence with GNNs:** While  $RF1$  assumes a node as event,  $RF2$  and  $RF3$  consider a node as activity. Naturally, an activity can be performed more than once within a process instance. However, a node is typically described by a static feature vector, and it does not exist an individual feature vector for each repetition of an activity. To address this problem, we follow the suggestion by Venugopal et al. [22] and only store the node features of the last occurrence of an activity. To address this problem, a sliding window approach may help to exploit more of the historical information of a node that represents an activity. For example, with a window of size three, information on the last three activity occurrences can be stored in a node vector. Another approach to address this problem can be creating an embedding vector per feature and calculating

the average over the vectors of the same feature. For example, such embedding vectors can be created using *word2vec* [13].

**Limitations:** Even though the results are promising, our work has two limitations. First, we selected standard values for most hyper-parameters of the DNN models. We suppose that the DNN model’s predictive quality can be further improved if their hyper-parameters are selected for each event log by performing a hyper-parameter search method such as tree-structured parzen estimators. Second, the third representation form’s adjacency matrix reflects a DFG created based on the entire event log. Since a DFG is also created based on prefixes from the test set, information from the test set is used in the training phase of these models. However, we decided to create the DFGs based on the entire event log to avoid a representation bias between a DFG created based on the training set and prefixes from the test set. A promising approach to address this bias can be trace alignment [8].

## 6 Conclusion

This paper investigated GGNNs for the prediction of the next activities in operational business processes. Motivated by the high expressiveness of GGNNs, we explored three conceptually different representation forms of process instance graphs. Our results show that the first representation form, assuming events as nodes and using a prefix-based adjacency matrix, works best, and GGNNs with this representation form achieve a higher *Precision*, *Recall*, and *F1-Score* than the baselines for both event logs used.

In future research, we plan to develop new GNN architectures for PBPM, such as spatial, temporal GCNs, combining graph convolutions with LSTM cells or GRUs, or graph-based transformer networks. It would also be interesting to investigate unsupervised graph-based approaches, such as *graph2vec* [14], to learn expressive representations of event log data that can be used as input for subsequent models, addressing a specific prediction task like the next activity prediction. Another avenue for future research is to investigate different discovery algorithms for process model creation. Since the created process models can be used as the underlying structure for process instance graphs, DL-based PBPM techniques can be developed that are process-aware [11]. Finally, we interpreted the next activity prediction as a graph-based classification task in this paper. However, since GNNs can also be designed for node-based or edge-based classification tasks, researchers can proof whether and how these classification approaches can be applied in BPM.

## References

1. Bishop, C.M.: Pattern Recognition and Machine Learning. Springer, Heidelberg (2006)
2. Breuker, D., Matzner, M., Delfmann, P., Becker, J.: Comprehensible predictive models for business processes. *MIS Q.* **40**(4), 1009–1034 (2016)
3. Cho, K., van Merriënboer, B., et al.: Learning phrase representations using RNN encoder-decoder for statistical machine translation. In: Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP), pp. 1724–1734 (2014)

4. Gilmer, J., Schoenholz, S.S., Riley, P.F., Vinyals, O., Dahl, G.E.: Neural message passing for quantum chemistry. In: Proceedings of the 34th International Conference on Machine Learning (ICML), pp. 1263–1272. PMLR (2017)
5. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proceedings of the 4th International Conference on Learning Representations (ICLR) (2016)
6. Kohavi, R., et al.: A study of cross-validation and bootstrap for accuracy estimation and model selection. In: Proceedings of the 14th International Joint Conference on Artificial Intelligence (IJCAI), pp. 1137–1145 (1995)
7. Wick, C.: Deep learning. *Informatik-Spektrum* **40**(1), 103–107 (2016). <https://doi.org/10.1007/s00287-016-1013-2>
8. Leemans, S.J., Poppe, E., Wynn, M.T.: Directly follows-based process mining: exploration & a case study. In: Proceedings of the 1st International Conference on Process Mining (ICPM), pp. 25–32. IEEE (2019)
9. Li, Y., Tarlow, D., et al.: Gated graph sequence neural networks. In: Proceedings of the 4th International Conference on Learning Representations (ICLR) (2016)
10. Harl, M., Weinzierl, S., Stierle, M., Matzner, M.: Explainable predictive business process monitoring using gated graph neural networks. *J. Decis. Syst.* (2020)
11. Márquez-Chamorro, A.E., Resinas, M., Ruiz-Cortés, A.: Predictive monitoring of business processes: a survey. *IEEE Trans. Serv. Comput.* **11**(6) (2017)
12. Mehdiyev, N., Evermann, J., Fettke, P.: A novel business process prediction model using a deep learning method. *Bus. Inf. Syst. Eng.* **62**(2), 143–157 (2020)
13. Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J.: Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th Advances in Neural Information Processing Systems (NIPS) (2013)
14. Narayanan, A., Chandramohan, M., Venkatesan, R., Chen, L., Liu, Y., Jaiswal, S.: graph2vec: learning distributed representations of graphs. In: Proceedings of the 16th International Workshop on Mining and Learning Graphs (2017)
15. Philipp, P., Georgi, R.X.M., Beyerer, J., Robert, S.: Analysis of control flow graphs using graph convolutional neural networks. In: Proceedings of the 6th International Conference on Soft Computing & Machine Intelligence (ISCM), pp. 73–77. IEEE (2019)
16. Poll, R., Polyvyanyy, A., Rosemann, M., Röglinger, M., Rupperecht, L.: Process forecasting: towards proactive business process management. In: Weske, M., Montali, M., Weber, I., vom Brocke, J. (eds.) *BPM 2018*. LNCS, vol. 11080, pp. 496–512. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-98648-7\\_29](https://doi.org/10.1007/978-3-319-98648-7_29)
17. Rama-Maneiro, E., Vidal, J.C., Lama, M.: Deep Learning for Predictive Business Process Monitoring: Review and Benchmark. arXiv preprint [arXiv:2009.13251](https://arxiv.org/abs/2009.13251) (2020)
18. Rizzi, W., Di Francescomarino, C., Maggi, F.M.: Explainability in predictive process monitoring: when understanding helps improving. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) *BPM 2020*. LNBP, vol. 392, pp. 141–158. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-58638-6\\_9](https://doi.org/10.1007/978-3-030-58638-6_9)
19. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. *IEEE Trans. Neural Netw.* **20**(1), 61–80 (2008)
20. Stierle, M., Weinzierl, S., Harl, M., Matzner, M.: A technique for determining relevance scores of process activities using graph-based neural networks. *Decis. Support Syst.* **144**, 113511 (2021)
21. Tax, N., Verenich, I., La Rosa, M., Dumas, M.: Predictive business process monitoring with LSTM neural networks. In: Dubois, E., Pohl, K. (eds.) *CAiSE 2017*. LNCS, vol. 10253, pp. 477–492. Springer, Cham (2017). [https://doi.org/10.1007/978-3-319-59536-8\\_30](https://doi.org/10.1007/978-3-319-59536-8_30)

22. Venugopal, I., Töllich, J., Fairbank, M., Scherp, A.: A comparison of deep-learning methods for analyzing and predicting business processes. In: Proceedings of the International Joint Conference on Neural Networks (IJCNN) (2021)
23. Verenich, I., Dumas, M., La Rosa, M., Nguyen, H.: Predicting process performance: a white-box approach based on process models. *J. Softw.: Evol. Process* **31**(6), e2170 (2019)
24. Weinzierl, S., Revoredo, K.C., Matzner, M.: Predictive business process monitoring with context information from documents. In: Proceedings of the 27th European Conference on Information Systems (ECIS), pp. 1–10. AISeL (2019)
25. Zhou, J., Cui, G., et al.: Graph neural networks: a review of methods and applications. *AI Open* **1**, 57–81 (2020)