



# Exploiting Instance Graphs and Graph Neural Networks for Next Activity Prediction

Andrea Chiorrini<sup>(✉)</sup>, Claudia Diamantini, Alex Mircoli, and Domenico Potena

Department of Information Engineering,  
Polytechnic University of Marche, Ancona, Italy  
a.chiorrini@pm.univpm.it, {c.diamantini,a.mircoli,d.potena}@univpm.it

**Abstract.** Nowadays, a lot of data regarding business process executions are maintained in event logs. The next activity prediction task exploits such event logs to predict how process executions will unfold up until their completion. The present paper proposes a new approach to address this task: instead of using traces to perform predictions, we propose to use the instance graphs derived from traces. To make the most out of such representation we train a message passing neural network, specifically a Deep Graph Convolutional Neural Network to predict the next activity that will be performed in the process execution. The experiments performed show promising performance hinting that exploiting information about parallelism among activities in a process can induce a performance improvement in highly parallel process.

**Keywords:** Deep Learning · next activity prediction · Predictive Process Monitoring · Graph Neural Networks · Process Mining

## 1 Introduction

Nowadays, many business processes maintain a significant amount of data regarding their executions in the form of events logs. The predictive process monitoring field is concerned with the exploitation of such event logs to predict how such executions will unfold up until their completion. Predictive process monitoring includes various tasks: one of them is the next activity prediction, that is concerned with the prediction of what will next happen in an execution.

In the last years predictive process monitoring, and particularly next activity prediction, is receiving an ever-increasing attention, and researchers are tackling the problem using various deep learning approaches [3, 4, 13–15, 19, 21]. It is also known that using graphs is an extremely convenient way of representing process executions [6, 20]. Recently, a new type of neural network architecture is gaining ground in the deep learning community: the graph neural network [16, 24]. Still, there is almost no work in the literature that evaluates the possibility of exploiting such family of networks for the predictive process monitoring tasks. Driven

by this lack of studies, we introduce a methodology to properly exploit the graph representation of processes executions [6] for the next activity prediction task.

In particular, our proposal requires a business process event log and a model of such process, which are used to build for each trace a proper corresponding instance graph. From this graph representation, the dataset is then created and the Deep Convolutional Graph Neural Network [25] is trained to perform the prediction for next activity. Experimental results show that information contained in instance graphs, in particular concerning the parallelism among activities, can improve the prediction accuracy.

The rest of the paper is organized as follows: in Sect. 2 a review of the task-relevant literature is performed. Section 3 describes the proposed methodology. In Sect. 4 a comparison with the results from the relevant literature is performed and commented. Finally, in Sect. 5 we draw some conclusions and list further directions of research.

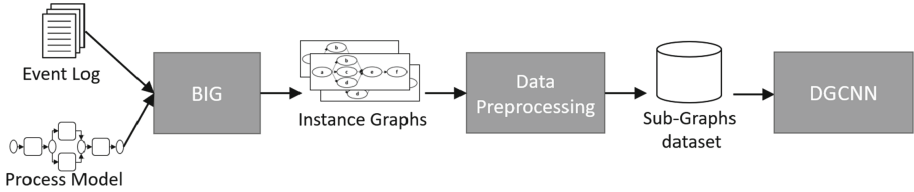
## 2 Related Work

In the scientific community there is an ever-increasing interest in the application of deep learning techniques to predictive process monitoring tasks. In [18] the authors proposed to adopt an LSTM architecture for the one step ahead event prediction and the suffix prediction, using the one-hot encoding of the associated activity and three temporal features related to the event’s timestamp. The LSTM architecture has also been used by [3, 8, 13]. The approach in [8] predicts only the next activity type, while [13] predicts the next activity and all the associated categorical attribute. The authors of [3] combine both [8] and [18] approaches to extend [8] to the next completion time prediction, using an abstract notion of class resource, i.e. group of resources that usually perform similar activities. Recently, in [14] a CNN architecture has been proposed: the authors convert the sequential temporal data in the log into a spatial representation to then treat data as images. In [15] the approach has been further extended by the authors.

The use of Generative Adversarial Nets (GANs) is proposed in [19] to address the next event prediction task, tackling in this way the lack of sufficient training data that often impact performances. In the GAN approach the authors used LSTM networks for both the generator and discriminator. For this reason they trained various networks, each one over sub-sequences of processes of specific length, thus producing more than one prediction model for each process, the same limit of the other LSTM based approaches.

In 2008 Scarselli et al. introduced “The Graph Neural Network Model” (GNN) [16], a neural network model capable of processing data in graph domains. Since their definition, graph neural networks have been increasingly used in several fields. In a recent survey [24] it has been proposed *a new taxonomy to divide the state-of-the-art GNNs into four categories, namely, recurrent GNNs, convolutional GNNs, graph autoencoders, and spatial-temporal GNNs*. In the same work the authors also outline the various application fields.

It is relevant to note that very recently a proposal of usage of Graph Convolutional Neural Network in the next activity and timestamp predictions has



**Fig. 1.** The BIG-DGCNN methodology pipeline

been formulated by [21]. Their approach differs from the one proposed in the present paper as they propose to use a Directly Follows Graph representation to model the traces while our approach adopts an Instance Graph representation. Furthermore, the network architecture used in this work and in [21] are different. In [21] a single graph convolutional layer followed by two fully connected layers are used, while in the present paper a more elaborated architecture is adopted as described in Sect. 3. Also lately, in [23] the author explored the opportunity of exploiting gated graph neural network for the next activity prediction. In the work different graph representations are tested but are only compared with self implemented baselines, focusing the work exclusively on network architecture.

### 3 Methodology

In this Section we discuss the proposed methodology, first explaining the rationale of the whole pipeline shown in Fig. 1, then delving into its relevant aspects.

The goal of the present work is to define a robust approach that makes use of information about the parallelism among activities in the next activity prediction task. To this end, we propose to represent each trace with its corresponding Instance Graph (IG), and to process IGs by graph neural networks, that are designed to natively manage graph structures.

It is known that replaying a trace on a Petri net it is possible to produce an instance graph representing the process execution [6]. The starting point of our methodology is thus an event log and a model of the process expressed as Petri net. The model can be given by a domain expert, opening to the possibility to provide the desired perspective over the process, or a Petri net can be derived by some process discovery algorithm. In both cases, a problem arises with highly variable processes whose event log includes many non-conforming traces. This situation often occurs when an a-priori model has been defined for a process that rapidly evolves thus making the model obsolete; or in decision-intensive processes where only a high-level model can be defined and executions vary from case to case (e.g. care process in a hospital). When process discovery is exploited to synthesize a process, lossy algorithms may be adopted (e.g. Heuristic Miner, infrequent Inductive Miner) that discard uncommon behaviours to ensure simpler and more meaningful models. In the presence of non-conforming traces a simple replay will lead to not valid IGs, e.g. disconnected graphs, or graphs with more than one terminal node. To deal with this issue, our approach also provides

a repairing of non-conforming traces by the adoption of the BIG algorithm [5]. This prevents the loss of data and grants to our system the possibility to predict also uncommon events. From BIG-generated IGs the dataset is then created and the graph neural network trained to perform the prediction of the next activity. Among the different architectures existing in the literature, the Deep Graph Convolutional Neural Network (DGNN) [25] has been chosen. In the following we will refer to our methodology as BIG-DGCNN.

### 3.1 Building Instance Graphs

An Instance Graph (IG) is a graph that describes a specific execution of some process model. In an IG each node represents an activity, and an edge between activity A and activity B denotes the existence of a *causal relation* between A and B, namely the fact that B cannot be executed until A is terminated; in other words, the execution of B depends on the execution of A. For a more formal definition of instance graphs and causal relations see [6]. In an instance graph parallelisms are shown besides causal relations, while choices are not represented. This is obvious, since in each single execution choices have already been made. Activities that can be done in parallel within one instance can appear in any order in the trace without changing the resulting instance graph. Hence, an execution is represented by an unique IG, which in turn represent different traces.

Instance graphs can be built from a set of traces in an event log and the corresponding process model. In this paper we refer to the Building Instance Graph (BIG) algorithm proposed in [5]. Unlike other approaches in the literature, BIG enables the representation of parallel behaviors and is able to handle traces that do not conform to the model. BIG is a two-steps algorithm: first an IG is extracted for each trace, then IGs from non-conforming traces are repaired. In the first step, a node is created in the IG for each event of the trace and the node is labeled with both the activity associated with the given event and the position of the event in the trace. For each pair of events  $(e_i, e_j)$  with  $i < j$  such that causal relation exists between the corresponding activities in the model, an edge is inserted between the corresponding nodes in the IG if and only if between the pair of events there is neither another causal successor of  $e_i$ , nor a causal predecessor of  $e_j$ .

It is worth noting that, if the trace does not fit the model, the corresponding IG is affected by several issues, e.g. being a disconnected graph and having multiple terminal nodes. From a semantic point-of-view these IGs represent many more behaviours than the model actually allows for, namely they over-generalize. Hence, the representation of the process behavior provided by these IGs is very poor. In order to mitigate these issues, in [5] a procedure for the repair of IGs is proposed.

First, non-synchronous movements in the trace, which lead to issues in the IG, are identified by using aligned-based conformance checking technique [2], which returns the kind of non-synchronous movements and their position in the trace. This information is used to repair each move-in-the-model and each move-in-the-log occurring in the alignment. The former leads to disconnected graphs, which

are repaired by identifying nodes corresponding to activities that are in a causal relation with the move-in-the-model and properly connecting them. For each move-in-the-log, the repair procedure changes the edges connecting the nodes corresponding to the events before and the events after the non-synchronous event; the causal relations among the activity corresponding to move-in-the-log and activities related to its predecessors and successors in the trace are used to build new edges.

### 3.2 Data Preprocessing

This Section describes the processing of the log to obtain the dataset for training and testing. First of all, since a log not necessarily has single specific ending and starting activities, we introduce such artificial events in the logs. This is done to guarantee that a trace always has a termination activity and to ensure that parallelism at the beginning and end of a process execution are properly represented in the IG. Second, BIG is applied to obtain an IG for each trace.

The approach of this paper learns a function that, given a *graph prefix*  $\tau_g$  of dimension  $g$ , returns a label  $a$  that can be interpreted as the next performed activity in the process execution. Hence, the third step of the proposed methodology is to build the pairs  $(\tau_g, a)$  from an IG for any  $g \in \{2, \dots, N - 1\}$  where  $N$  is the number of nodes in the IG (i.e. the trace length). Therefore from one IG, we produce  $N - 2$  pairs. The procedure is repeated for every IGs in the dataset.

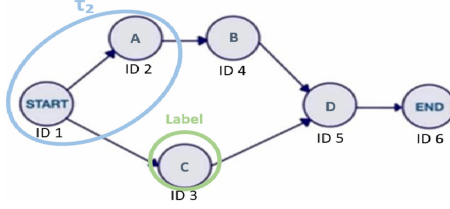
Each node in an IG has a progressive index associated with it that represents its position in the trace. This index determines an order of the nodes, which we use to progressively build the graph prefixes from the full IG. We denote by  $a_i$ ,  $i = 1, \dots, N$  the activity associated with the node of index  $i$ .

Given an IG, the graph prefix  $\tau_2$  is obtained by selecting the first two nodes and the edges between them. This prefix is labelled with the activity  $a_3$  of the next node (Fig. 2). The next prefix is simply derived by  $\tau_2$ , extending it with the node of index 3 and the edges connecting it to  $\tau_2$ . The associated label is  $a_4$  (Fig. 3). Iteratively, the procedure re-build the whole instance graph until the last node is selected as label.

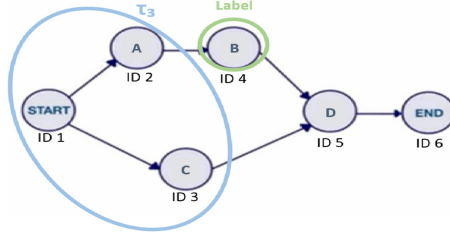
We adopt the one-hot encoding representation for the set of activities in the log.

### 3.3 Deep Graph Convolutional Neural Network

In this paper, we use a variant of the Deep Graph Convolutional Neural Network (DGCNN) proposed in [25]. The DGCNN is composed of three sequential stages. In Fig. 4, we show a representation of the overall neural network architecture. First it has several graph convolution layers which extract the features from the nodes local substructure and define a consistent vertex ordering. Second it has a SortPoolingLayer which sorts the vertex features according to the order defined in the previous stage, selecting the top nodes. In this way the dimension of the input is unified. At last, a 1-D convolution layer and a dense layer take the obtained representation to perform predictions.



**Fig. 2.** First labelling step



**Fig. 3.** Second labelling step

The graph convolution layer adopted by DGCNN is represented by the following formula:

$$Z = f(\tilde{D}^{-1}\tilde{A}XW) \quad (1)$$

where  $\tilde{A} = A + I$  is the adjacency matrix ( $A$ ) of the graph with added self-loops ( $I$ ),  $\tilde{D}$  is its diagonal degree matrix with  $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$ ,  $X \in \mathbb{R}^{n \times c}$  is the graph nodes information matrix (in our case the one-hot encoding of the activity labels associated to the nodes),  $W \in \mathbb{R}^{c \times c'}$  is the matrix of trainable weight parameters,  $f$  is a nonlinear activation function, and  $Z \in \mathbb{R}^{n \times c'}$  is the output activation matrix. In the formulas,  $n$  is the number of nodes of the input graph (in our case the graph prefix),  $c$  is the number of features associated to a node, and  $c'$  is the number of features in the next layer tensor representation of the node.

In a graph, the convolution operation aggregates node information in local neighborhoods so to extract local structural information. In order to extract multi-scale structural features, multiple graph convolution layers (Eq. 1) are stacked as follows:

$$Z^{k+1} = f(\tilde{D}^{-1}\tilde{A}Z^k W^k) \quad (2)$$

where  $Z^0 = X$ ,  $Z^k \in \mathbb{R}^{n \times c_k}$  is the output of the  $k^{th}$  convolution layer,  $c_k$  is the number of features of layer  $k$ , and  $W^k \in \mathbb{R}^{c_k \times c_{k+1}}$  maps  $c_k$  features to  $c_{k+1}$  features.

The graph convolution outputs  $Z^k, k = 1, \dots, h$  are then concatenated in a tensor  $Z^{1:h} := [Z^1, \dots, Z^h] \in \mathbb{R}^{n \times \sum_1^h c_k}$  which is then passed to the Sort-PoolingLayer. It first sorts the input  $Z^{1:h}$  row-wise according to  $Z^h$ , and then returns as output the top  $m$  nodes representations, where  $m$  is a user-defined

parameter. This way, it is possible to train the next layers on the resulting fixed-size graph representation.

In the original proposal the DGCNN includes a 1-D convolution layer, followed by several MaxPooling layers, one further 1-D convolution layer followed by a dense layer and a softmax layer.

In the present paper we simplify the architecture leaving only one 1-D convolution layer with dropout [17] followed by a dense and a softmax layer. This is because the process mining domain tend to present smaller graphs in comparison with those of typical application domains of graph neural networks [24].

For further information on the architecture we refer the interested reader to [25].

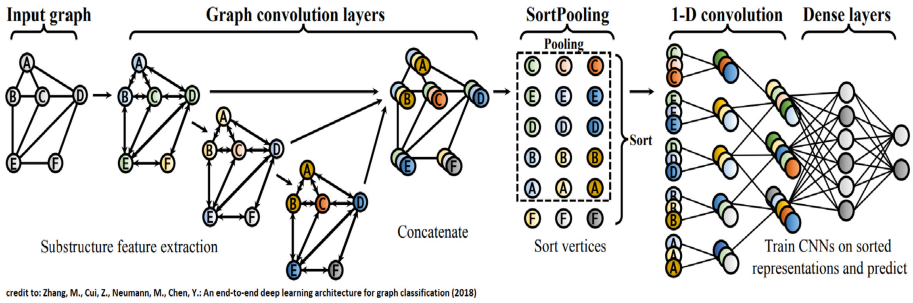


Fig. 4. DGCNN architecture. Taken from [25].

## 4 Experiments

### 4.1 Experimental Setup

In this Section, we characterize the benchmark datasets used, then we discuss the setting of parameters for the algorithms used in the experiments.

**Datasets.** In order to be comparable with the literature [3, 14, 18, 21], we tested our approach on 2 commonly used benchmark datasets, namely *Helpdesk* and *BPI12W*.

The *Helpdesk* dataset [22] contains traces from a ticketing management process of the help desk of an Italian software company. In this process all execution instances start with the insertion of a new ticket into the ticketing management system.

**Table 1.** Overview of benchmark dataset.  $|\sigma|$  is used to represent the trace length.

Dataset	N.traces	Tot.events	N.act.types	Min $ \sigma $	Max $ \sigma $	Avg $ \sigma $
Helpdesk	3804	13710	9	1	14	3.60
BPI12W	9658	72413	6	1	74	20.03

The *BPI12* dataset [7] is taken from a Dutch Financial Institute. The process represents a personal loan or overdraft applications within a global financing organization. The event log is a merge of three intertwined sub processes, related to the evolution of the status of the loan/draft application (*BPI12A*), of the work items belonging to the application (*BPI12W*), and of the offer belonging to the application (*BPI12O*) respectively. We considered the *BPI12W* sub-process. Furthermore, as usually done, we retained only the completed events in the log. Table 1, shows the characteristics of both datasets.

We hasten to note that *BPI12W* execution instances show no parallel activities, while some level of parallelism exists in *Helpdesk*. Thus the selection of these datasets allows us to better appreciate the impact of parallelisms on performance.

Finally, in order to be as much comparable as possible with the literature, we split each log keeping the first (chronologically ordered) 67% of the traces for training and the remaining part for testing.

**Parameter Settings.** In the BIG-DGCNN methodology there are two main algorithms that require the setting of parameters: the infrequent Inductive Miner (iIM) [12] used to derive the Petri net representing the process models requested for the execution of the BIG algorithm, and the DGCNN. Regarding iIM, the noise threshold must be set. This threshold is used to determine how much infrequent behaviour will be filtered out when building the model. In choosing such parameter we tested the fitness of the resulting model (i.e. the extent to which the discovered model can accurately reproduce the cases recorded in the log). We changed the noise threshold with 10% step from 0% to 100% and selected the smallest noise threshold that granted at least a 90% fitness. In this way, we obtain processes that are capable of modeling a vast majority of traces while still maintaining a good degree of generalization, thus emulating a model provided by an expert, and putting us in a close-to-real setting.

Regarding the parameters of the DGCNN, as stated before we set the number of 1-D convolution layers to one, followed by a dense layer, both with 32 neurons. We used ADAM [10] as optimization algorithm and trained the network for 100 epochs with an early stopping. We used as loss function the categorical cross entropy. For both datasets we varied the following parameters:

- the number of nodes selected by the SortPooling layer ( $m$ ), in  $\{3, 5, 30\}$
- the number of stacked graph convolution layer ( $h$ ), in  $\{2, 3, 5, 7\}$
- the batch size ( $bs$ ), in  $\{16, 32, 64, 128\}$



- the initial learning rate ( $lr$ ), in  $\{10^{-2}, 10^{-3}, 10^{-4}\}$
- the dropout percentage ( $do$ ), in  $\{0.1, 0.2, 0.5\}$

The configurations that provide the best accuracy are shown in Table 2.

**Table 2.** Best network parameters for the dataset.

Dataset	$m$	$h$	$bs$	$lr$	$do$
Helpdesk	30	5	32	$10^{-3}$	0.1
BPI12W	5	5	32	$10^{-3}$	0.2

**Tools and Hardware.** All the experiments have been performed using pytorch geometric [9] with torch version 1.8.1, on a Tesla T4 GPU, a Intel(R) Xeon(R) CPU@2.20 GHz, and a 12 GB RAM.

## 4.2 Results

Table 3 shows the accuracy achieved on the selected datasets, by our approach and other approaches exploiting different neural network architectures.

We can see that on Helpdesk BIG-DGCNN achieves the best performance. It is worth noting that in all the competitor approaches instances are described by a richer set of features, like time information, while our input graphs encode only activity labels and flow information.

On the other hand, on the BPI12W dataset there is a relevant performance degradation w.r.t. the other methods. As noted before, BPI12W shows no parallel activities. This seems to confirm our hypothesis that properly taking into account information on parallelisms can be beneficial to the next activity prediction, whilst graph neural networks do not develop their full potential on sequential processes. The hypothesis also gains confidence when the BIG-DGCNN performances are compared against the results presented in [19] on the whole BPI12

**Table 3.** Literature comparison, measured accuracy

Approach	Dataset	
	Helpdesk	BPI12w
BIG-DGCNN	<b>82.58%</b>	62.43%
GCNN [21]	79.54%	65.69%
MLP [21]	82.01%	65.59%
CNN [14]	73.93 %	78.17%
CNN [15]	-	<b>82.20%</b>
LSTM [18]	71.23%	76.00%
LSTM [3]	78.90%	77.80%

dataset. There, plain CNN [14], and LSTM [3, 18] approaches has been compared to a LSTM trained in a GAN framework. BIG-DGCNN is able to achieve an accuracy of 76.45% which is higher than the accuracy presented for all the plain methods, although it is lower than that achieved by LSTM+GAN. We remark that GAN is a novel learning framework, where any neural network architecture can be adopted. It is possible that the adoption of graph neural networks like those presented in this work could further improve performance, solving also a limit of the LSTM network used in [19] which is constrained to fixed size prefixes, hence forcing the training of a specialized network for every desired prefix length. We plan to experiment the GAN framework with graph neural networks in future work.

Since [21] also adopts a graph neural network architecture, one may want to analyse the reasons under the different accuracy. Various factors can be responsible for this: first, [21] adopts directly follows graphs, which are derived at a process level, while we build and properly repair the instance graph for each trace. Second, the representation of the graph is performed in a different way. The approach described in [21] only takes into account the last occurred event of a sequence of events with the same activity label when building the input graph. On the contrary, our approach considers all passed events of the trace, including the events relative to repeated actions. Third, the network used in this work is endowed with several graph convolution layers, while that used in [21] is a single layer graph convolutional neural network [11] variant.

## 5 Conclusions and Future Works

The main contribution of this work is the definition of BIG-DGCNN, a methodology to address the task of next activity prediction exploiting information about parallelism among activities in a process. The methodology adopts BIG [5] to repair non-conforming traces in order to always build a fully representative instance graph. Then it uses a rather new kind of neural network architecture, the Deep Graph Convolutional Neural Network [25], that is capable of effectively using the structural information of a graph in its functioning. The comparison with the literature highlights that BIG-DGCNN show promising performance in datasets relative to process with a consistent presence of parallelism, while performing less effectively in sequential datasets like BPI12W. Although it is well known that parallelism is a characterizing feature of business processes [1], variants of the approach that can better deal with sequential dataset can be worth of investigation. Other interesting future research directions include:

- extending the BIG-DGCNN inputs to all the available features of every benchmark dataset, aspect neglected in this work so to isolate the structural contribution of the process workflow,
- extending the experimentation to other datasets,
- testing the GAN training method using BIG-DGCNN thus “*eliminating the need for a large training data*” [19] and avoiding the fixed prefix restriction of GAN+LSTM.

## References

1. van der Aalst, W., et al.: Process mining manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (eds.) BPM 2011. LNBIP, vol. 99, pp. 169–194. Springer, Heidelberg (2012). [https://doi.org/10.1007/978-3-642-28108-2\\_19](https://doi.org/10.1007/978-3-642-28108-2_19)
2. Adriansyah, A., van Dongen, B.F., van der Aalst, W.M.: Conformance checking using cost-based fitness analysis. In: 2011 IEEE 15th International Enterprise Distributed Object Computing Conference, pp. 55–64. IEEE (2011)
3. Camargo, M., Dumas, M., González-Rojas, O.: Learning accurate LSTM models of business processes. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 286–302. Springer, Cham (2019). [https://doi.org/10.1007/978-3-030-26619-6\\_19](https://doi.org/10.1007/978-3-030-26619-6_19)
4. Chiorrini, A., Diamantini, C., Mircoli, A., Potena, D.: A preliminary study on the application of reinforcement learning for predictive process monitoring. In: Lee-mans, S., Leopold, H. (eds.) ICPM 2020. LNBIP, vol. 406, pp. 124–135. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-72693-5\\_10](https://doi.org/10.1007/978-3-030-72693-5_10)
5. Diamantini, C., Genga, L., Potena, D., van der Aalst, W.: Building instance graphs for highly variable processes. *Expert Syst. Appl.* **59**, 101–118 (2016)
6. van Dongen, B.F., van der Aalst, W.M.P.: Multi-phase process mining: building instance graphs. In: Atzeni, P., Chu, W., Lu, H., Zhou, S., Ling, T.-W. (eds.) ER 2004. LNCS, vol. 3288, pp. 362–376. Springer, Heidelberg (2004). [https://doi.org/10.1007/978-3-540-30464-7\\_29](https://doi.org/10.1007/978-3-540-30464-7_29)
7. van Dongen, B.: BPI challenge 2012, April 2012
8. Evermann, J., Rehse, J.R., Fettke, P.: Predicting process behaviour using deep learning. *Decision Support Syst.* **100**, 129–140 (2017). *Smart Business Process Management*
9. Fey, M., Lenssen, J.E.: Fast graph representation learning with PyTorch Geometric. In: ICLR Workshop on Representation Learning on Graphs and Manifolds (2019)
10. Kingma, D.P., Ba, J.: Adam: a method for stochastic optimization (2017)
11. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: Proceedings of the 5th International Conference on Learning Representations, ICLR 2017 (2017)
12. Leemans, S.J.J., Fahland, D., van der Aalst, W.M.P.: Discovering block-structured process models from incomplete event logs. In: Ciardo, G., Kindler, E. (eds.) PETRI NETS 2014. LNCS, vol. 8489, pp. 91–110. Springer, Cham (2014). [https://doi.org/10.1007/978-3-319-07734-5\\_6](https://doi.org/10.1007/978-3-319-07734-5_6)
13. Lin, L., Wen, L., Wang, J.: MM-Pred: a deep predictive model for multi-attribute event sequence. In: Proceedings, Society for Industrial and Applied Mathematics, pp. 118–126 (2019)
14. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Using convolutional neural networks for predictive process analytics. In: 2019 International Conference on Process Mining (ICPM 2019), pp. 129–136 (2019)
15. Pasquadibisceglie, V., Appice, A., Castellano, G., Malerba, D.: Predictive process mining meets computer vision. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNBIP, vol. 392, pp. 176–192. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-58638-6\\_11](https://doi.org/10.1007/978-3-030-58638-6_11)
16. Scarselli, F., Gori, M., Tsoi, A.C., Hagenbuchner, M., Monfardini, G.: The graph neural network model. *IEEE Trans. Neural Networks* **20**(1), 61–80 (2009)
17. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R.: Dropout: a simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **15**(1), 1929–1958 (2014)

18. 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)
19. Taymouri, F., Rosa, M.L., Erfani, S., Bozorgi, Z.D., Verenich, I.: Predictive business process monitoring via generative adversarial nets: the case of next event prediction. In: Fahland, D., Ghidini, C., Becker, J., Dumas, M. (eds.) BPM 2020. LNCS, vol. 12168, pp. 237–256. Springer, Cham (2020). [https://doi.org/10.1007/978-3-030-58666-9\\_14](https://doi.org/10.1007/978-3-030-58666-9_14)
20. van der Aalst, W., van Dongen, B., Herbst, J., Maruster, L., Schimm, G., Weijters, A.: Workflow mining: a survey of issues and approaches. *Data Knowl. Eng.* **47**(2), 237–267 (2003)
21. Venugopal, I., Tollich, J., Fairbank, M., Scherp, A.: A comparison of deep learning methods for analysing and predicting business processes. In: Proceedings of International Joint Conference on Neural Networks, IJCNN. IEEE Press, July 2021
22. Verenich, I.: Helpdesk (2016). <https://doi.org/10.17632/39bp3vv62t.1>. <https://data.mendeley.com/datasets/39bp3vv62t/1>
23. Weinzierl, S.: Exploring gated graph sequence neural networks for predicting next process activities. In: Marrella, A., Weber, B. (eds.) BPM 2021. LNBIP, vol. 436, pp. 30–42. Springer, Cham (2021). [https://doi.org/10.1007/978-3-030-94343-1\\_3](https://doi.org/10.1007/978-3-030-94343-1_3)
24. Wu, Z., Pan, S., Chen, F., Long, G., Zhang, C., Yu, P.S.: A comprehensive survey on graph neural networks. *IEEE Trans. Neural Netw. Learn. Syst.* **32**(1), 4–24 (2021)
25. Zhang, M., Cui, Z., Neumann, M., Chen, Y.: An end-to-end deep learning architecture for graph classification. In: 32nd AAAI Conference on Artificial Intelligence (2018)

**Open Access** This chapter is licensed under the terms of the Creative Commons Attribution 4.0 International License (<http://creativecommons.org/licenses/by/4.0/>), which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons license and indicate if changes were made.

The images or other third party material in this chapter are included in the chapter's Creative Commons license, unless indicated otherwise in a credit line to the material. If material is not included in the chapter's Creative Commons license and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder.

