
*Production Systems and Information Engineering*
Volume 10 (2), pp. 11–26
doi: 10.32968/psaie.2022.2.2.


# ACTIVITY LOGS IN PRACTICE

Péter Mileff
University of Miskolc, Hungary
Department of Information Engineering
mileff@iit.uni-miskolc.hu

Judit Dudra
Bay Zoltán Nonprofit Ltd. for Applied Research, Hungary
Department of Structural Integrity and Production Technologies
judit.dudra@bayzoltan.hu

**Abstract.** Modern information technology is now present virtually everywhere, in all areas. For the increasingly complex processes, complex information systems are developed that can be used to provide effective support for the processes. There is a lot of data flowing through information systems that is now essential to examine. RPA offers a solution for this, which allows partial or even complete automation of processes. One of the important basic units of RPA may be the activity logs generated in practice. In this publication, this area is reviewed. The most important formats are briefly presented, followed by a runtime model whose aim is to hide the differences in the formats, to achieve a general structure. Building on this module, an MLP model that implements the prediction of atomic events is finally presented. The publication approaches the problem from a practical point of view and proves the effectiveness of the model with test results.

*Keywords*: Activity log, MLP neural network, process mining

## 1. Introduction

Today's increasingly complex social model results in increasingly complex business processes. It is a natural and general endeavor to be able to support these processes with IT systems, thereby increasing efficiency within the company. Without this, the processes can often be slow, and due to their complexity, in many cases their support and follow-up is unsatisfactory without an appropriate support system. Today, however, it is not enough to implement only a complex management system, the appropriate automation has also appeared on the market as a new requirement. Due to the complexity of the business processes, the interface is often quite diverse and complex. If one of these

processes is analyzed in detail, the process solution schemes can be well distinguished. Rules can be defined and set up to form the basis for ongoing decisions. Once a rule base is defined, it is practically the point at which the system can begin to be fully or even partially automated. Nowadays, there are systems where the customer is essentially talking or mailing to an "automaton", where some kind of artificial intelligence-based decision support system is helping to solve the problem and conducting the process. With such systems (RPA - Robotic Process Automation), efficiency can be further increased

One of the new RPA-based research trends today is the automatic discovery of rules[1]. Due to their operation, the systems performing administration consist of data and the process steps that perform transformations on them. A very simple example is the process of ordering an item online, where the order itself is an exact, well-defined process. Within the process, there may be branches or alternative routes for each activity. By following the example above, a payment can be made online or even by cash on delivery. From the state changes of the data used by the process, a sample of rules can be extracted, which could be used to solve the process even with full automation. However, this requires that the states be logged either in a database or in one or more log files. Because RPA is a modern approach to increasing efficiency, most information systems are not adequately prepared to store changes to data in an appropriate format. In most cases, we start by processing some kind of log files, which we try to extract rules by scanning them in a detailed and automated way.

It can be formulated as a general goal to be able to predict certain events based on activity logs extracted from an information system using a model that uses an efficiently configurable artificial neural network.One of the key issues of IT systems based on artificial intelligence is the appropriate data set, as it enables the design of the processes built on it, the learning algorithms and the appropriate learning. A high quality data set is therefore important and needs to be addressed with particular attention. The recorded data, the workflow steps, are organized into a higher level sequence, transaction. Because the system is used by multiple users at the same time, a log file will be created that randomly mixes activities for different transactions. This paper examines the general structural issues of activity logs and their processability in a Python environment from a practical point of view. A general in-memory model format for general handling and description of multiple types of input sets is presented.

## 2. Activity monitoring

The most important starting point for any process that we want to build on the results of user activity in the future will be the so-called event log. The area that deals with these is called Process Mining, which aims to analyze

data from a process perspective. It seeks answers to questions such as "What is the current state of the process?", "Are there unnecessary steps that could be eliminated?", "Where are the bottlenecks?" And "Are there any deviations from the established and prescribed process rules?". In order to be able to analyze the data in the event log in any form, it must be written to the log with some process-specific approach. This makes it possible to decide which step in the event log belongs to which process. Perhaps one of the most important questions in this topic may be, "Where does the data come from in the event log?"

Usually, some kind of activity monitoring software is used to monitor user activities. Activity monitoring software records the use of applications and programs on the monitored workstation on-screen user activities are logged in a pre-designed and well-structured log. So logs are information databases that store all the activities that took place that day. Thanks to today's modern technology, there are many options and solutions available for monitoring, monitoring and managing activities. Some important techniques:

- **Logging and Analysis:** The most common form of user activity monitoring. In this case, we store the events in a classic text file or database. All events and processes are well defined. The resulting log file stores the information in a well-structured format for later processing and analysis.
- **Video recordings of sessions:** user activity is recorded as a video stream. An important goal is to be able to analyze later, to prevent possible security breaches, and to reproduce the interactions that have taken place.
- **Screenshot capture:** similar considerations apply to the technique as in the video-based approach. In this method, images are recorded, which provides a basis for further analysis. In several practical approaches, the current activity is framed with a box on the captured screenshot. For example, a user-selected item, pressed buttons, and so on.
- **Keyboard usage logging:** Logging of keystrokes may be important in cases where the corporate or other system used by users expects less graphical interaction, and more processes may require manual data entry / typing.

## 3. Activity log formats

In practice, the format of activity logs can be of any design. The main aspect is always to create a logical and exact structure for storing and retrieving data. Over the years, several types have emerged, the most important are:

- **CSV:** a well-known and familiar format in an enterprise environment

- **XES:** an XML-based standard format for storing and transmitting data
- **OCEL:** a standardized object-centric format released in 2021
- **Custom database model:** in some systems we can even store events in databases. This is usually only possible if the software system has been pre-designed this way.

Whatever format is used in practice, an activity log must basically consist of at least three mandatory elements: a case id, an activity and a timestamp.

- **Case id:** each case is a step in the execution instance of a process. For example, in an ordering process, handling an order is a case. A very important criterion is that in each case we need to know which process it belongs to.
- **Activity:** activities are different steps in the process or state changes. IT systems can record not only activities that are important to us, but also less interesting debugging information. Having less relevant activities in the log is not a problem in itself, they can be filtered out later. However proper naming of activities is very important from the beginning. Most processes are complex, and the analysis does not go too far if the steps in the process map show purely technical status numbers or operation codes.
- **Timestamp:** each activity requires at least one timestamp to sort each event in the correct order. And if you want to analyze the duration of the activity, you must specify a start and end timestamp for each activity. Efforts should be made to record timestamps as accurately as possible.

### 3.1. The OCEL format

OCEL is a format created in 2021 that aims to provide a general standard to interchange object-centric event data with multiple case notions [2]. It complements the XES standard and is supported by most process mining tools. The standard supports two file format types: JSON-OCEL, XML-OCEL in order to support a widespread collection of languages and systems. The elements of the format and their relationship are shown in the figure below:

The standard supports the storage of events, objects, and their attributes. Considering the integration of OCEL-based support into any information system, it is an object-based format that has an appropriate level of structure while being modern. Although in practice support for the OCEL format is currently very rudimentary. If we look at the popular Python environment, the ocel-standard package in the official Python repositories provides processing, which is currently available in version 0.0.3.1. Although the package is

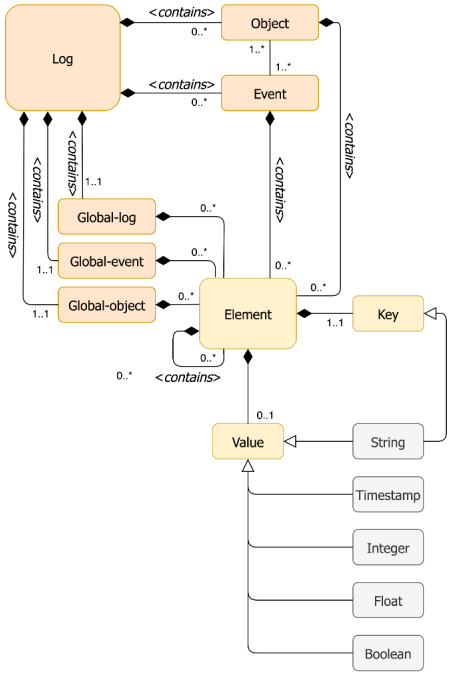


**Figure 1.** OCEL event log complete meta-model structure[2]

very rudimentary, it has already provided enough opportunity for processing to be done.

### 3.2. The XES format

The XES standard defines a grammar for a tag-based language whose aim is to provide designers of information systems with a unified and extensible methodology for capturing systems behaviors by means of event logs and event streams is defined in the XES standard[3]. An XML Schema describing the structure of an XES event log/stream and a XML Schema describing the structure of an extension of such a log/stream are included in this standard. The purpose of this standard is to provide a generally acknowledged XML format for the interchange of event data between information systems in many applications domains on the one hand and analysis tools for such data on the other hand.

### 3.3. The CSV format

The CSV format is one of the types that was available from the beginning. The format has several advantages:

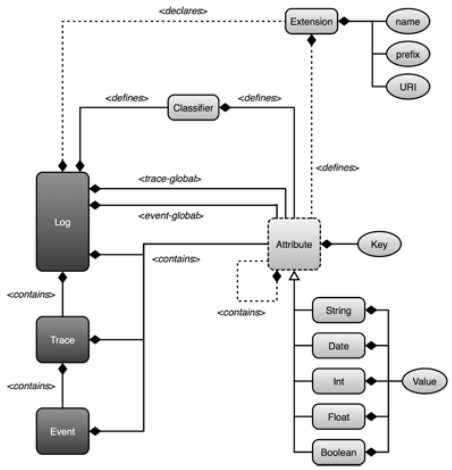


**Figure 2.** XES event log meta-model structure[8]

- **text file:** thanks to the text format, the log file can be opened with any editor and its contents can be viewed
- **easy to use:** the format is simple to structure, making individual processing efficient
- **wide support:** due to its simplicity, it can be effectively integrated into many information systems

A practical sample of the CSV event log format:

```
CaseID,ActivityID,CompleteTimestamp
1,"open","2021-06-17 12:12:01"
1,"edit","2021-06-17 12:13:10"
1,"convert","2021-06-17 12:14:22"
1,"close","2021-06-17 12:15:30"
2,"open","2021-06-17 12:12:01"
...
```

While the XES and OCEL formats already fit into a well-defined structure, this is unfortunately not the case for CSV. The big problem is that the columns can be of any name, and the order of the columns can vary depending on the system they come from. The structure of CSV logs from different systems or even system modules can therefore vary greatly, making processing difficult. Nevertheless, it is extremely popular in the industry.

## 4. Introduction of a general descriptive format

When designing any system with similar expectations, it is advisable to think from the beginning that the layer providing the data should be able to serve data from multiple directions. The input side should not be limited to a single format. From a software engineering point of view, the problem of handling multiple formats is also addressed in many other areas (e.g. game development), but there is very little information on how to deal with this problem effectively within the field of process mining.

A fundamental problem is that supporting different formats requires different, format-specific implementations of the system. An important aspect is the need to separate the data service layer from the logic module that performs the business logic (e.g. prediction). Although in theory it is possible to have a business module that implements all formats and is able to interpret format-specific data, in practice this is a wrong design pattern. For in this case, the data integration logic is integrated into a module that does not have this as its main role. At the implementation level, duplicate or very similar methods are forced to appear, and the code is not clean and well maintained. Long-term development is not efficient.

An efficient solution is a general-purpose (runtime) logic format. The solution has several advantages:

- This solution allows data from different sources to appear in a specific structure
- Data validation and transformation can be performed in one place
- Any module that needs data already sees a single model, not different types of log structures

The developed general logic model allows any format to be integrated later. Figure 2. shows the model of this.

As a result, the data integration module is able to pass a validated data set to the other modules.

### 4.1. General descriptive format structure

The logical structure of this general event model structure is the following:

- **Log:** the highest logical level that represents an event log file. Set of Log traces
- **Trace:** the realization of a specific process / case. Each trace has a unique identifier (trace_id) and is made up of events.

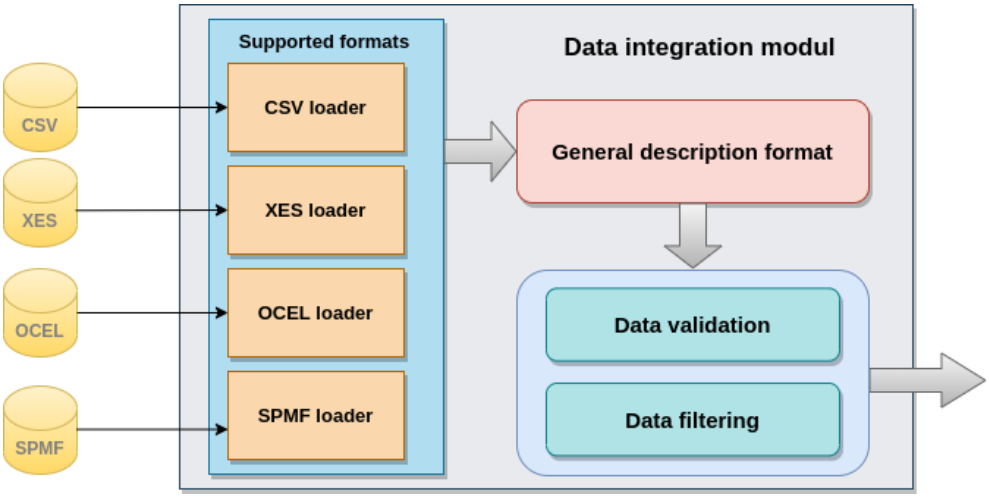


**Figure 3.** Logical structure of data integration module

- **Event:** the lowest level structure. Stores an elementary event. Minimum required data: name, time.
- **Attribute:** beside the main components it is advisable to store other useful parameters.

The event model structure is defined in Python language as below:

```
class Event:

    def __init__(self, name: str, timestamp: str):
        self.name = name
        self.timestamp = timestamp

class Trace:

    def __init__(self, trace_id, event_list=None):
        self.trace_id = trace_id

        if event_list is None:
            event_list = []

        self.event_list = event_list

    def add_event(self, event: Event):
        self.event_list.append(event)

    def print_traces(self):
        for event in self.event_list:
            logging.info(str(self.trace_id) + "-" + event.name)

    def get_number_of_events(self):
        return len(self.event_list)
```

```
class Log:

    def __init__(self, name: str, trace_list=None):
        self.name = name

        if trace_list is None:
            trace_list = []

        self.trace_list = trace_list

    def add_trace_list(self, _trace_list):
        self.trace_list.append(_trace_list)

    def add_trace(self, trace):
        self.trace_list.append(trace)

    def print_log_traces(self):
        for trace in self.trace_list:
            logging.info(trace.trace_id)
```

The current purpose of the structure is to store only the most important data. If necessary, it can be expanded as required.

## 4.2. Practical implementation of CSV loading

In the following a sample code is shown, which reads and converts a CSV log file into the generic event log structure defined above.

```
def load_trace_list_csv(file_name):
    dataset_all = pandas.read_csv(file_name)

    trace_array = []

    for index, row in dataset_all.iterrows():

        activity = row['ActivityID']
        case_id = row['CaseID']
        timestamp = row['CompleteTimestamp']

        found_case = False
        for oo in trace_array:
            if oo.trace_id == case_id:
                new_event = Event(activity, timestamp)
                oo.add_event(new_event)
                found_case = True
                break

        if not found_case:
            new_event = Event(activity, timestamp);
            new_trace = Trace(case_id)
            new_trace.add_event(new_event)
            trace_array.append(new_trace)

    log = Log(file_name)
    log.add_trace_list(trace_array)

    return log
```

## 5. MLP model based event forecasting

Of course, the role of event logs is not limited to the persistent storage of data. One of the possible uses of the data is to make forecasts. For example, in case of a more complex process: what will be the next expected step, which may offer an opportunity to automate part of the process, or a potential assistance opportunity for the case management staff.

In practice, several types of solutions can be used to predict the expected future values of atomic events. Artificial neural networks represent one of the main directions in which one of the most commonly used models is multilayer perceptron networks (MLP). Predicting the expected next element in the sequence of events is challenging, especially when working with long sequences, noisy data, multi-step predictions, and multiple input and output variables. Artificial neural networks, most notably deep learning methods, offer a promising opportunity to predict time series such as automatic learning of temporal dependence and automatic management of temporal structures such as trends and seasonality. The following figure shows the processing of logs with neural networks in general:

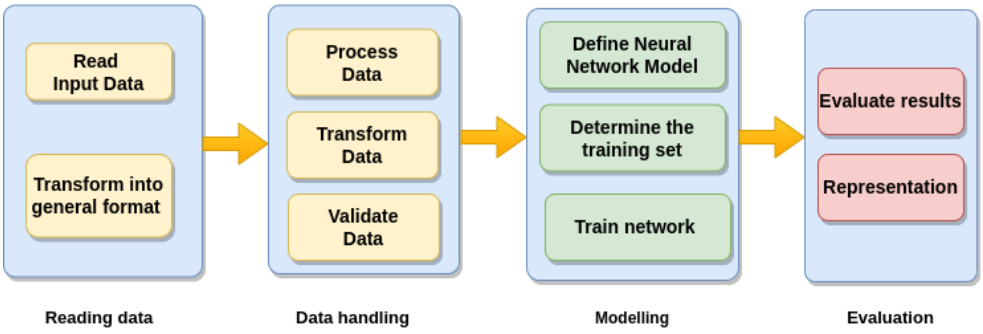


**Figure 4.** Elementary steps in MLP modeling and evaluation

### 5.1. The applied MLP model

One of the most common implementation environments for artificial intelligence-based algorithms are Python and Keras. The univariate MLP model of the forecast was implemented using these. For the development we used a standard benchmark data file available on the Internet (pdc_2016_1.xes), the experiments were performed on this (symbol set size: 18, longest sequence length: 30, number of data samples: 1000).

### *5.1.1. Data structure preparation*

Neural networks work with numerical values. In order to model a non-numerical problem with a neural network, it must be mapped to a numeric form [4]. Of course, the event prediction task also belongs to this group, because in practice, separate events are usually modeled with some kind of atomic identifier. While in practice, in a real information system, the representation of a number-based event would be appropriate in theory, but they often differ from the numerical representation due to human readability.

The available XES dataset uses letters (18 pieces) to name the events. So the first task is to map the data set to a numeric value after loading the data, which can be done by assigning a number to each event ID letter in XES (e.g. a (0), b (1), c (2), ...).

The next step after numeric mapping is to bring consecutive sequences of related events into the same format. Not all activity-related events within XES have the same number of events. However, the Keras-based MLP model expects data in slices of the same size, leaving no choice but to bring it to a uniform format. We need to define the longest pattern and adjust the sequence of events for the other activities by adding zeros from the left to zero for the longest size. Example:

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12]

[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 13, 13, 13, 13, 15, 14, 16, 1, 6, 3, 2, 4, 5, 7, 8, 10, 9, 11, 12]

### *5.1.2. Dataset slicing*

In order to be able to model the problem in a Keras environment, further data preparation is essential. Most machine learning algorithms use supervised learning. In our case, the data in the event sequences must be transformed into a suitable supervised learning format. Supervised learning is where there is an input variable (X) and an output variable (y) and uses an algorithm to learn the mapping function from input to output. A data set sequence can be transformed by using the previous steps as an input variable and the following steps as an output variable.

The series can be transformed into patterns with input and output components that can be used as part of the learning process, for example, to teach a deep learning neural network.

This is called a sliding window transformation. In this case, the window width is 3 time steps. For MLP models, Keras will expect data in this (or similar)

| X, | y |
|---|---|
| [1, 2, 3] | 4 |
| [2, 3, 4] | 5 |

**Figure 5.** Sample input sequence 1, 2, 3, 4, 5, ...

format. The loaded input dataset must therefore be converted to this format. The following Python code will do this automatically:

```
def split_input_sequence(sequence, n_steps):
X, y = list(), list()
for i in range(len(sequence)):
        end_ix = i + n_steps
        if end_ix > len(sequence)-1:
                break

        seq_x, seq_y = sequence[i:end_ix], sequence[end_ix]
        X.append(seq_x)
        y.append(seq_y)
return array(X), array(y)
```

### *5.1.3. MLP model in Keras environmen*

After the preparation of the data, the next key element is to create network model implementing univariate prediction in the Keras environment, which in our case is represented by the following:

```
def create_mlp_model(self, h):

model = tf.keras.Sequential(
    [
      tf.keras.layers.Dense(h, activation='relu', input_dim=self.dimension),
              tf.keras.layers.Dense(h),
      tf.keras.layers.Dense(len(self.c_dict) + 1, activation='softmax'),
    ]
   )
model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
return model
```

The central element of the Keras library is the *model* (*keras.models*) which is represented by the *Sequential* class. In terms of its operation, it is realized as a linear set of model layers. In the first layer of the model, the shape of the input must be specified. This is accomplished with a layer called the *Dense* type, which requires the number of *batch* and input attributes to be created. In our case, the size of the time window of the input dataset can be parameterized, which is represented by *self.dimension*. The popular ReLU[4] was used as an activation function [5]. In addition, the *Adam*[6] optimization algorithm was used to compile the model, which is the stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. The following network is created in Keras with a specific value of 100 $h$:

| Layer (type) | Output Shape | Param # |
|---|---|---|
| dense_2 (Dense) | (None, 100) | 1100 |
| dense_3 (Dense) | (None, 100) | 10100 |
| dense_4 (Dense) | (None, 19) | 1919 |

### *5.1.4. Creating the training dataset*

The last step is the automatic generation of the training samples, where the original log dataset was used to create these. The role of the *gen_ train_ data* function is to generate an appropriate teaching dataset. The function uses the loaded XES data for this process: it creates an array of input data slices of the specified size based on the value obtained in the parameter using the *create_ dataset* function, similar to the input data sample presented earlier.

```
def create_dataset(self, dataset, look_back, data_x, data_y):
        c = len(self.c_dict)
        for i in range(len(dataset) - look_back - 1):
            if dataset[i + look_back] > 0:
                a = dataset[i:(i + look_back)]
                for j in range(len(a)):
                    a[j] = a[j] / c
                b = [0 for _ in range(c + 1)]
                b[dataset[i + look_back]] = 1
                data_x.append(a)
                data_y.append(b)
        return np.array(data_x), np.array(data_y)


    def gen_train_data(self, m):
        self.m = m
        train_x, train_y = [], []
        for i in range(len(self.data)):
            self.create_dataset(self.data[i], self.m, train_x, train_y)


        return train_x, train_y
```

## 6. MLP prediction results

We can interpret the correctness of learning as a measure of evaluation. Keras has a built-in option during the teaching process to measure the "*loss*" and "*accuracy*" values generated during the teaching process, which can be used for later comparisons. The "*categorical_ crossentropy*" in the above code defines the error function in the model, which is commonly used for multi-class classification.

**Table 1.** Comparison of running results

| Epoch | Loss | Accuracy | Time |
|---|---|---|---|
| 50 | 1.2645 | 0.4345 | 13 sec |
| 200 | 1.1102 | 0.51 | 35 sec |
| 500 | 1.0078 | 0.5586 | 1:55 min |
| 1000 | 0.9360 | 0.5874 | 3:31 min |
| 2000 | 0.9132 | 0.6018 | 7:23 min |
| 4000 | 0.8642 | 0.6109 | 15:33 min |

The teaching process is long and time consuming. In the experiment, we taught with several epoch values. The epoch number is a hyperparameter that defines the number of times the teaching algorithm traverses the entire teaching pattern. Neural networks are non-deterministic systems that can even provide different outputs for the same input. For this reason, during the research work, the evaluation of one result was always determined on the basis of several different runs. The tests were performed on a Core i7-9700 3 GHz CPU Linux based operating system. The following table summarizes the loss and accuracy values for epoch values.

The diagrams below illustrate the result of learning processes with different epoch numbers, the change in the value of accuracy.

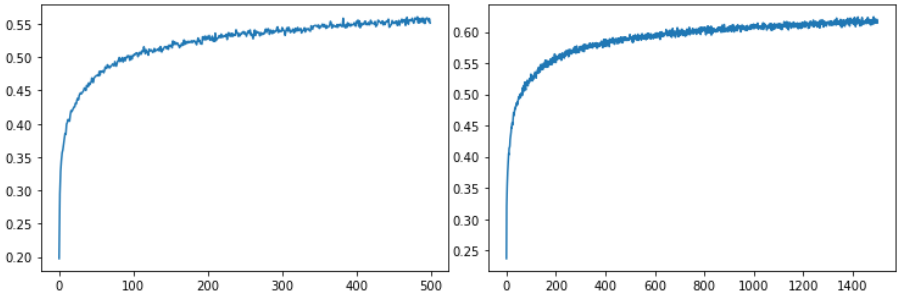


**Figure 6.** Accuracy values convergence with different epoch values

Change in value of loss function for 4000 epoch numbers:

As a result, the trained MLP model was able to perform elementary event prediction based on the samples. Efficiency meets expectations and can be further improved with additional methods.

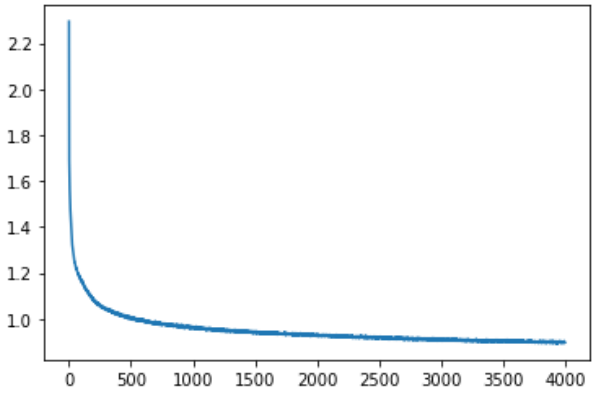


**Figure 7.** Loss function variation

## 7. Conclusion

The business processes of modern companies are becoming more and more complex, which can be supported by increasingly complex systems. Of course, as in any other area, the question of automation arises also here, which can be used to simplify complex processes in part. RPA is just such an initiative to build an automated decision support system that can solve or even predict certain steps in a complex process by logging and monitoring user activity. Due to the relatively modern nature of the area, it is not yet mature enough, on the shelf products are not available for instant integration. In this publication, we have summarized the most important requirements and characteristic problems of the field, and a general runtime model structure was presented to provide a common path for different types of log files, which provide a good basis for the implementation of such a system. Finally an event prediction MLP neural network model was introduced for an effective event prediction based on log information.

**Acknowledgement.** The described article was carried out as part of the 2020-1.1.2-PIACI-KFI-2020-00165 "ERPA - Development of Robotic Process Automation solution for heavily overloaded customer services" project implemented with the support provided from the National Research, Development and Innovation Fund of Hungary, financed under the 2020–1.1.2-PIACI KFI funding scheme.

## References


[1] Wil M. P. van der Aalst.: *Process Mining: Data Science in Action*, Springer; 2nd ed. April 26, 2016.

[2] Process and Data Science Group (PADS): *OCEL standard*, http://www.ocel-standard.org, 2022.

[3] Christian W. G.: *An Introduction to the XES Standard*, https://fluxicon.com/blog/2010/09/intro-to-xes/, 2022.

[4] Ron K.: *Practical Deep Learning with Python: A Python-Based Introduction*, No Starch Press, 25 Feb. 2021.

[5] Ruth Vang-mata: *Multilayer Perceptrons: Theory and Applications*, Nova Science Pub Inc, March 1, 2020.

[6] Ian G., Yoshua B., Aaron C., Francis B.: *Deep Learning (Adaptive Computation and Machine Learning Series*, MIT Press (Hardcover),3 Jan. 2017.

[7] João M. P. Cardoso, José Gabriel F. Coutinho, Pedro C. Diniz: *Embedded Computing for High Performance*. Efficient Mapping of Computations Using Customization, Code Transformations and Compilation, pp. 17-56., 2017.

[8] IEEE: *XES Standard*, https://xes-standard.org, 2022.

[9] Scikit-learn Online: *Supervised Neural Network Models*, https://scikit-learn.org/stable/modules/neural_networks_supervised.html, 2022.

[10] Seyedali M., Hossam F., Ibrahim A.: *Evolutionary Machine Learning Techniques: Algorithms and Applications (Algorithms for Intelligent Systems)*, Springer; 1st ed., 25 Nov. 2020. https://doi.org/10.1007/978-981-32-9990-0

[11] Marlon D., Marcello La R., Jan M., Hajo A. R.: *Fundamentals of Business Process Management*, Springer Berlin, Heidelberg, 2018.

[12] R'bigui, Hind & Cho, Chiwoon: *The state-of-the-art of business process mining challenges*, International Journal of Business Process Integration and Management, 2017. https://doi.org/10.1504/ijbpim.2017.10009731

[13] Jaswinder S.; Rajdeep B.: *A Study on Single and Multi-layer Perceptron Neural Network*, 3rd International Conference on Computing Methodologies and Communication (ICCMC), 2019. https://doi.org/10.1109/iccmc.2019.8819775

[14] Thulasi B.: *Multi-layered deep learning perceptron approach for health risk prediction*, J Big Data 7, 50, 2020. https://doi.org/10.1186/s40537-020-00316-7

[15] Mohammadreza F. S., Mozhgan V., Gyunam P., Marco P., Sebastiaan J. van Z., Wil M. P. van der A.: *Event Log Sampling for Predictive Monitoring*, In: Munoz-Gama, J., Lu, X. (eds) Process Mining Workshops. Lecture Notes in Business Information Processing, vol 433. Springer, Cham, ICPM 2021. https://doi.org/10.1007/978-3-030-98581-3_12