

Master's Thesis

Neural Computing for
Event Log Quality Improvement

Hoang Thi Cam Nguyen

Department of Management Engineering

Graduate School of UNIST

2018

Neural Computing for Event Log Quality Improvement

Hoang Thi Cam Nguyen

Department of Management Engineering

Graduate School of UNIST

Neural Computing for Event Log Quality Improvement

A thesis
submitted to the Graduate School of UNIST
in partial fulfillment of the
requirements for the degree of
Master of Science

Hoang Thi Cam Nguyen

12/15/2017 of submission

Approved by

Advisor

Marco Comuzzi

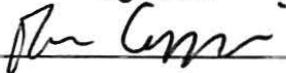
Neural Computing for Event Log Quality Improvement

Hoang Thi Cam Nguyen

This certifies that the thesis of Hoang Thi Cam Nguyen is approved.

12/15/2017 of submission

signature



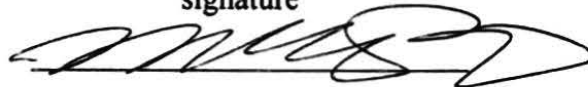
Advisor: Marco Comuzzi

signature



Changyong Lee: Thesis Committee Member #1

signature



Sungil Kim: Thesis Committee Member #2

Abstract

An event log is a vital part used for process mining such as process discovery, conformance checking or enhancement. Like any other data, the initial event logs can be too coarse resulting in severe data mining mistakes. Traditional statistical reconstruction methods work poorly with event logs, because of the complex interrelations among attributes, events and cases. As such, machine learning approaches appear more suitable for reconstructing or repairing event logs. However, there is very limited work on exploiting neural networks to do this task.

This thesis focuses on two issues that may arise in the coarse event logs, incorrect attribute values and missing attribute values. We are interested in exploring the application of different kinds of autoencoders on the task of reconstructing event logs since this architecture suits the problem of unsupervised learning, such as the ones we are considering. When repairing an event log, in fact, one cannot assume that a training set with true labels is available for model training. We also propose the techniques for preprocessing and training the event logs data. In order to provide an insight on how feasible and applicable our work is, we have carried out experiments using real-life datasets.

Regarding the first issue, we train autoencoders under purely unsupervised manner to deal with the problem of anomaly detection without using any prior knowledge of the domain. We focus on developing algorithms that can capture the general pattern and sequence aspect of the data.

In order to solve the second issue, we develop models that should not only learn the representation and underlying true distribution of the data but also be able to generate the realistic and reliable output that has the characteristic of the logs.

Keywords: Process mining, Business process, Data quality, Event log, Neural network.

Table of contents

List of figures	v
List of tables	vii
Nomenclature	ix
1 Introduction	1
1.1 Problem scenario	1
1.2 Objectives	3
1.3 Outline	4
2 Background and related work	5
2.1 Machine learning background	5
2.1.1 Feed-forward neural networks	5
2.1.2 Recurrent Neural Networks (RNNs)	6
2.1.3 Autoencoders (AEs)	8
2.2 Related work on quality of event logs	13
3 Preliminaries	17
3.1 Event log definition	17
3.2 Datasets	18
3.2.1 Artificial datasets	18
3.2.2 Real-life datasets	19
4 Multivariate anomaly detection	21
4.1 Introduction	21
4.2 Methods	21
4.3 Anomalous attribute simulation	23

4.4	Input data treatment	24
4.5	Experiments	26
4.6	Evaluation criteria	27
4.7	Results	28
4.8	Discussion	33
5	Event log reconstruction	35
5.1	Introduction	35
5.2	Methods	36
5.3	Missing attribute simulation	36
5.4	Input data treatment	38
5.5	Experiments	40
5.6	Evaluation criteria	41
5.7	Results	42
5.8	Discussion	44
6	Conclusion and future work	47
6.1	Conclusion	47
6.2	Future work	48
	References	51
	Appendix A Statistical Description and Visualization	59
	Appendix B Scatter Plot of Anomalous Time	65
	Appendix C Receiver Operating Characteristic Curve	69
	Appendix D Implementation	73

List of figures

1.1	The petrinet process model of BPI challenge 2013 dataset	1
1.2	The proposed framework for event log quality improvement	3
2.1	Feed-forward Neural Network	5
2.2	Recurrent Network architecture	6
2.3	A LSTM cell	7
2.4	Autoencoders	8
2.5	The butterfly architecture of undercomplete autoencoder with five layers	8
2.6	A standard variational autoencoder	11
2.7	Reparametrization trick	11
2.8	The architecture of a simple sequence autoencoder	12
4.1	Multivariate anomaly detection procedure	22
4.2	Event log pre-processing example	24
4.3	An example of vectorized input	25
4.4	Reconstruction error of Time attribute	29
4.5	Reconstruction error of Activity attribute	30
5.1	Event log reconstruction procedure	36
5.2	Event log pre-processing example	38
B.1	Normal and abnormal of selected activity duration of BPI 2013 log	65
B.2	Normal and abnormal of selected activity duration of BPI 2012 log	66
B.3	Normal and abnormal of selected activity duration of small log	67
B.4	Normal and abnormal of selected activity duration of small log	68
C.1	Receiver Operating Characteristic of Time attribute	70
C.2	Receiver Operating Characteristic of Time attribute	71

D.1	Multivariate Anomaly Detection: Directory structure	74
D.2	Event Log Reconstruction: Directory structure	75

List of tables

1.1	A subset of an event log	2
3.1	Configurations for log generation	19
3.2	A part of a dataset exported by Disco	20
4.1	Choices of hidden size for experiments.	27
4.2	Execution Time in multivariate anomaly detection experiments.	27
4.3	Performance of Anomalous Time Detector	31
4.4	Performance of Threshold-based Anomalous Activity Detector	32
4.5	Performance of Argmax-based Anomalous Activity Detector	32
5.1	Example of missing attribute value setting	37
5.2	Number of missing values in each dataset	37
5.3	Choices of hidden size for experiments	40
5.4	Execution Time in multivariate anomaly detection experiments.	41
5.5	Model performance for missing timestamp value reconstruction	42
5.6	Model performance for missing activity label reconstruction	43
A.1	BPI 2013 Challenge: Descriptive statistics of activity duration and frequency	59
A.2	BPI 2013 Challenge: Descriptive statistics of case duration	59
A.3	BPI 2012 Challenge: Descriptive statistics of activity duration and frequency	60
A.4	BPI 2012 Challenge: Descriptive statistics of case duration	61
A.5	Small log: Descriptive statistics of activity duration and frequency	62
A.6	Small log: Descriptive statistics of case duration	62
A.7	Large log: Descriptive statistics of activity duration and frequency	63
A.8	Large log: Descriptive statistics of case duration	63

Nomenclature

Acronyms / Abbreviations

AE	Autoencoder
ELR	Event Log Reconstruction
EM	Expectation Maximization
fpr	false positive rate
KL	Kullback-Leibler divergence
LAE	Long Short-Term Memory Autoencoder
LSTM	Long Short-Term Memory
MAD	Multivariate Anomaly Detection
MLP	Multi-Layer Perceptron
NN	Neural Network
OHC	One Hot Encode
RNN	Recurrent Neural Network
ROC	Receiver Operating Characteristic
tpr	true positive rate
VAE	Variational Autoencoder

Chapter 1

Introduction

1.1 Problem scenario

Historical information about execution of business processes can be recorded in so-called *event logs* using data produced by enterprise information systems. A record in an event log is an individual event that occurred in a particular process instance, or *case*, and includes attributes such as a case id, timestamp of occurrence, activity label, i.e., what was executed, and resources, i.e., who was in charge of execution/supervision. Event logs enable a plethora of process analyses, such as process discovery, conformance analysis, performance analysis or organisational information mining [51]. For instance, Table 1.1 shows an example event log of a loan origination process and a process model depicted in Figure 1.1 that can be mined from the event log, using traditional process discovery techniques.

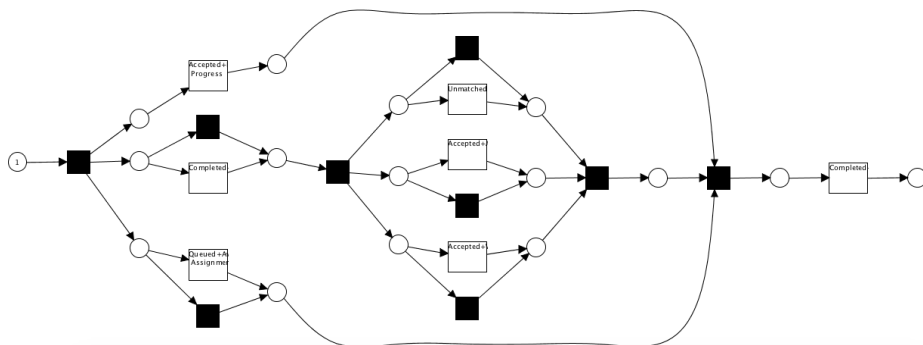


Fig. 1.1 A petri net process model of BPI challenge 2013 dataset mined by using ProM¹. This model visualizes the flow of activities between different resources within the company.

¹<http://www.promtools.org/doku.php>

Table 1.1 Event log BPI Challenge 2013. Only information of the first two cases is shown.

Case ID	Activity	Resource	Complete Timestamp	Lifecycle	Product	Resource Country
Case 1	Queued-Awaiting Assignment	Value 1	11/01/2006 23:49:42	Awaiting Assignment	PROD191	India
	Accepted-In Progress	Value 1	15/03/2012 19:53:52	In Progress	PROD191	India
	Accepted-Assigned	Value 1	15/03/2012 19:56:17	Assigned	PROD191	India
	Accepted-In Progress	Value 1	15/03/2012 20:09:05	In Progress	PROD191	India
	Completed-Closed	Value 1	15/03/2012 20:11:33	Closed	PROD191	India
Case 2	Accepted-In Progress	Value 2	07/11/2006 18:00:36	In Progress	PROD753	Sweden
	Accepted-In Progress	Value 2	07/11/2006 21:05:44	In Progress	PROD753	Sweden
	Accepted-Wait	Value 2	02/12/2009 22:24:32	Wait	PROD753	Sweden
	Accepted-In Progress	Value 2	03/09/2011 14:09:09	In Progress	PROD753	Sweden
	Accepted-In Progress	Value 3	20/01/2012 18:23:24	In Progress	PROD753	China
	Completed-Closed	Value 3	20/01/2012 18:23:27	Closed	PROD753	China

The quality of data-enabled analysis strongly depends on the quality of the underlying data used for it [2]. This holds also for event log-enabled business process analysis. For instance, in the monitoring of process KPIs, low quality information about resources, i.e., inaccurate or incomplete resource attribute values in event logs, prevents calculating an entire class of indicators related with individual resources' efficiency in executing the tasks to which they assigned.

A certain level of errors in event logs, unfortunately, is often unavoidable, particularly where manual logging is involved. Mans et al. [24], for instance, report that errors in event logs of health care processes mainly occur due to manual logging and that, among them, missing or incorrect case id and resource information occur at higher frequency than missing or abnormal timestamps.

Therefore, more research is needed to address the challenge of improving the quality of event logs, which in turn will enable higher quality analyses of business processes.

Quality of data, in general, can be improved by (i) improving the way in which data are captured while they are being generated and (ii) improving the data after they have been acquired [2]. In this thesis, we focus on (ii), that is, improving the quality of existing event logs. There are two stages to improve the quality of data that have been acquired, namely data *cleaning* and *imputation*. The former refers to identifying and removing abnormal values in a dataset, whereas the latter is the process of replacing, or reconstructing, missing values with reliable substituted values.

It should be noted that an event log has unique characteristics which make it different from other types of datasets, such as the ones traditionally used in health care or social science research. While an individual record in other datasets, e.g., medical datasets, can be considered as a complete observation of a phenomenon, an event in an event log is part of a case, which represents an actual observation, that is, a particular execution of a business process. This multi-layered structure of event logs, combined with temporal relations among

events determined by timestamps, require learning models, such as neural networks, able to learn more complex models of data. Due to these characteristics, statistical methods seem to be ineffective, which leaves the problem on how to improve the quality of business event log in a more appropriate way. This thesis aims to address this problem.

1.2 Objectives

The goal of this research is to deal with the two aforementioned data quality issues of the event log, *anomalous values* and *missing values*, that should be done during *cleaning* and *imputation* stage. The method proposed in this project uses *autoencoders*, i.e., a class of deep feed-forward neural networks that aim at reconstructing their own input after having learnt a hidden latent distribution of the input data [14]. The autoencoders developed in this paper are able to handle both continuous, e.g., timestamps, and discrete attributes, e.g., activity and resource labels. Since the three most important attributes carried by all event logs are case id, timestamp and activity, we only use these attributes in our works; and restrict the proposed model on reconstructing the values of timestamp, as an example of numerical attribute, and activity name, as an example of categorical attribute. The remaining attribute, case id, is maintained accurate and complete. We test the performance of different model variants on artificial and real event logs randomly perturbed with noise; then compare them in terms of the efficiency and computation. In addition, we also introduce the way to transform and present an event log into a numeric matrix so that it can be fed into the learning models.

The proposed framework for event log quality improvement is depicted in Figure 1.2. It is important to note that this process can be totally automated without human intervention. There are two procedures corresponding to the main goals in this thesis.

1. The first goal is *data cleaning*.
2. The second goal is *data imputation*.



Fig. 1.2 The proposed framework for event log quality improvement.

1.3 Outline

The remainder of this thesis is organised as follows:

Chapter 2 introduces the background knowledge of machine learning and gives an overview of some topics related to event quality issue.

Chapter 3 provides the preliminaries which are needed for the remainder of this thesis.

Chapter 4 identify the and suggest the approach for detecting anomalies in event logs.

Chapter 5 addresses event log quality issues and provide the solutions on how to repair incomplete event logs.

Chapter 6 summarises and identifies future research opportunities.

Chapter 2

Background and related work

2.1 Machine learning background

In this section, we would like to give explanations about the definition of the neural networks that we use in our study. This section also addresses some fundamental theoretical aspects for model learning.

2.1.1 Feed-forward neural networks

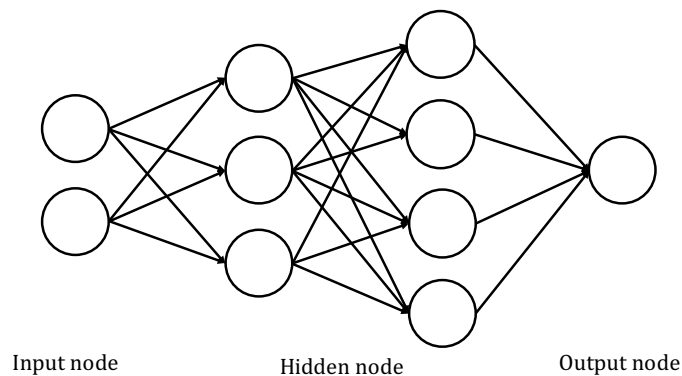


Fig. 2.1 Feed-forward Neural Network

A Feed-forward Neural Network, also called a Multi-Layer Perceptron (MLP), is an artificial neural network of which computational units interconnect in the way that there is no loop or cycle [52]. Thus, in this network, the information can be transferred from input nodes to output nodes through hidden nodes. The network can be comprised of a single layer or multiple layers of perceptron. We can view one perceptron [35] as a mathematical model

that includes a set of weights, and activation functions (linear or non-linear function). These weight values define the behavior of the whole network.

In fact, we can train the network to approximate the values for the weights. Amongst many neural network learning algorithms, *backpropagation* [38] is the most popular. Backpropagation is an iterative algorithm comprising of two phases, *forward phase* and *backward phase*. In the former phase, a training set with known output is given. Therefore, we can evaluate the network's output with the input based on the random weights. After that, in the second phase, we compute the error between the output and the desired output, then take the derivatives of the error with respect to the weight values in the network. Finally, we adjust the weight based on this gradient.

2.1.2 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks

Recurrent Neural Network [36] is a counterpart of MLP, however, the information in the network does not only travel forward. Since there is a loop inside the network that enables information to be persisted, RNNs form a powerful class of models that has an ability to solve multiple prediction and modeling tasks with sequences. The cycle formed by the connections between neurons in RNNs is visualised in Fig. 2.2.

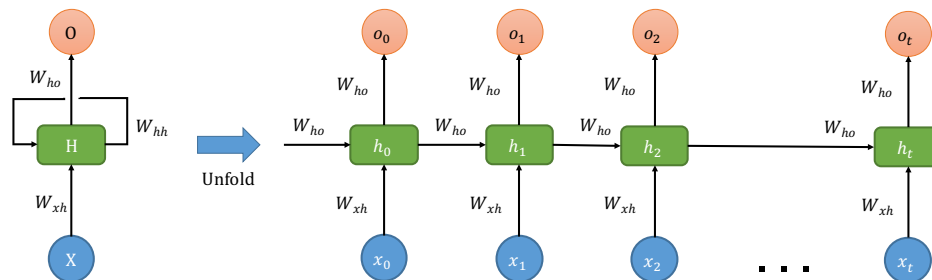


Fig. 2.2 An unrolled architecture of recurrent neural network.

We can see in the diagram that at the time t , one chunk of the network, H , receives input x_t and outputs h_t . A loop makes it possible to pass information from one time step to the next. If we unfold the RNN diagram above, we can view it as multiple normal neural networks. Each piece passes information to its successor in the network.

2.1 Machine learning background

More formally, given a sequence of input vectors $\mathbf{x}_1, \mathbf{x}_2, \dots, \mathbf{x}_T$, each in \mathbb{R}^n , hidden vectors $\mathbf{h}_1, \mathbf{h}_2, \dots, \mathbf{h}_T$ and output vectors $\mathbf{o}_1, \mathbf{o}_2, \dots, \mathbf{o}_T$ are computed using the following equations:

$$\begin{aligned} \mathbf{h}_t &= \mathbf{f}(\mathbf{W}_{xh}\mathbf{x}_t + \mathbf{W}_{hh}\mathbf{h}_{t-1} + b_h) \\ \mathbf{o}_t &= \mathbf{W}_{ho}\mathbf{h}_t + b_o \\ \forall t \in 1 : T, \end{aligned} \quad (2.1)$$

where $\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{W}_{ho}$ are the input-to-hidden, hidden-to-hidden and hidden-to-output weight matrices, respectively, b_h and b_o are the bias vectors. The undefined hidden vector h_0 can be zero or randomly initialised prior to the training procedure. Function \mathbf{f} can be any activation function.

Long Short-Term Memory (LSTM)

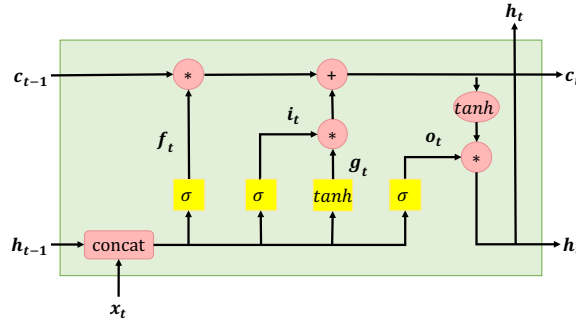


Fig. 2.3 Visualisation of Equation 2.2 for the LSTM module.

Standard RNN lacks the ability to learn long-range temporal patterns due to the vanishing or exploding gradient problems [18, 5]. To alleviate such problems, a modification of the standard RNN with "memory" cells was proposed, *long short-term memory* (LSTM) [19]. In LSTM, besides hidden and input vectors at each time step, an additional memory cell vector \mathbf{c}_t is added. The equations describing the behaviour of LSTM are presented below:

$$\begin{aligned} \mathbf{i}_t &= \sigma(\mathbf{W}^i \mathbf{x}_t + \mathbf{U}^i \mathbf{h}_{t-1} + b^i) & \mathbf{g}_t &= \tanh(\mathbf{W}^g \mathbf{x}_t + \mathbf{U}^g \mathbf{h}_{t-1} + b^g) \\ \mathbf{f}_t &= \sigma(\mathbf{W}^f \mathbf{x}_t + \mathbf{U}^f \mathbf{h}_{t-1} + b^f) & \mathbf{c}_t &= \mathbf{f}_t \odot \mathbf{c}_{t-1} + \mathbf{i}_t \odot \mathbf{g}_t \\ \mathbf{o}_t &= \sigma(\mathbf{W}^o \mathbf{x}_t + \mathbf{U}^o \mathbf{h}_{t-1} + b^o) & \mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t) \end{aligned} \quad \forall t \in 1 : T, \quad (2.2)$$

where $\mathbf{i}_t, \mathbf{f}_t, \mathbf{o}_t$ are referred to as input, forget and output gates; matrices $\mathbf{W}^i, \mathbf{W}^f, \mathbf{W}^o, \mathbf{W}^g$ map input vectors to corresponding gate vectors; matrices $\mathbf{U}^i, \mathbf{U}^f, \mathbf{U}^o, \mathbf{U}^g$ map hidden vectors to corresponding gate vectors; vectors $\mathbf{b}^i, \mathbf{b}^f, \mathbf{b}^o, \mathbf{b}^g$ are the biases, σ is a sigmoid activation

function applied element-wise, \odot denotes the Hadamard product. The undefined hidden vector h_0 and cell vector c_0 can be zero or randomly initialised prior to the training procedure.

2.1.3 Autoencoders (AEs)

Autoencoders are a class of neural networks used for unsupervised learning and as generative models [14]. Given a dataset $X = \{x^i\} \in \mathbb{R}^n$, an autoencoder tries to learn a vector X' having a distribution similar to X . This is done in two separate processes (see Fig. 2.4 for a standard representation of an autoencoder). First, a vector Z , i.e., a *code*, is formed (or *encoded*) from X to learn a hidden, or *latent* model $q_\theta(z|x)$ of the data, where θ are the weights and biases of the encoder; second, a decoder $p_\phi(x|z)$ reconstructs a vector X' having similar distribution to X from the code Z (where ϕ are the weights and biases of the decoder neural network).

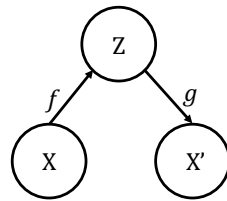


Fig. 2.4 An autoencoder comprises of two components: an encoder f and a decoder g , mappings $p_{encoder}(h|x)$ to $p_{decoder}(x|h)$

Several hidden layers can be stacked in between input and output layers in both processes, allowing a model to create a higher dimensional representation of the data. The autoencoder of Fig. 2.5, for instance, uses one hidden layer for both encoding and decoding, leading to a 5-layered deep neural network.

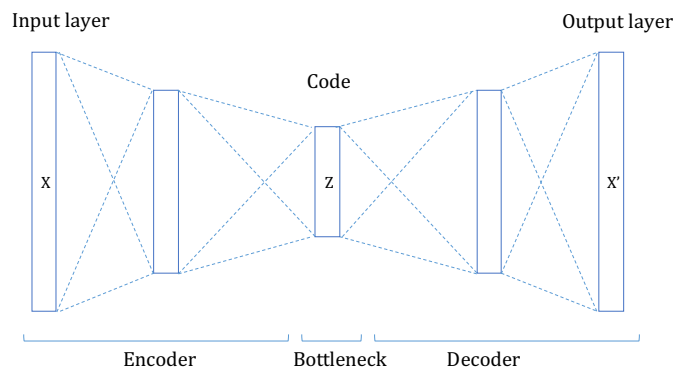


Fig. 2.5 The butterfly architecture of undercomplete autoencoder. The dimensions of input and output vectors are the same while the number of neurons of the hidden layers is smaller. The representation of the data lies in the bottleneck which is latent space.

2.1 Machine learning background

The number of hidden units in the bottleneck layer, i.e., the dimension of the code, can be lower or higher than n . If the size of the code is lower than n , then an autoencoder is forced to learn a *compressed* representation of X , by identifying a limited number of interesting features characterising the latent distribution of X . If the size of the code is higher than n , then an autoencoder can still be used to learn interesting structure in data, particularly if specific constraints on hidden units are enforced, e.g., sparsity of hidden units when processing pixels of an image.

Autoencoders initially have been used for dimensionality reduction and feature learning. The basic idea of autoencoders (with code size lower than n) is similar to the one of Principal Components Analysis (PCA) [1], that is, to project high dimensional data onto a manifold, which can represent data with a lower dimensional code. Unlike PCA, autoencoders are not restricted to linear transformations and they are able to reconstruct their output from latent variables. In addition, it can also obtain the true distribution of the data especially in the task of pattern analysis [4]. Therefore, of more interest to many researchers is the ability of the trained model that can extract useful underlying factors in order to fully understand the data and perform other tasks. Autoencoders have been successfully applied in many real-world problems, such as paraphrase detection [43] or anomaly detection [39]. Nowadays, they also find several applications as generative models, since the learnt code Z can be used to generate new datasets from the same latent distribution of input X .

The learning process in an autoencoder aims at optimising a loss function $L(X, X')$, where L is a suitable likelihood function. L can be formulated based on the ultimate objective of the training process. When the data is the output of linear function or the values are in the range of $(-\infty, \infty)$ (similar to the output in Chapter 4), we can use mean square error which can be written in Equation 2.3 to measure how close the reconstructed input x' is to the original input x .

$$L(x, x') = \frac{1}{N} \sum_i (x'_i - x_i)^2 \quad (2.3)$$

When the data resemble a vector of probabilities, i.e., values comprised between 0 and 1, as in the method proposed in Chapter 5, the loss function in Equation 2.4 is used to optimize. This loss function considers the average cross-entropy of N observations x_i in X . The cross-entropy between two probability distributions p and q over the same underlying set of observations measures the average number of information units needed to identify an observation drawn from a coding scheme optimised for the distribution q is used, rather than

the true distribution p .

$$L(X, X') = \frac{1}{N} \sum_{i=1}^N x_i \log x'_i + (1 - x_i) \log(1 - x'_i) \quad (2.4)$$

In this study, we will look into how to derive the value of Z through three models, simple autoencoder, variational autoencoder and sequential autoencoder, in order to construct the event log from the corrupted version of it.

Autoencoder

An autoencoder [37, 23, 7, 17], also called an autoassociator or a Diabolo network, is a special case of feedforward networks, however, it requires less effort during training stage.

Similar to other feedforward neural networks, the neurons in the hidden layers of autoencoder computes the weighted sums of the input neurons and biases, then passed through a non-linear transformation by some activation function $f(\cdot)$, such as Sigmoid, Tanh or Rectified Linear Unit (ReLU):

$$z = f(Wx + b) \quad (2.5)$$

Variational Autoencoder (VAE)

Variational Autoencoder [22] is a variant of autoencoder. Similar to autoencoder, the model comprises of encode and decode process which enables extracting the representation of the dataset and mapping the latent variable to the output which distribution is similar to the input. One noticeable difference between two models is how the hidden code is approximated. Variational autoencoder was proposed to perform efficient inference approximation of intractable posterior by using Stochastic Gradient Variational Bayes resulting in fast back-propagation and training process without relying on traditional expensive inference schemes such as Markov Chain Monte Carlo.

Stated more formally, variational autoencoder is trained in such a way that the probability of X is maximized in the training set according to:

$$p(x) = \int p(x|z, \theta) P(z) dz \quad (2.6)$$

This integral of the marginal likelihood is intractable leading to the intractability of the posterior $p(z|x) = p(x|z)p(z)/p(x)$, which makes it impossible to apply expectation-maximization (EM) algorithm to approximate. Hence, variational autoencoder deals with this problem by

2.1 Machine learning background

attempting to optimize the *variational lower bound* \mathcal{L} which is written as:

$$\mathcal{L}(\theta, \phi, x) = \underbrace{-D_{KL}[q_\phi(z|x)||p_\theta(z)]}_{\text{KL/regularization term}} + \underbrace{E_{q_\phi(z|x)}[\log p_\theta(x|z)]}_{\text{reconstruction term}} \quad (2.7)$$

where $q_\phi(z|x)$ is a probabilistic encoder, the unobserved variables z is a code and $p_\theta(x|z)$ is a probabilistic decode.

The structure of a standard variational autoencoder is depicted in Fig. 2.6.

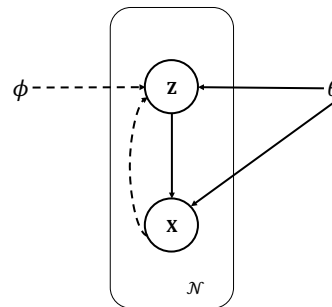


Fig. 2.6 A standard variational autoencoder. Dashed lines denote encoder network approximating $q_\phi(z|x)$, solid lines denote decoder network $p_\theta(x|z)$. The model is jointly learn the parameters ϕ and θ while training.

The first term in Equation 2.7, Kullback-Liebler (KL) divergence, plays as an additional constraint on how to construct the hidden representation z and it measures how much information is lost when using distribution q to represent distribution p . We need to find $q_\phi(z|x)$ so that $D_{KL}[q_\phi(z|x)||p_\theta(z)]$ is ideally close to zero. To optimize D_{KL} , we can adopt gradient descent procedure which uses the gradient of D_{KL} , however, we cannot take the gradient w.r.t ϕ under the integral sign. This can be solved by using *reparameterization trick* [22] that is depicted in Fig. 2.7.

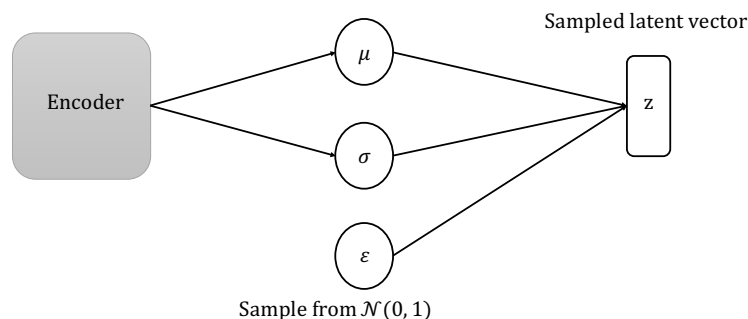


Fig. 2.7 Reparametrization trick.

The trick suggests that the sample $z \sim q_\phi(z|x)$ can be computed as $\mu + \sigma \odot \varepsilon$ where $\varepsilon \sim N(0, 1)$ and $q_\phi(z|x) \sim N(z, \mu, \sigma)$. The term μ and σ are the outputs of encoder layers. Under this setting, the formula of regularization term at the datapoint i can be written as:

$$-D_{KL}[q_\phi(z)||p_\theta(z)] = \frac{1}{2} \sum_{j=1}^J (1 + \log((\sigma_j)^2) - (\mu_j)^2 - (\sigma_j)^2) \quad (2.8)$$

where μ_j and σ_j denote the j -th element of vector μ and σ . The KL divergence can be computed and differentiated, which allows us to use back-propagation.

The second term of the *Lower Bound* \mathcal{L} is the representation for the performance of generative network (decoder), which measures the error of the outputs and can have either Bernoulli or Gaussian form. Therefore, the loss term can be formulated depends on the training objective as discussed above.

To sum up, a VAE is an autoencoder with added constraints on the encoded representation Z . In particular, the features in a code Z learnt by a VAE are forced to roughly follow a given probabilistic distribution $p(z)$, e.g., a unit gaussian distribution. This helps when VAE are used as a generative model, since new output data roughly similar to the input data can be generated by drawing values from such a distribution and pass them into the decoder part of the neural network.

RNN Encoder–Decoder

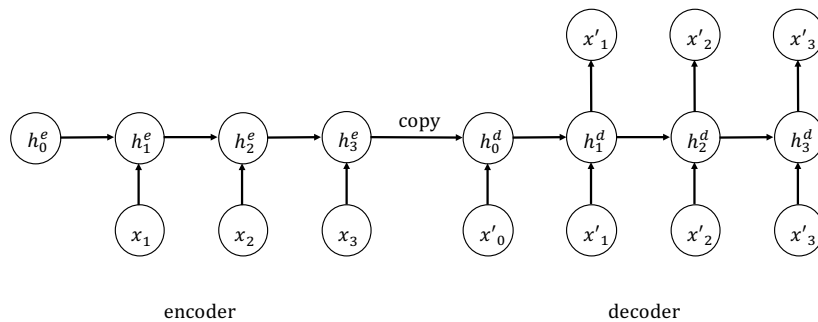


Fig. 2.8 The architecture of a simple sequence autoencoder to reconstruct output x' from input x .

RNN Encoder–Decoder, also known as Sequential Autoencoder, was first proposed by Cho et al. [13] to learn the representation of a sequence of words applied in the field of Natural Language Processing. Similar to AE and VAE, the strategy of this model is to map the input to a fixed-size vector with the encoder, then decode the vector back to the target.

2.2 Related work on quality of event logs

However, with one RNN as a encoder and another RNN as decoder, RNN Encoder–Decoder is able to estimate the conditional probability $p(y_1, \dots, y_{T'}) \mid (x_1, \dots, x_T)$ where (x_1, \dots, x_T) is the input sequence and $(y_1, \dots, y_{T'})$ is the corresponding output sequence, which can be applied in sequence learning tasks. The illustration of this model is shown in Fig. 2.8.

Different types of recurrent units, such as Long Short-Term Memory unit or Gated Recurrent unit, can be used to build a RNN Encoder–Decoder. Since the work of Tax et al. [47] has shown the empirical results in using LSTM in order to predict the continuation of a running case, in our study, we interested in using LSTM cell block for the encoder and decoder. We call this model Long Short-Term Memory Autoencoder (LAE).

2.2 Related work on quality of event logs

In this section, we review a number of recent works that have proposed in the context of data quality improvement. First, we briefly discuss the data quality issues that were identified, and then we will have a closer look at the previous approaches which were used to encounter these issues.

In Process Mining Manifestor, Aalst et al. [48] define maturity levels ranging from 1 to 5 stars as the indicators of the trustworthiness of an event log used in process mining. Following this, they introduce a guiding principle in the practice of process mining that it should only be applied to well-defined semantic logs which are at least rated as 3-star in order to avoid problematic analysis and unreliable results. Therefore, with high priority, the log should be treated by more systematic approaches.

There has been several works focusing on precisely defining and formulating data quality problem [32, 26, 28]. Different approaches lead to different taxonomies, nevertheless, their findings show that the data problem manifests in very similar ways. Data quality issues in event logs have been classified by Bose et al. [6] and, in the specific context of process mining in the health care, by Mans et al. [24]. Bose et al., in particular, have identified missing, incorrect, imprecise, and irrelevant data as type of sources of event log quality degradation. In addition, they analyze the manner which these issues appear in reality. As the result, the two issues that most frequently occur in real-life logs used in the analysis are missing event, and imprecise activity name; the issue related to timestamp is also a common cause of inconsistent result in process mining. Whereas, Mans et al. point out imprecise resource, i.e. the recorded resource refers to a specific operating room instead of the person who performed the surgery, is more likely to happen than other issues in Hospital Information System (HIS).

The work of Suriadi et al. [46] classifies a set of event log imperfection patterns that may guide the event log quality improvement phase. These patterns help understanding the sources of imperfection in an event log and, therefore, can guide the improvement of logging activities during process execution. This thesis focuses on a closely related, but different issue, that is, reconstructing suspicious and missing values in an event log that has already been acquired. In this thesis, we do not assume any knowledge about the process that has generated an event log.

The quality of event logs is strictly related with detecting noise in event logs, i.e., infrequent behaviour, and with repairing event logs. Noise is typically removed in a pre-processing phase, using frequency-based approaches [11]. As such, it can be seen in our context as a data *cleaning* activity and the task of detecting noise can be considered as novelty detection. Various state-of-art techniques in novelty detection, such as Frequentist and Bayesian approaches, information theory, and neural network, can be found in [30]. In this paper, the authors categorize all techniques into five groups that are probabilistic, distance-based, reconstruction-based, domain-based, and information-theoretic. Then, they compare the groups in terms of model complexity and application. In the work of Abhinav et al. [44], Hidden Markov Model, which is a typical example of probabilistic approach, has proven to be successfully applied to identify credit transaction fraud. In fact, this method has been extensively used in the context of anomaly detection for time series dataset since it calculates the probability of the behavior happens in the current stage based on the previous stage. In this work, we adopt a similar approach to extract the sequential information from the event log.

Aalst et al. [49] introduce the way to exploit α -algorithm in order to identify anomalous trace, however, this method requires the complete log containing only normal behaviors to be given as input. In contrast, this thesis assumes that the only information available for repairing and reconstructing values in an event log is the event log itself. In [15], Ghionna et al. proposed a two-step method to filter the exceptional trace by first extracting the normal patterns and then applying a clustering procedure under the assumption that outliers are characterized by infrequent patterns. Recently, Nolle et al. [27] have proposed to use autoencoders for denoising event logs, showing remarkable performance, albeit on artificially generated logs. Event in this case, however, noise in event logs is defined at the event level, e.g., missing or duplicated events, rather than at the event attribute level as we consider in this thesis. Our work also examines the use of reconstruction-based approach, autoencoders, to identify the anomalies. This method relies on frequent data patterns to reconstruct the noisy

input data and use the reconstructed input as a measure of normality, hence, it is susceptible to loops of anomalous data in the logs [27].

Regarding the second context, data *imputation*, various methods have been proposed to do this task in large datasets, such as deletion of observations with missing values or reconstructing data using statistical and artificial intelligence techniques. Promising results have been shown particularly in imputation of medical domain datasets [20, 40]. Beaulieu-Jones et al. [20], in the task of dealing data missing completely at random and data missing not at random, find that using bottle-neck architecture autoencoder integrated with dropout as a regularizer gives robust results at a variety of information loss. They also show that autoencoders are able to surpass KNN and SVM even when the missing data is increased. One more advantage of this algorithm is that it runs in linear time. Inspired by this paper, in our work, we use the same approach, but only focus on dealing with data missing completely at random in the logs.

For time series, Zhengping Che et al. [9] deploys a deep neural network based on Gated Recurrent Unit (GRU) [12], which is so-called GRU-D. Their model takes two missing pattern, namely masking and time interval. While the model identifies which inputs are observer or missing based on information from masking, it retrieves input observation pattern from time interval. Both representations, masking and time interval, are fed into the model so that their GRU-D is enabled to not only capture the long-term temporal dependencies of time series but take advantage of the missing patterns to enhance the predictions. They also compare the performance of RNN-based and non-RNN-based to see the benefit of using RNN on extracting sequential information. Since the time attribute in the log can be considered as time series, in this thesis, we also use RNN-based method.

Recently, Gondara et al. [16] propose a multiple-imputation model based on denoising autoencoders, which are able to process a variety of data types (continuous, categorical and mixed), distributions (random and uniform), and missing patterns (missing completely at random and missing not at random). Their model not only perform well on large datasets but also on small size ones. More remarkably, the model is capable to cope well with the situation in which users do not provide complete observations for training. Nevertheless, their model can only takes the fixed length sequence as input, whereas in this thesis, we aim to reconstruct the trace sequence of different length.

In the field of business processes, several ad-hoc methods have been proposed for repairing event logs by reconstructing missing events. Rogge-Solti et al. [34] propose a method based on stochastic Petri nets and Bayesian networks. A similar approach can then be used to improve process documentation [33]. Bayomie et al. [3] have proposed a method

to reconstruct the value of case identifier in an unlabeled event log. These approaches take a different perspective from the one considered in this thesis, since they assume that a process model is available. Missing events or attribute values can be then reconstructed by combining process knowledge with knowledge discovery techniques.

Chapter 3

Preliminaries

This section introduces the formal description of the concepts related to event logs. First, we start with the definition of the event log and its notation; then we will discuss on how we collect the datasets for our work.

3.1 Event log definition

An event log is a set of events capturing the instances of a single process. Each process instance is referred to a *case* or a *trace* and each event belonging to a single case is referred to a *task* or an *activity*. All events corresponding to a trace are chronologically ordered. An event may also carry optional additional information such as *time*, *transaction type*, *resource*, *costs*, etc. All these additional properties are considered as *attributes*. However, in this thesis, we assume the minimal information presented, in which each event has 3 attributes: case id, timestamp and activity name. The first attribute, case id, is complete while other two attributes need to be reproduced to become more reliable.

Before getting into the real-life dataset, let us introduce some required notation of event logs (an example event log is shown in Table 1.1). Let \mathcal{E} be the event universe, i.e. the set of all possible event identifiers. Events are characterised by attributes, e.g., they belong to a particular case, have a timestamp, correspond to an activity, and are executed by a particular person. Let $AN = \{a_1, \dots, a_n\}$ be a set of all possible attributes names and \mathcal{D}_{a_i} the domain of attribute a_i , i.e., the set of all possible values for the attribute a_i . Attributes can be numerical or categorical. Numerical attributes, e.g., timestamps, assume value within a certain numerical interval, that is, $\mathcal{D}_{a_i} = [v_{i,min}, v_{i,max}] \in \mathbb{R}$. Categorical attributes assume values within a given set, such as a set of activity identifiers (strings) for the activity label attribute, i.e.,

$\mathcal{D}_{a_i} = \{v_{i,1}, \dots, v_{i,K}\}$. For any event $e \in \mathcal{E}$ and attribute name $a \in AN$, $\#_a(e) \in \mathcal{D}_a$ is the value of attribute named a for event e .

Let \mathcal{D}_{id} be the set of event identifiers, \mathcal{D}_{case} the set of case identifiers, \mathcal{D}_{act} the set of activity labels, \mathcal{D}_{tst} the set of possible timestamps, and \mathcal{D}_{res} the set of possible resource identifiers. For each event $e \in \mathcal{E}$, we define a set of standard attributes:

- $\#_{id}(e) \in \mathcal{D}_{id}$ is the event identifier of e ;
- $\#_{case}(e) \in \mathcal{D}_{case}$ is the case identifier of e ;
- $\#_{tst}(e) \in \mathcal{D}_{tst}$ is the timestamp of e ;
- $\#_{act}(e) \in \mathcal{D}_{act}$ is the activity name of e ;
- $\#_{res}(e) \in \mathcal{D}_{res}$ is the resource involved in e ;

In the pre-processing phase, an event log is transformed into a suitable format to be fed into an autoencoder. The detail on how to do this will be provided in Chapter 4 and 5.

3.2 Datasets

Next, we would like to describe how we collected the datasets. In our experiment, we evaluate the performance of the proposed models by using two real-life event logs collected from an open-source website, **4TU**¹ and two artificial event logs generated by **PLG2** [8]. While deep learning offers great solutions for predictive analytics, it requires sufficient amounts of high dimensional data for more efficient and stable performance [10]. Therefore, in order to get benefit from the deep architecture model and fully extract the valuable information of data, we should choose big-size datasets. In our study, we aim to explore the performance of the models under the extreme situation as well. As the result, for evaluation purposes, we use the process models of different complexities in terms of the number of distinct activities, traces and loops in the logs.

3.2.1 Artificial datasets

We use the PLG2 tool to generate two artificial logs. PLG2 allows to create logs from process models randomly generated. In the thesis, we have considered one small and one large log, characterised by the parameters shown in the Table 3.1:

¹https://data.4tu.nl/repository/collection:event_logs_real

Table 3.1 Configurations for log generation..

Data	Number of traces	Number of distinct activities	Number of OR gateways	Number of AND gateways
Small log	2,000	14	0	4
Large log	15,000	10	2	2

The generated log can be saved as BPMN file. We import the BPMN file into Disco² and convert into CSV (Comma Separated Values).

3.2.2 Real-life datasets

From the collection of real datasets, we choose one small dataset, BPI 2013 Challenge [45], and one big dataset, BPI 2012 Challenge [50]. The event logs used in the experiments are briefly described below:

BPI 2013 Challenge: This log comprises 1,487 cases and 6,660 events with 7 different activities obtained from the incident and problem management process of Volvo IT Belgium. The case with the most number of activities comprises of 35 activities. The size of this dataset is relatively small compared to the other dataset.

BPI 2012 Challenge: This log consists of 13,087 cases and 262,200 events capturing the whole process of loan and overdraft application in Dutch Financial Institute. There are 36 different activities observed in the process and the case with the most number of activities comprises of 175 activities. In the experiments with this dataset, the training was done with 5,496 cases while the validation set and the test set consisted of both 1,832 instances.

We use the full version of the datasets in which one activity may occur many times in a trace. The standard format of the dataset file is XES (eXtensible Event Stream). We convert each data into CSV by using Disco. Case ID, complete timestamp and activity are three main information carried by both datasets. Case IDs are anonymized by selecting the anonymization option. An example of dataset which is exported by using Disco is shown in Table 3.2.

More details in terms of the statistical descriptions and visualisations of the datasets are provided in Appendix A.

²<https://fluxicon.com/disco/>

Table 3.2 Event log BPI Challenge 2013.

Case ID	Activity	Complete Timestamp
Case 1	Queued-Awaiting Assignment	11/01/2006 23:49:00
	Accepted-In Progress	15/03/2012 19:53:00
	Accepted-Assigned	15/03/2012 19:56:00
	Accepted-In Progress	15/03/2012 20:09:00
	Completed-Closed	15/03/2012 20:11:00
Case 2	Accepted-In Progress	07/11/2006 18:00:00
	Accepted-In Progress	07/11/2006 21:05:00
	Accepted-Wait	02/12/2009 22:24:00
	Accepted-In Progress	03/09/2011 14:09:00
	Accepted-In Progress	20/01/2012 18:23:00
	Completed-Closed	20/01/2012 18:23:00

Chapter 4

Multivariate anomaly detection

4.1 Introduction

Anomaly detection, also called outlier detection, is a process of identifying unusual patterns in the datasets. Unlike the standard classification task, the anomaly detection problems are mostly addressed by using the concepts in the domain of unsupervised learning, taking only the structure of the dataset into consideration. Detecting the anomalies at an early stage is critical for users since many analysts use extensive amounts of historical data and the analysis is prone to the existence of the anomalies in the dataset. Therefore, there have been various research on anomaly detection system which can be used for many practical applications such as intrusion detection [31], fraud detection [29] and data leakage prevention [42]. However, there is the lack of anomaly detection algorithms as well as available datasets that can be used for analyzing the business process containing irregular behaviors due to the fact that anomaly detection is not very frequently researched topic in the field of business process management. Therefore, in this chapter, we present the way to simulate the anomaly and propose a novel anomaly detection algorithm for detecting anomalous events without requiring any domain knowledge. The algorithms work based on the assumption that when there is sufficient or abundant normal data compared to anomalous data, the models are able to generalize the normal behaviors and treat the other data as noise.

4.2 Methods

This section presents in detail the proposed method for detecting anomalous attribute values in event logs. The steps of the proposed are shown in Fig. 4.1. A low quality event log,

i.e., with anomalous values, is taken as input. In a pre-processing phase, event logs are transformed into a format that can be fed into autoencoders. As discussed later in this section, for an autoencoder, each case in an event log represents an observation belonging to the input X . Hence, each case in an event log is transformed into a matrix of events and features that can be fed into an autoencoder. As far as model training is concerned, in this chapter we experiment with VAE, AE and LAE, which require the same type of input.

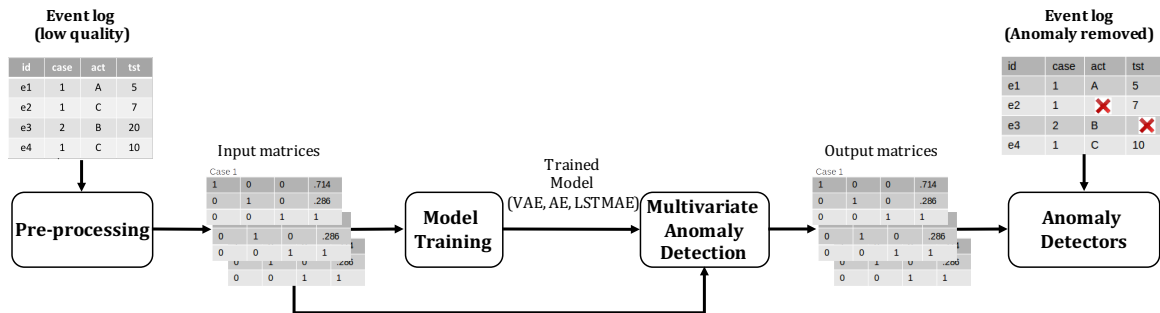


Fig. 4.1 Multivariate anomaly detection procedure.

After having the autoencoders trained, for each case x with noise, we can use the model to generate a reconstructed trace x' . We define the reconstruction error indicating the distance between the input vector and the output vector. Under the assumption that the model will reproduce the "normal" values for all attributes, reconstruction error of the normal traces will be smaller than the error of abnormal traces. As the result, we use the reconstruction error as the signal to detect anomalies in the event log. With a choice of threshold, if the reconstruction error is less than the anomaly detection threshold, the variable is normal and vice versa. It turns out that setting the threshold is critical in terms of the performance and the objective of the detector. With a low threshold, we can detect more anomalies, however, there is a trade-off that the number of false positives also increases and vice versa. We report the results by using the average reconstruction error as the threshold. The two detectors for time and activity attributes are set up as follow:

1. **Anomalous Time Detector:** For detecting anomalous time, the reconstruction error is defined as the absolute value of the difference between the input time and output time attribute. Then, the instance, which the error falls below the threshold, is classified as normal; and the instance is classified as abnormal when the error is exceeded the threshold. Since this detector relies on the choice of threshold, we call it threshold-based detector.

2. **Anomalous Activity Detector:** In order to detect anomalous activity, we set up two classifiers. One is similar to time anomalous time detector which is the threshold-based detector. However, in this case the error is defined as the mean of the absolute error between probability distribution associated with the output and input activity attribute. In the second detector, instead of using the probability distribution, we use the activity label as the signal of anomaly. The activity is considered as anomalous when its reconstructed label and input label are unmatched. We call this argmax-based detector since the input or output label is the label with maximum probability.

4.3 Anomalous attribute simulation

During the execution of a business process, most of the abnormal instances happened unexpectedly and the structure of these data cannot be observed or described in closed form, which make it impossible to learn the model that can fit the pattern of the anomalies. As a result, it is difficult to simulate a dataset with anomalies given a clean dataset. As far as we are aware, there is no appropriate labeled event log dataset of anomalies available, thus our additional challenge is generating random noisy data points in order to train and evaluate our model.

Initially, we aim to obtain the imbalanced datasets in proportions of 10% anomalies and 90% normal data. These proportions indicate that the likelihood of the anomalies exist in the observation is quite high and this inaccurate information may exacerbates the performance of trained model. However, we want to consider the extreme case in which the ratio of the anomalies is much higher than in reality. The two attributes perturbed are activity and timestamp, so the probability that the mutation occurs is set to 5% in each attribute.

As the anomaly scenario, an activity is anomalous when its label is recorded incorrectly. Therefore, we perturb 5% randomly-selected activities by replacing them with another activity which is also randomly sampled.

For the complete timestamp, since we cannot mutate this attribute directly, we make it anomalous indirectly through the duration of the activity corresponding to it. A duration of an activity is anomalous if it is larger than the sum of mean and standard deviation derived from the set of durations of that activity. With this scenario, we replace the duration by anomalous value and repeat until the number of anomalous timestamps is 5% of the total number of data points. After having the anomalous duration, we can transform back to timestamp and derive the cumulative duration within a case. The perturbed duration of selected activities in the four datasets is visualised in Appendix B.

4.4 Input data treatment

Autoencoders require as input observations coded as a matrix of numerical values. Values should also be scaled to facilitate faster convergence. If, in fact, attribute values vary across different ranges, then attributes characterised by larger ranges would be more important than other attributes characterised by smaller ranges during the learning phases. To meet this requirement, an event log is pre-processed as described in the following to transform each case into a matrix of numeric values of fixed size (see Fig. 4.2 for an example).

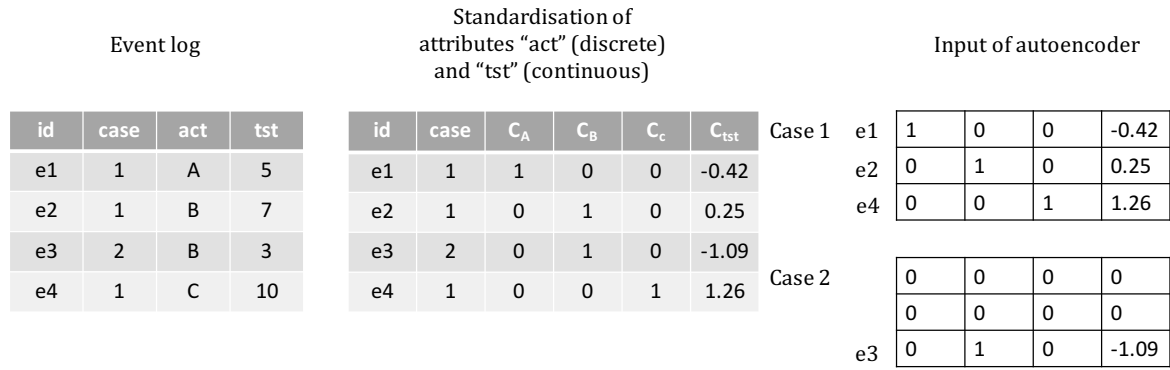


Fig. 4.2 Event log pre-processing for MAD: example

Categorical attributes (see encoding of the activity attribute in Fig. 4.2) are encoded using a one-of-K scheme. That is, for an attribute a_i a column $c_{i,k}$ in the autoencoder input matrix is created for each value $v_{i,k} \in \mathcal{D}_{a_i}$. A row is then created for each event e in an event log, with values in columns $c_{i,k}, k = 1, \dots, K$ assigned as follows:

$$c_{i,k}(e) = \begin{cases} 1 & \text{if } \#_{a_i}(e) = v_{i,k} \\ 0 & \text{otherwise} \end{cases}$$

As can be seen from the scatter plots shown in Appendix B, the anomalous data points are not far away from the mean in the distribution. In addition, there is a significant difference regarding the duration across different activities, which suggests us to scale the data by using standardisation instead of normalisation; so that the duration with small value will not be shrink. Especially, standardisation will be helpful in case the dataset has some extreme values such as the outliers, it avoids the situation in which the normal value data is scaled to a very small interval. Hence, numerical attributes $\#_{a_i}(e)$ (see encoding of timestamps Fig. 4.2) are encoded by standardising their value with the mean value L_{mean} and standard deviation value L_{std} assumed by attribute a_i in the event log. That is, for a continuous attribute a_i , a column

4.4 Input data treatment

is created such that:

$$c_i(e) = \frac{\#_{a_i}(e) - L_{mean}}{L_{std}}, \quad (4.1)$$

with $L_{mean} = \#_{a_i}(e_{mean})$, $L_{std} = \#_{a_i}(e_{std})$ and

$$e_{mean} = \frac{1}{N} \sum e', \forall e' \in \mathcal{E} \quad (4.2)$$

$$e_{std} = \sqrt{\frac{\sum (e' - e_{mean})^2}{N}}, \forall e' \in \mathcal{E} \quad (4.3)$$

Obtained rows are then grouped by case id and ordered by timestamp value. As a result, an individual case is represented by a p by q matrix, where p is the number of activities in the case and q is the number of columns resulting from the standardisation described above. Since an autoencoder requires a fixed-size matrix as input, zero-padding is applied to all cases for which $p < p_{max}$, where p_{max} is the highest number of activities in a case in an event log (see zero-padding of the first two rows for case 2 in Fig. 4.2 in which it is assumed that case 1 is the one with highest number of activities in the event log, another example is shown in Fig. 4.3). In the experiments that we conducted, the results do not depend on the position of the zero-padding rows in an input matrix.

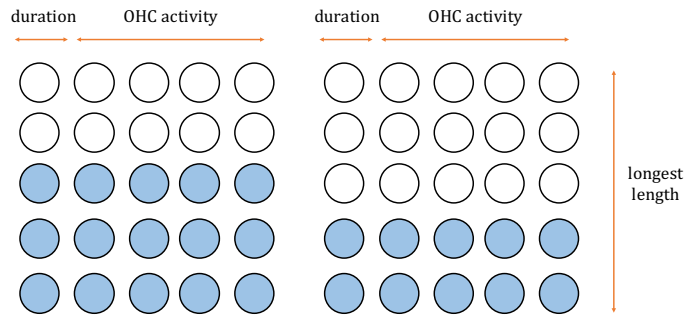


Fig. 4.3 An example of two vectorized inputs of sequence length of 4 and 3. The longest length is five. The white circles denote for zero-padding edges while the blue circles denote for numeric data points.

The objective of the model training step is to train a model that can capture the general behavior of data in an event log. Once a model has been trained, each case in an event log is reconstructed into an output matrix of elements $c'_{i,j}$ of size $p \times q$. As a result of the generating step, in the output matrix, the anomalous attributes values are replaced by a new value for numerical attributes and probabilities for the categorical attributes which are supposed to be the real values.

We introduce a masking matrix to indicate zero-padding values which should not be taken into consideration when the model computes the loss for updating weights. Elements $m_{i,j}$ of the masking matrix $M \in \mathbb{R}^{p \times q}$ are defined as:

$$m_{i,j} = \begin{cases} 0 & \text{if } c_{i,j} = 0 \text{ (0-padding)} \\ 1 & \text{otherwise} \end{cases} \quad (4.4)$$

As discussed in Section 4.2, we use distance-based method to classify normal and abnormal behavior, hence, we consider a modified mean squared error loss function, which uses the masking matrix of Eq. 4.4 before averaging across all values in an input matrix:

$$L(c_{i,j}, c'_{i,j}) = \frac{1}{p \times q} \sum_{i=1}^p \sum_{j=1}^q m_{i,j} \cdot (c_{i,j} - c'_{i,j})^2 \quad (4.5)$$

4.5 Experiments

A network with many non-linear transformation layers can have the better representation of the data, leading to the fact that in practice people are in favour of deep artificial neural networks than shallow network [25]. However, in all experiments of this thesis, we only use the shallow network to show that our model has real-world applications.

For VAE and AE, we construct a simple architecture with two hidden layers (one in encoder and the other in decoder) and one code layer. The number of neurons in hidden layers is chosen so that it is smaller than the input size. The overview of hidden layer size used for the experiments can be found in Table 4.1. It is important to note here that we can achieve better results compared to the one shown in this thesis if we consider the dimension of all layers as the extra hyperparameters and do fine tuning. We also use non-linear activation function and dropout in the hidden layers, which enables faster convergence and avoids overfitting problem. Based on guidance from the literature [14, 22, 16], internal neurons use *Tanh* and *ReLU* activation functions in AE and VAE, respectively. For LAE, we use LSTM cells which hidden size is smaller than the number of features to build the RNN encoder-decoder. The choice of hidden layer used for each dataset is given in Table 4.1.

We conducted several experiments to find good hyperparameters, with early stopping based on the validation loss. Each experiment run in 100 iterations, in each iteration, the data is loaded in batch size of 16. The models are optimised by Adam algorithm [21] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We apply adaptive learning rate with an exponential decay factor of 0.99 to adjust the learning rate after each iteration. The weights of each layer are initialized

4.6 Evaluation criteria

Table 4.1 Choices of hidden size for experiments.

Data	Input size	Hidden size (AE, VAE)	Number of features	Hidden size (LAE)
BPI 2013	280	[100, 50]	8	1
BPI 2012	6,475	[300, 100]	37	20
Small log	210	[100, 50]	15	1
Large log	88	[50, 20]	11	5

from a Xavier normal random distribution. In VAE and AE, we used dropout at 0.2 in order to avoid overfitting problem while in LAE, we use clipping to avoid vanishing/exploding gradient problem.

It is critical to choose the right learning rates since it affect the speed of convergence and the quality of model. Therefore, we tune the learning rate manually. We start the training procedure with the learning rate of 0.01 and gradually decrease by 0.001 after each training. By doing this, we find that the learning rate 0.0001 is sufficient good for the training procedure of three models. We train the model until the condition of early stopping, which the error in the validation set does not improve after 10 epochs, is met; or the maximum number of epochs is reached.

The method has been implemented in Pytorch¹. The implementation code is publicly available at². Experiments run on an Intel i7 Linux machine equipped with 16GB memory and a GeForce GTX 1080 GPU. The execution time of one epoch training is given in Table 4.2.

Table 4.2 Execution time reported in milliseconds.

Data	VAE	AE	LAE
BPI 2013	250	200	300
BPI 2012	1,900	1,500	8,200
Small log	250	200	350
Large log	1,700	1,300	2,000

4.6 Evaluation criteria

Because of the imbalance classes in the dataset, evaluation using the standard accuracy rate is not a good choice since the model may act like general classification models such

¹<https://github.com/pytorch>

²<https://github.com/hoangnguyen3892/multivariate-anomaly-detection-for-event-logs>

that it can cover of the majority examples, whereas the minority ones are misclassified frequently; leading to the high accuracy. The performance of proposed methods should be evaluated mainly based on their ability of identifying the anomalies. Therefore, we assess the performance by determining the metrics of each class label. There are 4 possible outcomes of the binary classification of a variable labeled either i - abnormal or j - normal, which are TP_i (true positive: The label i is correctly assigned to label i), FP_i (false positive: Label j is incorrectly assigned to label i), TN_i (true negative: Label j is correctly identified as label j), and FN_i (false negative: Label j is incorrectly identified as label i). The precision/specificity (π_i), recall/sensitivity (ρ_i) and F-score (f_i) of label i are defined as follow:

$$\pi_i = \frac{TP_i}{TP_i + FP_i} \quad \rho_i = \frac{TP_i}{TP_i + FN_i} \quad f_i = 2 \times \frac{\pi_i \times \rho_i}{\pi_i + \rho_i} \quad (4.6)$$

Then, the average score is the weighted mean of all classes:

$$\bar{\pi} = \frac{s_i \times \pi_i + s_j \times \pi_j}{s_i + s_j} \quad \bar{\rho} = \frac{s_i \times \rho_i + s_j \times \rho_j}{s_i + s_j} \quad \bar{f} = \frac{s_i \times f_i + s_j \times f_j}{s_i + s_j} \quad (4.7)$$

where s_i and s_j is the total number of abnormal and normal observations, respectively.

In addition to using the above metrics for evaluation, we also create visualisations for better understanding the performance of the detection models. The first visual evaluation that we use is the histogram of reconstruction errors to examine the distribution of this value across normal and abnormal data. Since we use threshold-based method described in Section 4.2 to separate the usual and unusual data points, we put two overlaid histograms of reconstruction errors of normal and abnormal data in the same plot in order to check whether the detection algorithm can distinguish between them. We also plot receiver operating characteristic (ROC) curve to illustrate two operating characteristics at different threshold settings, please refer to Appendix C for details.

4.7 Results

In this section, we evaluate the computational and the performance of three proposed models. The histograms of reconstruction error of threshold-based anomalous time and anomalous activity detector are presented graphically in Fig. 4.4 and 4.5, respectively. Table 4.3, 4.4 and 4.5 show the evaluation scores for the anomalous time detector, the threshold-based anomalous activity detector and the argmax-based anomalous activity detector in this order.

4.7 Results



Fig. 4.4 Reconstruction error of Time attribute. The blue histogram denotes for reconstruction error of normal points and the green one denotes for anomalous points.

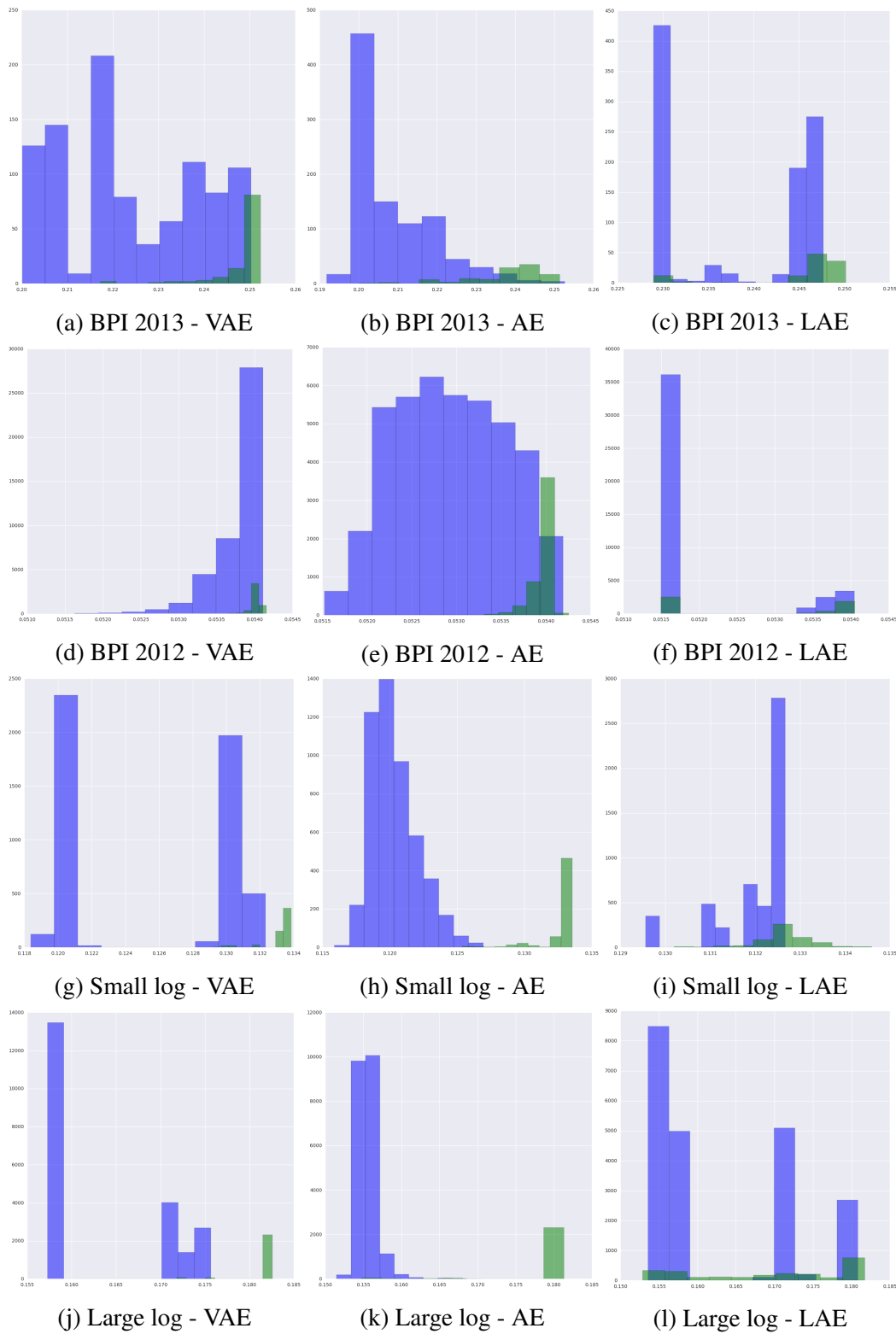


Fig. 4.5 Reconstruction error of Activity attribute. The blue histogram denotes for reconstruction error of normal points and the green one denotes for anomalous points.

It is clear that all detectors work better than a random classification model. In particular, VAE and AE are able to discriminate efficiently between normal and anomalous data in the artificial logs. Consider the real-life logs, the models do not yield significant results in BPI 2013 and unfortunately, all models fail in detecting anomaly in BPI 2012. Overall, the anomalous activity detectors show higher performance than the anomalous time detector.

Anomalous Time Detector. The Fig. 4.4g, 4.4h, 4.4j and 4.4k demonstrate that most of the anomalous attributes can be detected effectively by using VAE and AE. In contrast, from Figures 4.4a–4.4f, we can observe the detectors mostly yield nonsensical results for time attribute in two real-life logs. Based on the reconstruction error, the models are unable to distinguish abnormal time from normal ones since the error patterns are similar between two groups. When analyzing the output for the real logs, we find that the detectors encounter several distractions from the normal attribute which has a similar value. The first distraction is the loop in the process flow, which an activity may happen multiple times in a single case. Another distraction is caused by the same range of activity duration. If we take a look insight the duration of normal and abnormal activities visualised in Appendix B, we can recognise that there is no clear borderline between the duration of two groups in two real-life datasets.

Table 4.3 Performance of Anomalous Time Detector.

Data	Class	VAE			AE			LAE			Support
		Precision	Recall	F-score	Precision	Recall	F-score	Precision	Recall	F-score	
BPI 2013	Normal	0.87	0.60	0.71	0.92	0.62	0.74	0.90	0.63	0.74	956
	Anomalous	0.07	0.26	0.11	0.15	0.57	0.24	0.12	0.44	0.20	115
	Average	0.78	0.56	0.64	0.84	0.62	0.69	0.82	0.61	0.68	1,071
BPI 2012	Normal	0.92	0.64	0.75	0.93	0.70	0.80	0.92	0.67	0.78	43,368
	Anomalous	0.12	0.49	0.19	0.14	0.47	0.21	0.12	0.42	0.18	4,455
	Average	0.85	0.62	0.7	0.85	0.68	0.74	0.84	0.65	0.72	47,823
Small log	Normal	0.98	1.00	0.99	0.98	0.99	0.99	0.94	0.84	0.89	5,086
	Anomalous	1.00	0.81	0.89	0.93	0.81	0.86	0.23	0.47	0.31	514
	Average	0.98	0.98	0.98	0.98	0.99	0.99	0.88	0.81	0.83	5,600
Large log	Normal	0.98	1.00	0.99	0.98	0.99	0.99	0.95	0.69	0.80	21,888
	Anomalous	1.00	0.79	0.88	0.89	0.81	0.85	0.16	0.62	0.26	2,112
	Average	0.98	0.98	0.98	0.97	0.97	0.97	0.87	0.65	0.72	24,000

Anomalous Activity Detector. Fig. 4.5 gives an insight into the distribution of reconstruction errors of activity variable. Except the figures reported for LAE (Fig. 4.5c, 4.5f, 4.5i and 4.5l), it can be seen that the distribution of the normal and abnormal groups shown in Fig. 4.5 are quite different. For example, in Fig. 4.5a and 4.5b, the distribution of the errors of normal cases is left-skewed while errors of abnormal cases are clustered to the right. More specifically, the majority of normal data has small reconstruction error and the model reproduces the input with large error for most of abnormal data. Nevertheless, the precision is relatively low compared to the recall indicates that the false positive rate is still high. More significant positive results are presented graphically in the artificial logs, which there is no

intersection between the distribution of abnormal and normal data resulting in the ability of the detectors to clearly separate two types of attribute. In conclusion, we achieve sufficient good results in predicting anomalous activity in artificial logs while the performance in real-logs should be improved.

Table 4.4 Performance of Threshold-based Anomalous Activity Detector.

Data	Class	VAE			AE			LAE			Support
		Precision	Recall	F-score	Precision	Recall	F-score	Precision	Recall	F-score	
BPI 2013	Normal	0.99	0.80	0.89	0.97	0.94	0.97	0.97	0.50	0.66	5,015
	Anomalous	0.35	0.91	0.51	0.69	0.78	0.73	0.17	0.86	0.28	585
	Average	0.95	0.94	0.94	0.95	0.94	0.94	0.89	0.54	0.62	5,600
BPI 2012	Normal	0.97	0.87	0.92	0.96	0.97	0.97	0.93	0.84	0.89	21,552
	Anomalous	0.40	0.80	0.53	0.73	0.61	0.67	0.26	0.49	0.34	2,448
	Average	0.92	0.86	0.88	0.94	0.93	0.94	0.88	0.66	0.73	24,000
Small log	Normal	0.99	1.00	0.99	1.00	1.00	1.00	0.97	0.35	0.52	960
	Anomalous	1.00	0.89	0.94	0.97	1.00	0.99	0.14	0.90	0.24	111
	Average	0.99	0.99	0.99	1.00	1.00	1.00	0.88	0.41	0.49	1,071
Large log	Normal	0.99	1.00	1.00	1.00	1.00	1.00	0.95	0.62	0.76	42,925
	Anomalous	1.00	0.95	0.97	0.99	0.96	0.97	0.18	0.73	0.29	4,898
	Average	0.99	0.99	0.99	0.99	0.99	0.99	0.87	0.64	0.71	47,823

Table 4.5 Performance of Argmax-based Anomalous Activity Detector.

Data	Class	VAE			AE			LAE			Support
		Precision	Recall	F-score	Precision	Recall	F-score	Precision	Recall	F-score	
BPI 2013	Normal	0.99	0.71	0.83	0.96	0.99	0.97	0.97	0.49	0.65	960
	Anomalous	0.27	0.93	0.42	0.84	0.60	0.70	0.16	0.86	0.28	111
	Average	0.91	0.74	0.79	0.94	0.95	0.94	0.89	0.53	0.61	1,071
BPI 2012	Normal	0.99	0.36	0.53	1.00	0.85	0.92	0.93	0.87	0.90	42,925
	Anomalous	0.15	0.98	0.26	0.43	0.99	0.60	0.29	0.45	0.35	4,898
	Average	0.91	0.42	0.5	0.92	0.63	0.7	0.87	0.83	0.84	47,823
Small log	Normal	0.99	0.66	0.79	1.00	1.00	1.00	0.94	0.14	0.24	5,015
	Anomalous	0.25	0.97	0.40	1.00	0.98	0.99	0.11	0.93	0.20	585
	Average	0.92	0.69	0.75	1.00	1.00	1.00	0.86	0.22	0.24	5,600
Large log	Normal	1.00	0.79	0.88	0.99	1.00	1.00	0.92	0.75	0.83	21,552
	Anomalous	0.35	0.98	0.51	1.00	0.95	0.97	0.17	0.44	0.24	2,448
	Average	0.93	0.81	0.84	0.99	0.99	0.99	0.85	0.72	0.77	24,000

Table 4.4 and 4.5 summarises the performance of two different approaches used for the activity classifier, threshold and argmax. During our experiments, we find that there is no absolute superior approach which produces better performance over the other one as long as we are employing the "butterfly" architecture autoencoder. Increasing the size of hidden layers deteriorates the performance of argmax-based detector, whereas the performance of threshold-based detector still remains.

Comparison between the proposed learning models. Comparing the performance between the three models, the results show that VAE and AE outperforms LAE in all datasets, even though we expect that with the complicated computations, LAE will give competitively better results compared to the others. The scores obtained from AE is slightly higher than

those from VAE. In terms of the model complexity, the LAE architecture is more complex than VAE and AE models, resulting in more expensive computational cost (refer to Table 4.2). In addition, tuning LAE requires a lot of effort. From the experiments, we can conclude that AE provides more benefits in this context.

4.8 Discussion

In this work, we investigate the use of autoencoders for detecting anomalies in business process logs and we also integrate our anomaly detection algorithm into real-life datasets. We train the network purely unsupervised by using different types of neural layers, vanilla feed-forward layer and stochastic layer. From the experiments, it is clear that despite the simplicity of the networks and optimisation settings, the autoencoders can be exploit to learn the underlying distribution and general pattern even with the corruption of noise. No prior knowledge about the process is required during model construction stage.

Even though the proposed network has been shown some promising results in solving the problems of anomaly detection, it still faces some challenges and limitations. Firstly, the efficient of the model seems to be restricted by the way we define the boundary between anomalous and normal observations, especially in the case of real logs. When we look at the generated anomalous durations, there are many anomalous data that lies close to the normal, which makes the model hard to distinguish between them. In fact, when we consider the problem arising with the timestamp, we are more interested in fixing imprecise timestamp, i.e. the time does not reflect the true duration of the corresponding activity, than incorrect timestamp, i.e. time is not recorded in order, since it is easy to recognise the later situation. Unfortunately, this pattern is not easily simulated.

Since the model is highly sensitive to the variance introduced by noise, we can improve the performance of current model by identifying and removing easy-to-recognize anomalies before constructing the normal subspace since these anomalies may contaminate the subspace during learning stage.

Another limitation is that the evaluation used for detection algorithm is solely based on metrics and visualisations, which makes it not clear to see how effective our approach is compared to others. In order to give more convinced results, we should compare the performance of our model with other ones.

For the future investigations, we want to cope with the two limitations mentioned above. With the high priority, the lack of standard and appropriate dataset should be solved first in

order to improve the reliability and validity of our models. Then, the next step is to compare our approach with other currently-used frameworks.

Chapter 5

Event log reconstruction

5.1 Introduction

The problem with data collection is that it suffers from information loss, which is an important issue in all domains. Traditional data imputation methods, such as mean or median value substitution, are not very effective with event logs, since one cannot assume that attribute values follow a known specific distribution. Instead of using traditional methods, in this thesis, we propose adopting machine learning techniques to learn a model of data from which missing value can be then imputed. The core idea underlying the proposed method is to treat imputation as a special case of process predictive monitoring. While predictive monitoring aims at predicting the next value of an attribute in a case [47], e.g., the next activity to be executed, by looking at what happened in an event log in the past, when imputing missing value we can look at the entire event log to impute a value for missing attributes. From a methodological standpoint, predictive monitoring is a supervised learning problem, since previous history in an event log can be used to extract a set of correctly labelled observations to train a model. In reconstructing missing attribute values, however, correct observations are, by definition, not available for training a learning model. Therefore, event log reconstruction becomes a case of unsupervised learning. In this chapter, we present the way to simulate the data with missing values and propose a novel reconstruction algorithm to *impute* a substituted value that most truthfully reflects the execution of the business process that has generated the event to which the missing attribute belongs.

5.2 Methods

This section presents in detail the proposed method for imputing missing attribute values in event logs. The steps of the proposed are shown in Fig. 5.1. A low quality event log, i.e., with missing values, is taken as input. As mentioned in Section 4.2, we need to do the transformation steps in the pre-processing phase. In terms of the training model, in this chapter we also experiment with AE, VAE and LAE. After having learnt a model of the input, a model is used to reconstruct missing values in an event log using the latent distribution learnt by an autoencoder.

In a post-processing phase, for each case the continuous attribute values reconstructed by the model are then translated back into the traditional format of event logs by applying the inverse of the transformation adopted in the pre-processing phase. Meanwhile, the output corresponding to the categorical attributes is fired by *Softmax* function to get the probability of the likelihood of each label occurring.

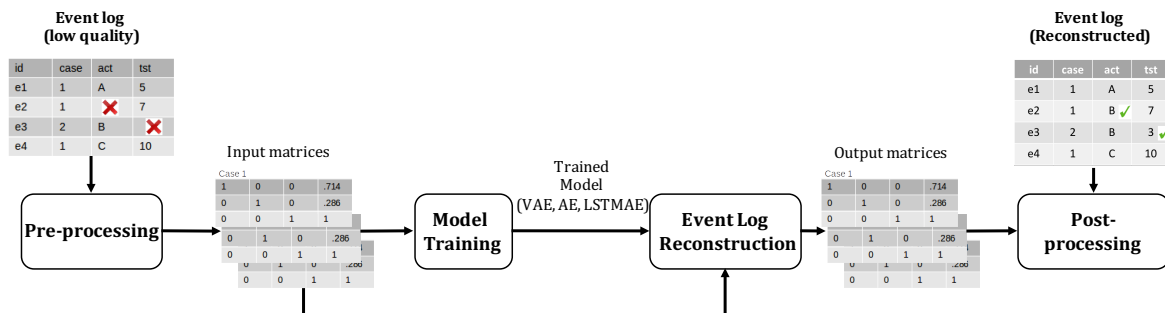


Fig. 5.1 Event log reconstruction procedure.

5.3 Missing attribute simulation

The downloaded event logs are clean and complete, so there is a need to introduce missing values to evaluate the proposed method. We restrict the proposed model to imputing the values of timestamp of completed activities, as an example of numerical attribute, and activity name, as an example of categorical attribute. The remaining attribute, i.e., case id, is maintained accurate and complete.

Since we are interested in investigating the impact of informative missingness on the performance of our algorithm, we consider 30%, 35%, 40% and 50% as ratios of attribute values missingness in event logs. To decide which values to set as missing, we randomly sample two integers x_1 and x_2 from a discrete uniform distribution, with x_1 signifying the

5.3 Missing attribute simulation

event to be set as missing, i.e., the *row* in an event log, and x_2 the attribute to be set as missing, i.e., the *column* in an event log. Then, we set the observation at the location of x_1 and x_2 as missing. Table 5.1 shows an example of setting missing values using the BPI Challenge 2012 dataset.

Table 5.1 Example of missing attribute value setting, using the BPI Challenge 2012 event log.

Case ID	Activity	Complete Timestamp
Case 1	A_SUBMITTED-COMplete	01/10/2011 07:38:45
Case 1	A_PARTLYSUBMITTED-COMplete	01/10/2011 07:38:45
Case 1	A_PREACCEPTED-COMplete	NaT
Case 1	NaN	NaT
Case 1	NaN	01/10/2011 18:36:46

In this case, x_1 is drawn from a uniform distribution in the range $[1, 5]$, that is, 5 events belong to the log, and x_2 from a uniform distribution in range $[1, 2]$, i.e., only 2 attributes can be missing (activity or timestamp). In Table 5.1, 4 missing values have been set, for $(x_1, x_2) \in \{(3, 2), (4, 1), (4, 2), (5, 1)\}$. Note that continuous missing attributes are set to NaN, whereas discrete missing attributes are set to NaT. As a result of this procedure, the propensity for an attribute value to be missing is completely random.

The number of missing values introduced for the later experiments is shown in Table 5.2.

Table 5.2 Number of missing values in each dataset

Data	Missingness ratio	Variable	Train	Validate	Test	Total
BPI 2013	30%	Time	1,371	327	317	2,015
		Activity	1,352	321	308	1,981
	35%	Time	1,577	374	399	2,350
		Activity	1,523	408	381	2,312
	40%	Time	1,799	432	425	2,656
		Activity	1,797	423	452	2,672
	50%	Time	2,226	545	535	3,306
		Activity	2,146	654	554	3,354
BPI 2012	30%	Time	47,329	16,546	14,501	78,376
		Activity	47,624	16,971	14,349	78,944
	35%	Time	55,546	19,584	16,930	92,060
		Activity	55,183	19,457	16,840	91,480
	40%	Time	63,500	22,125	19,154	104,779
		Activity	63,485	22,333	19,163	104,981
	50%	Time	79,253	27,901	23,902	131,056
		Activity	79,431	27,777	23,936	131,144

5.4 Input data treatment

Event log				Normalisation of attributes "act" (discrete) and "tst" (continuous)				Input of autoencoder							
id	case	act	tst	id	case	C_A	C_B	C_C	C_{tst}						
e1	1	A	5	e1	1	1	0	0	0	Case 1	e1	1	0	0	0
e2	1	B	7	e2	1	0	1	0	0.4		e2	0	1	0	0.4
e3	2	B	3	e3	2	0	1	0	0		e4	0	0	1	1
e4	1	C	10	e4	1	0	0	1	1	Case 2		0	0	0	0
												0	0	0	0
											e3	0	1	0	0

Fig. 5.2 Event log pre-processing for ELR: example

Before feeding the data into the training network, we need to do preprocessing steps on this dataset to make it more appropriate to our model. Similar to the steps described in Section 4.4, the categorical attributes are encoded using one-of-K scheme which K is the number of labels (see encoding of timestamps Figure 5.2).

Numerical attributes $\#_{a_i}(e)$ (see encoding of timestamps Figure 5.2) are encoded by normalising their value between the minimum value L_{min} the maximum value L_{max} assumed by attribute a_i within the case to which event e belongs. That is, for a continuous attribute a_i , a column is created such that, for each event e in an event log:

$$c_i(e) = \frac{\#_{a_i}(e) - L_{min}}{L_{max} - L_{min}}, \quad (5.1)$$

with $L_{max} = \#_{a_i}(e_{max})$, $L_{min} = \#_{a_i}(e_{min})$ and

$$e_{max} = \{e \in \mathcal{E} : \#_a(e) < \#_a(e') \wedge \#_{case}(e) = \#_{case}(e') \Rightarrow e' = e, \forall e' \in \mathcal{E}\} \quad (5.2)$$

$$e_{min} = \{e \in \mathcal{E} : \#_a(e) > \#_a(e') \wedge \#_{case}(e) = \#_{case}(e') \Rightarrow e' = e, \forall e' \in \mathcal{E}\} \quad (5.3)$$

Alternatively, numerical attributes values can be normalised using the minimum and maximum values $l_{i,min}$ and $l_{i,max}$ in \mathcal{D}_{a_i} . This choice, however, is highly sensitive to abnormally high values in the domain \mathcal{D}_{a_i} .

For both categorical and numerical attributes, missing values are encoded to the value 0.

Then, we need to transform the data within a case into matrix by doing two steps, groupby and padding, as mentioned in Section 4.4 in order to get $p \times q$ matrix as a presentation of a case. For a closer look at these steps, please refer to Figure 5.2.

The objective of the model training step is to train a model that can learn the *latent* distribution of data in an event log. Once a model has been trained, each case in an event log is reconstructed into an output matrix of elements $c'_{i,j}$ of size $p \times q$. As a result of the model learning step, in an output matrix missing attributes values (denoted by 0 in the input matrix) are mapped to valid value for numerical attributes and to probabilities for the categorical attributes. In other words, the task of reconstructing a missing value of a numerical attribute is a regression task, while reconstructing values of categorical attributes is a classification task.

To define the loss function to be optimised, during model training we introduce a masking matrix to distinguish missing values and zero-padding values from non-zero values in an input matrix. The loss function, in fact, must be designed in such a way that 0 values in an input matrix should not be learnt, since they correspond to either artificially added zero-padding values or initially missing values. Elements $m_{i,j}$ of the masking matrix $M \in \mathbb{R}^{p \times q}$ are defined as:

$$m_{i,j} = \begin{cases} 0 & \text{if } c_{i,j} = 0 \text{ (missing or 0-padding)} \\ 1 & \text{otherwise} \end{cases} \quad (5.4)$$

As introduced in Section 2.1.3, we consider a cross entropy-based loss function for autoencoders, which uses the masking matrix of Eq. 5.4 and in which cross-entropy is averaged across all values in an input matrix:

$$L(c_{i,j}, c'_{i,j}) = \frac{1}{p \times q} \sum_{i=1}^p \sum_{j=1}^q c_{i,j} \cdot m_{i,j} \log c'_{i,j} + (1 - c_{i,j}) \cdot m_{i,j} \cdot \log(1 - c'_{i,j}) \quad (5.5)$$

For VAE, in the loss function (see Eq. 2.7) we consider unit gaussian distributions for the KL regularization term and the cross-entropy function defined above for the reconstruction term.

The last step is to post-process the output matrices to reconstruct an event log in its traditional format. In this post-processing step missing values of numerical attributes are transformed into valid values in an event log by inverting Eq. 5.1, whereas an output value a_i of a categorical attribute is reconstructed by softmax activation, that is, the attribute value $a_{i,k}$ with highest probability value $c'_{i,k}$ in an output matrix is chosen as reconstructed value.

5.5 Experiments

The configurations of the autoencoders in terms of activation functions and types of hidden layers used in this experiments are set the same as those in anomaly detection in Section 4.5 except that we use sigmoid as the last activation function since it squashes the output to 0 and 1. Additional layers may be stacked in the encoding and decoding layers to increase the performance of the learning of process while deteriorating the time performance.

Following the acquisition of the appropriate dataset, we conduct experiments with three variants of autoencoders. The input data is loaded into the model with mini batch-size of 16, and the weights are initialized by Xavier uniform initializer to assist in faster convergence and avoiding local minima and exploding/vanishing gradient descent. We train the model with 100 epoch using the learning rate of 0.001, an adaptive learning rate with an exponential decay factor of 0.99. The parameters are optimized by using Adam algorithm [21] with $\beta_1 = 0.9$ and $\beta_2 = 0.999$. We monitor the training procedure by manually terminating the training when the loss on the validation set has not improved after 10 iterations.

Table 5.3 Choices of hidden size for experiments.

Data	Input size	Hidden size (AE, VAE)	Number of features	Hidden size (LAE)
BPI 2013	280	[100, 50]	8	1
BPI 2012	6,475	[300, 100]	37	20
Small log	210	[100, 50]	15	25
Large log	88	[50, 20]	11	25

Once obtaining the trained model, we let the model do the imputation for the test set. The output of regression task is evaluated by the metrics, Mean Absolute Error (MAE) and Root Mean Square Error (RMSE) while the classification task is evaluated by Accuracy Score.

After model training, we let the model do the imputation on the test set (see Fig. 5.1). The performance of the regression task (for numerical attributes) is evaluated using the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE), while for the classification task (for categorical attributes), we consider the accuracy score.

The method has been implemented in Pytorch¹. The implementation code can be found at². With the same machine described in Section 4.5, the running time is shown in Table 5.4.

¹<https://github.com/pytorch>

²<https://github.com/hoangnguyen3892/event-log-reconstruction>

Table 5.4 Execution time reported in milliseconds.

Data	VAE	AE	LAE
BPI 2013	250	200	350
BPI 2012	2,400	1,800	9,500
Small log	300	200	600
Large log	1,900	1300	2,500

5.6 Evaluation criteria

In this section, we describe baseline models used for the evaluation of our reconstruction model. As we notice, there is no existing procedure to do the imputation for both activity and complete timestamp of the event logs. Therefore, we utilize some baseline methods do to single imputation for activity imputation and complete timestamp separately.

The performance of the proposed method is evaluated against baseline imputation methods commonly used in the literature. For imputing the values of categorical attributes, i.e., activity name in our experiments, a traditional approach is to consider the most frequent observation [41]. In our experiments, the baseline BL for missing activity values is the most frequent activity name in an event log. For imputing values of numerical attributes, the median or mean value are often considered [41]. In our experiments, we consider 4 possible baselines for imputing missing timestamps values, i.e., reconstructing timestamp values using the median (BL_1) and mean (BL_2) duration of all activities in an event log, and using the median (BL_3) and mean (BL_4) duration of the activity to which a missing timestamp belongs. Note that, if both activity name and timestamp are missing for an event, then the activity name is imputed first using the proposed method, and this imputed value is used to calculate the baseline values of the missing timestamp.

Even though handling the missingness in this sequence can be solved by using median and mean substitution methods, these methods can lead to imprecise information which changes the order of activity. We ignore this problem in these dummy models and use the imputed values for computing the error metrics.

As mentioned previously in our paper, the error metrics for the time prediction are Mean Absolute Error (MAE) and Root Mean Square Error (RMSE); the error metric for activity prediction is accuracy. These metrics are computed with the values obtained from dummy models to compare with the results of our proposed models.

5.7 Results

Table 5.5 and Table 5.6 shows the time and activity imputation results on the datasets respectively. It can be seen that the our proposed models outperform the baselines in all experiments and their performance is somehow affected by the availability of the data. Furthermore, the goodness of the reconstruction models is remarkably efficient in the task of imputing missing activities. Another thing we observe during training is that it is difficult to guarantee and accelerate the convergence of training in the small-size dataset compared with the large-size dataset.

Table 5.5 Model performance for missing timestamp value reconstruction, measured by Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) in days.

Event log	Missingness	Metric	VAE	AE	LAE	BL_1	BL_2	BL_3	BL_4
BPI 2013	30%	MAE	7.23	6.70	6.69	8.68	14.35	8.97	12.90
		RMSE	13.50	13.32	13.39	16.47	19.40	16.43	18.42
	35%	MAE	7.80	7.34	7.93	15.35	19.88	15.45	18.67
		RMSE	14.38	14.21	14.32	113.84	113.94	113.83	113.85
	40%	MAE	8.52	7.70	7.53	10.14	16.79	10.93	15.34
		RMSE	15.95	13.99	13.95	18.21	23.53	18.42	22.03
50%	MAE	8.65	8.32	8.42	19.10	26.05	19.68	26.27	
	RMSE	14.92	14.73	15.48	139.66	140.02	139.62	140.14	
BPI 2012	30%	MAE	0.98	0.77	0.71	1.12	1.46	1.09	1.13
		RMSE	2.12	1.92	1.95	3.80	3.86	3.70	3.68
	35%	MAE	1.07	0.81	1.76	1.15	1.49	1.11	1.13
		RMSE	2.23	1.95	3.01	3.78	3.85	3.66	3.64
	40%	MAE	1.08	0.89	1.12	1.18	1.54	1.13	1.19
		RMSE	2.23	2.10	2.24	3.89	3.97	3.79	3.78
50%	MAE	1.17	1.06	1.05	1.43	1.85	1.39	1.43	
	RMSE	2.40	2.46	2.39	4.29	4.42	4.19	4.18	
Small log	30%	MAE	0.02	0.02	0.02	0.06	0.06	0.05	0.05
		RMSE	0.03	0.03	0.03	0.13	0.11	0.11	0.11
	35%	MAE	0.02	0.02	0.02	0.06	0.06	0.05	0.05
		RMSE	0.03	0.03	0.03	0.13	0.12	0.11	0.11
	40%	MAE	0.02	0.02	0.03	0.07	0.06	0.05	0.05
		RMSE	0.04	0.04	0.04	0.08	0.07	0.06	0.07
50%	MAE	0.03	0.03	0.03	0.13	0.12	0.11	0.11	
	RMSE	0.04	0.04	0.05	0.15	0.13	0.14	0.14	
Large log	30%	MAE	0.02	0.02	0.02	0.06	0.06	0.06	0.06
		RMSE	0.04	0.04	0.04	0.13	0.12	0.12	0.12
	35%	MAE	0.03	0.03	0.03	0.07	0.07	0.06	0.06
		RMSE	0.04	0.04	0.04	0.13	0.13	0.13	0.13
	40%	MAE	0.03	0.03	0.03	0.07	0.07	0.07	0.07
		RMSE	0.05	0.05	0.05	0.14	0.13	0.11	0.12
50%	MAE	0.04	0.04	0.04	0.08	0.08	0.08	0.07	
	RMSE	0.06	0.06	0.06	0.15	0.14	0.15	0.13	

5.7 Results

Timestamp reconstruction. Results in Table 5.5 show that, in relative terms, both VAE, AE and LAE perform better than the baselines BL_{1-4} . The mean and maximum case duration is 8.62 days and 137.22 days, respectively, for the BPI 2012 event log and 178.88 days and 2,254.85 days, respectively, for the BPI 2013 event log. The reported numbers in small log are both 0.33 days; in large log are 0.25 days. Statistical description of case duration can be found in Appendix A.

The performance of the proposed method on the BPI 2012 event log is rather stable, with MAE and RMSE not exceeding 13.6% (0.85%) and 27.8% (1.75%) of mean (maximum) case duration, respectively. The performance in respect of the BPI 2013 event log is substantially less stable. The high variability of results for the BPI 2013 event log is due to the distribution of activity durations. Activities in this event log tend to have very short or very long duration, which makes it difficult to learn the time characteristics in the logs. This is also supported by the noticeable gap between mean (BL_1, BL_3) and median (BL_2, BL_4) imputation for this event log. Mean values are in fact affected by extreme values or outliers in the dataset, and therefore achieves worse results. On the small log, MAE and RMSE are approximately 9.1% and 12.1% of mean case duration, while these numbers on the large log are 16% and 24%, respectively. Comparing to the performance of model in the real and artificial log, we find that the model can achieve more stable result in the later dataset. This is due to the fact that there are less disturbances and distractions from noise in the artificial logs.

Table 5.6 Model performance for missing activity label reconstruction, measured by accuracy.

Data	Missingness	VAE	AE	LAE	BL
BPI 2013	30%	73.05%	78.57%	74.35%	44.16%
	35%	74.80%	75.33%	73.75%	46.46%
	40%	73.70%	78.76%	76.55%	44.47%
	50%	71.48%	76.33%	72.02%	47.11%
BPI 2012	30%	69.05%	79.19%	48.81%	10.77%
	35%	64.88%	78.69%	27.93%	10.32%
	40%	64.33%	75.92%	31.93%	10.28%
	50%	60.78%	74.90%	36.95%	10.17%
Small log	30%	94.66%	96.36%	61.83%	5.22%
	35%	94.07%	96.90%	67.44%	6.25%
	40%	91.18%	95.41%	66.02%	7.29%
	50%	90.11%	93.74%	65.83%	7.12%
Large log	30%	87.14%	87.69%	83.04%	11.84%
	35%	86.72%	87.02%	82.86%	12.55%
	40%	86.38%	86.84%	82.44%	12.32%
	50%	84.64%	85.20%	82.04%	12.32%

Activity reconstruction. Results in Table 5.6 show a remarkable efficiency of the proposed method to reconstruct missing activity names. We can also observe that the performance of reconstructing missing activity names is more stable compared to missing timestamp reconstruction, even under high levels of information missingness. The model is able to impute missing values efficiently with higher accuracy than baselines from the very first iterations. This may be due to the fact that a sequence of activities in an event log tend to follow a particular pattern determined by the process control flow. This pattern can be learned by our model during the training process, which helps improving the accuracy of activity imputation.

Effect of missingness ratio. As the number of missing attribute values increases, the performance of the proposed reconstruction models deteriorates. However, the missingness ratio does not appear to have a large effect on the activity name imputation performance (see Table 5.6). As remarked before, this may be due to the patterns of sequence of activities. Once these control flow structures have been learned by a model, they easily can be used to reliably impute missing values. It seems, therefore, that the performance of the model should be evaluated in the future in respect of the complexity of a process model control flow. Note that the results do not vary extensively in the baseline methods using the mean and median values. This is because we only consider completely random missing values, which do not have a significant impact on average values calculate from the dataset.

Comparison between the proposed learning models. Overall, AE seems to perform better than VAE and LAE in most scenarios. In addition, the two later models converge slower than AE under the same settings. This should not surprise, since VAE and LAE are by definition better suited to *generative* use cases, i.e., when the objective is to generate new datasets X' with a similar distribution to the input X , whereas AEs suit better the use case of exact reconstruction of the input dataset X , through the steps of encoding and decoding. The problem considered in this paper is more similar to the latter one, since we aim at reconstructing exactly a set of missing values in an event log.

5.8 Discussion

This paper has presented a method to reconstruct missing attribute values in event logs. This increases the event logs quality by reducing the number of missing values, which in turn enables higher quality business process analysis. The method proposed uses autoencoders, a special class of feed-forward deep neural networks that aim at reconstructing their own input. Even though the performance of reconstructing timestamp values is unstable, especially in

the case of the small-sized dataset, we have showed that autoencoders give better results than baseline imputation methods when applied to both real-life and artificial event logs.

The proposed method has been evaluated for imputing the values of missing timestamps and activity labels. However, it can be generalised to other variables that typically belong to an event log. For instance, cost can be considered as a numerical variable, whereas resource identifier is a categorical variable that can be handled similarly to activity names. Given its central role in uniquely identifying cases, imputing the value of missing *case id* values is more challenging and deserves the development of ad-hoc methods.

The work presented has several limitations. First, the bias introduced by the distribution of the values of timestamps leads to poor imputation of timestamps when compared to activity names. Also, the proposed method only considers two variables for training and it can be improved by considering other existing information in event logs or by extracting more features. Addressing these limitation is a direction for future work. Moreover, future research should also look beyond improving event log quality after they have been acquired, by considering how a process logging infrastructure can be instrumented with data quality controls. Another interesting avenue for future research concerns investigating the impact of control flow complexity on the efficiency of reconstructing the values of activity labels. Finally, we also aim at evaluating the actual impact of event log quality improvement on the results of process mining analysis.

Chapter 6

Conclusion and future work

6.1 Conclusion

This thesis presents a two-step procedure using autoencoders for unsupervised representation learning in order to solve the problem of the event log quality. In general, autoencoders compress the pattern of data in the event log to a low-dimensional space which is later decompressed to reconstruct the true data distribution. Then, the recovered distribution can be used to recognise the noisy values, which is applied into *data cleaning* step, or reproduce the missing values, which is applied into *data imputation* step.

As demonstrated by experiments in Chapter 4, the proposed methods work efficiently in the generated logs. However, the results obtained in real-life event logs are not as good as the ones obtained in artificial logs. This is because there is no clear borderline between simulated anomalies and normal data. In other experiments shown in Chapter 5, all models have been shown to achieve good results in imputing the missing attributes. In fact, performing this task is easier than the previous since noise is removed and the model only learn by the correct data. Among the methods mentioned in this thesis, AE appears to be more useful in applying to this context.

In conclusion, we have shown that our proposed models outperform traditional approach in both real-life and artificial event logs. Furthermore, the method can be generalised and extended to other variables. For instance, cost and resource can be treated as a numerical and categorical variable, respectively. Finally, this work is an evidence that neural networks is applicable in the field of business process.

6.2 Future work

For the future investigations corresponding to the first sub-procedure, data cleaning, we should explore how to simulate anomalies more appropriate in the real-life logs. Regarding data imputation, it is worth to investigate more the distribution of time value and look insight the process logging infrastructure. Aside from those, in the future work, we would like to check the conformance of our novel framework in an end-to-end experiment in which we should check to what extent the models can enhance the quality of further analysis.

Publications

The results of this thesis will be published in:

1. A paper titled "Event Log Reconstruction using Autoencoders" submitted to CAiSE 2018¹.
2. A journal paper titled "Neural computing to improve event log quality" currently under preparation for journal submission (expected in January 2018).

¹<https://caise2018.ut.ee/>

References

- [1] Abdi, H. and Williams, L. J. (2010). Principal component analysis. *Wiley interdisciplinary reviews: computational statistics*, 2(4):433–459.
- [2] Batini, C., Cappiello, C., Francalanci, C., and Maurino, A. (2009). Methodologies for data quality assessment and improvement. *ACM computing surveys (CSUR)*, 41(3):16.
- [3] Bayomie, D., Helal, I. M., Awad, A., Ezat, E., and ElBastawissi, A. (2015). Deducing case ids for unlabeled event logs. In *Business Process Management Workshops*, pages 242–254. Springer.
- [4] Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.
- [5] Bengio, Y., Simard, P. Y., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Trans. Neural Networks*, 5(2):157–166.
- [6] Bose, R. J. C., Mans, R. S., and van der Aalst, W. M. (2013). Wanna improve process mining results? In *Computational Intelligence and Data Mining (CIDM), 2013 IEEE Symposium on*, pages 127–134. IEEE.
- [7] Bourlard, H. and Kamp, Y. (1988). Auto-association by multilayer perceptrons and singular value decomposition. *Biol. Cybern.*, 59(4-5):291–294.
- [8] Burattin, A. (2015). PLG2: multiperspective processes randomization and simulation for online and offline settings. *CoRR*, abs/1506.08415.
- [9] Che, Z., Purushotham, S., Cho, K., Sontag, D., and Liu, Y. (2016). Recurrent neural networks for multivariate time series with missing values. *arXiv preprint arXiv:1606.01865*.
- [10] Chen, X. W. and Lin, X. (2014). Big data deep learning: Challenges and perspectives. *IEEE Access*, 2:514–525.
- [11] Cheng, H.-J. and Kumar, A. (2015). Process mining on noisy logs—can log sanitization help to improve performance? *Decision Support Systems*, 79:138–149.
- [12] Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder-decoder approaches. *CoRR*, abs/1409.1259.
- [13] Cho, K., van Merriënboer, B., Gülçehre, Ç., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078.

- [14] Doersch, C. (2016). Tutorial on Variational Autoencoders. *Arxiv preprint*.
- [15] Ghionna, L., Greco, G., Guzzo, A., and Pontieri, L. (2008). Outlier detection techniques for process mining applications. In *Proceedings of the 17th International Conference on Foundations of Intelligent Systems, ISMIS'08*, pages 150–159, Berlin, Heidelberg. Springer-Verlag.
- [16] Gondara, L. and Wang, K. (2017). Multiple imputation using deep denoising autoencoders. *arXiv preprint arXiv:1705.02737*.
- [17] Hinton, G. E. and Zemel, R. S. (1994). Autoencoders, minimum description length and helmholtz free energy. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems 6*, pages 3–10. Morgan-Kaufmann.
- [18] Hochreiter, S. (1991). Untersuchungen zu dynamischen neuronalen netzen. *Diploma, Technische Universität München*, page 91.
- [19] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- [20] K. Beaulieu-Jones, B. and Moore, J. (2016). Missing data imputation in the electronic health record using deeply learned autoencoders. 22:207–218.
- [21] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980.
- [22] Kingma, D. P. and Welling, M. (2013). Auto-Encoding Variational Bayes. *ArXiv e-prints*.
- [23] Lecun, Y. (1987). *PhD thesis: Modeles connexionnistes de l'apprentissage (connectionist learning models)*. Universite P. et M. Curie (Paris 6).
- [24] Mans, R. S., van der Aalst, W. M., Vanwersch, R. J., and Moleman, A. J. (2013). Process mining in healthcare: Data challenges when answering frequently posed questions. In *Process Support and Knowledge Representation in Health Care*, pages 140–153. Springer.
- [25] Montúfar, G., Pascanu, R., Cho, K., and Bengio, Y. (2014). On the Number of Linear Regions of Deep Neural Networks. *ArXiv e-prints*.
- [26] Müller, H. and Freytag, J. (2005). *Problems, Methods, and Challenges in Comprehensive Data Cleansing*. Informatik-Berichte // Institut für Informatik, Humboldt Universität zu Berlin. Humboldt-Univ. zu Berlin.
- [27] Nolle, T., Seeliger, A., and Mühlhäuser, M. (2016). Unsupervised anomaly detection in noisy business process event logs using denoising autoencoders. In *International Conference on Discovery Science*, pages 442–456. Springer.
- [28] Oliveira, P., Rodrigues, F., and Henriques, P. R. (2005). A formal definition of data quality problems. In Naumann, F., Gertz, M., and Madnick, S. E., editors, *IQ*. MIT.
- [29] Phua, C., Lee, V., Smith-Miles, K., and Gayler, R. (2013). A comprehensive survey of data mining-based fraud detection research (bibliography).

- [30] Pimentel, M. A., Clifton, D. A., Clifton, L., and Tarassenko, L. (2014). A review of novelty detection. *Signal Processing*, 99(Supplement C):215 – 249.
- [31] Portnoy, L., Eskin, E., and Stolfo, S. (2001). Intrusion detection with unlabeled data using clustering.
- [32] Rahm, E. and Do, H. H. (2000). Data cleaning: Problems and current approaches. *IEEE Data Eng. Bull.*, 23(4):3–13.
- [33] Rogge-Solti, A., Mans, R. S., van der Aalst, W. M., and Weske, M. (2013a). Improving documentation by repairing event logs. In *IFIP Working Conference on The Practice of Enterprise Modeling*, pages 129–144. Springer.
- [34] Rogge-Solti, A., Mans, R. S., van der Aalst, W. M., and Weske, M. (2013b). Repairing event logs using timed process models. In *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, pages 705–708. Springer.
- [35] Rosenblatt, F. (1962). *Principles of neurodynamics: perceptrons and the theory of brain mechanisms*. Report (Cornell Aeronautical Laboratory). Spartan Books.
- [36] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986a). Learning representations by back-propagating errors. *Nature*, 323(6088):533–536.
- [37] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1986b). Parallel distributed processing: Explorations in the microstructure of cognition, vol. 1. chapter Learning Internal Representations by Error Propagation, pages 318–362. MIT Press, Cambridge, MA, USA.
- [38] Rumelhart, D. E., Hinton, G. E., and Williams, R. J. (1988). Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA.
- [39] Sakurada, M. and Yairi, T. (2014). Anomaly detection using autoencoders with non-linear dimensionality reduction. In *Proceedings of the MLSDA 2014 2Nd Workshop on Machine Learning for Sensory Data Analysis*, MLSDA'14, pages 4:4–4:11, New York, NY, USA. ACM.
- [40] Shah, A. D., Bartlett, J. W., Carpenter, J., Nicholas, O., and Hemingway, H. (2014a). Comparison of random forest and parametric imputation models for imputing missing data using mice: A caliber study. *American Journal of Epidemiology*, 179(6):764–774.
- [41] Shah, A. D., Bartlett, J. W., Carpenter, J., Nicholas, O., and Hemingway, H. (2014b). Comparison of random forest and parametric imputation models for imputing missing data using mice: a caliber study. *American Journal of Epidemiology*, 179(6):764–774.
- [42] Sigholm, J. and Raciti, M. (2012). Best-effort data leakage prevention in inter-organizational tactical manets. In *MILCOM 2012 - 2012 IEEE Military Communications Conference*, pages 1–7.
- [43] Socher, R., Huang, E. H., Pennin, J., Manning, C. D., and Ng, A. Y. (2011). Dynamic pooling and unfolding recursive autoencoders for paraphrase detection. In Shawe-Taylor, J., Zemel, R. S., Bartlett, P. L., Pereira, F., and Weinberger, K. Q., editors, *Advances in Neural Information Processing Systems 24*, pages 801–809. Curran Associates, Inc.

- [44] Srivastava, A., Kundu, A., Sural, S., and Majumdar, A. (2008). Credit card fraud detection using hidden markov model. *IEEE Transactions on dependable and secure computing*, 5(1):37–48.
- [45] Steeman, W. (2013). Bpi challenge 2013.
- [46] Suriadi, S., Andrews, R., ter Hofstede, A. H., and Wynn, M. T. (2017). Event log imperfection patterns for process mining: Towards a systematic approach to cleaning event logs. *Information Systems*, 64:132–150.
- [47] Tax, N., Verenich, I., La Rosa, M., and Dumas, M. (2017). Predictive business process monitoring with lstm neural networks. In *International Conference on Advanced Information Systems Engineering*, pages 477–492. Springer.
- [48] van der Aalst, W., Adriansyah, A., de Medeiros, A. K. A., Arcieri, F., Baier, T., Blicke, T., Bose, J. C., van den Brand, P., Brandtjen, R., Buijs, J., Burattin, A., Carmona, J., Castellanos, M., Claes, J., Cook, J., Costantini, N., Curbera, F., Damiani, E., de Leoni, M., Delias, P., van Dongen, B. F., Dumas, M., Dustdar, S., Fahland, D., Ferreira, D. R., Gaaloul, W., van Geffen, F., Goel, S., Günther, C., Guzzo, A., Harmon, P., ter Hofstede, A., Hoogland, J., Ingvaldsen, J. E., Kato, K., Kuhn, R., Kumar, A., La Rosa, M., Maggi, F., Malerba, D., Mans, R. S., Manuel, A., McCreesh, M., Mello, P., Mendling, J., Montali, M., Motahari-Nezhad, H. R., zur Muehlen, M., Munoz-Gama, J., Pontieri, L., Ribeiro, J., Rozinat, A., Seguel Pérez, H., Seguel Pérez, R., Sepúlveda, M., Sinur, J., Soffer, P., Song, M., Sperduti, A., Stilo, G., Stoel, C., Swenson, K., Talamo, M., Tan, W., Turner, C., Vanthienen, J., Varvaressos, G., Verbeek, E., Verdonk, M., Vigo, R., Wang, J., Weber, B., Weidlich, M., Weijters, T., Wen, L., Westergaard, M., and Wynn, M. (2012). *Process Mining Manifesto*, pages 169–194. Springer Berlin Heidelberg, Berlin, Heidelberg.
- [49] van der Aalst, W. and de Medeiros, A. (2005). Process mining and security: Detecting anomalous process executions and checking process conformance. *Electronic Notes in Theoretical Computer Science*, 121(Supplement C):3 – 21. Proceedings of the 2nd International Workshop on Security Issues with Petri Nets and other Computational Models (WISP 2004).
- [50] Van Dongen, B. (2012). Bpi challenge 2012.
- [51] van Eck, M. L., Lu, X., Leemans, S. J., and van der Aalst, W. M. (2015). Pm²: A process mining project methodology. In *International Conference on Advanced Information Systems Engineering*, pages 297–313. Springer.
- [52] Zell, A. (1994). Simulation neuronaler netze.

I would like to dedicate this thesis to my family for their support and encouragement.

Acknowledgements

I would like to thank many people for contributing to this work.

First of all, I am deeply grateful to my supervisor, Professor Marco Comuzzi, who has been the most reliable supervisor and given me the freedom to do what I am interested in. Without his inspirations, insights and supports, I could not have this work done.

I would like to thank Mr. Hoang Minh Le for being so responsive to all of my questions regarding technical knowledge, and providing me the computing and software facility to run all of the experiments in this thesis.

I would like to thank Mr. Du Phan for creating study group, getting me interested in machine learning. I wish him success in his future endeavors.

I would like to thank all of my friends in Korea who have made my stay here more memorable and meaningful. I am also grateful to my friends back home. They have been always available to keep me strong to overcome the obstacles in my path.

And last but not least, I would like to thank my family for their encouragement, sacrifices and understanding.

Appendix A

Statistical Description and Visualization

BPI challenge 2013

Table A.1 BPI 2013 Challenge: Descriptive statistics of activity duration and frequency.

Activity	μ	σ	Frequency
Completed-Cancelled	28,252.33	48,530.94	3
Queued-Awaiting Assignment	1,000,673.29	4,089,422.39	875
Completed-Closed	4,487,671.02	9,756,199.10	1,565
Accepted-Wait	3,474,376.57	9,675,577.78	527
Accepted-Assigned	2,284,207.51	7,115,575.88	614
Accepted-In Progress	3,862,876.81	11,479,844.40	3,066
Unmatched-Unmatched	659,730.10	1,613,426.68	10

Table A.2 BPI 2013 Challenge: Descriptive statistics of case duration.

Stat	Value in seconds	Value in days
Mean	15,455,650.00	178.88
Standard Deviation	20,550,850.00	237.86
Min	-	-
25th percentile	2,063,427.00	23.88
50th percentile	7,086,718.00	82.02
75th percentile	21,441,580.00	248.17
Max	194,818,900.00	2,254.85

BPI challenge 2012

Table A.3 BPI 2012 Challenge: Descriptive statistics of activity duration and frequency.

Activity	μ	σ	Frequency
A_REGISTERED-COMplete	209.03	5,203.87	2,246
O_ACCEPTED-COMplete	488.71	721.68	5,113
W_Valideren aanvraag-SCHEDULE	62.7	211.2	5,023
W_Afhandelen leads-START	15,560.8	28,078.36	5,897
O_DECLINED-COMplete	524.37	799.24	802
A_PREACCEPTED-COMplete	85.19	151.96	7,367
O_SELECTED-COMplete	134.32	249.12	7,030
A_CANCELLED-COMplete	22,143.71	188,709.42	2,807
W_Beoordelen fraude-SCHEDULE	80.33	152.42	124
W_Afhandelen leads-SCHEDULE	27.57	19.83	4,771
W_Completeren aanvraag-SCHEDULE	0.6	5.15	7,371
W_Nabellen offertes-COMplete	12,720.04	115,166.4	22,976
O_CREATED-COMplete	4.29	11.58	7,030
A_DECLINED-COMplete	195.85	366.31	7,635
W_Beoordelen fraude-COMplete	546.16	5,785.19	270
A_ACTIVATED-COMplete	82.17	328.19	2,246
W_Valideren aanvraag-START	112,510.23	155,271.12	7,891
W_Completeren aanvraag-COMplete	17,090.33	166,754.11	23,967
A_PARTLYSUBMITTED-COMplete	0.58	1.24	13,087
W_Beoordelen fraude-START	61,002.15	100,268.84	270
A_FINALIZED-COMplete	95.1	160.53	5,015
A_ACCEPTED-COMplete	418.45	296.49	2,243
W_Afhandelen leads-COMplete	872.77	9,844.51	5,898
W_Nabellen offertes-START	264,853.48	287,348.28	22,406
O_SENT-COMplete	0.06	0.12	7,030
W_Valideren aanvraag-COMplete	594.5	6,170.41	7,895
A_SUBMITTED-COMplete	-	-	13,087
W_Nabellen incomplete dossiers-SCHEDULE	1,095.67	989.65	2,383
A_APPROVED-COMplete	222.82	672.36	2,246
W_Nabellen offertes-SCHEDULE	0.21	0.29	6,634
W_Wijzigen contractgegevens-SCHEDULE	1,013,819.24	2,542,798.47	12
O_SENT_BACK-COMplete	47.27	97.84	3,454
W_Completeren aanvraag-START	60,638.93	185,331.21	23,512
O_CANCELLED-COMplete	5,213.6	79,518.84	3,655
W_Nabellen incomplete dossiers-COMplete	737.92	8,291.54	11,407
W_Nabellen incomplete dossiers-START	49,978.11	102,731.69	11,400

Table A.4 BPI 2012 Challenge: Descriptive statistics of case duration.

Stat	Value in seconds	Value in days
Mean	745,100.10	8.62
Standard deviation	1,047,978.00	12.13
Min	1.86	-
25th percentile	54.47	-
50th percentile	69,857.43	0.81
75th percentile	1,226,653.00	14.20
Max	11,855,940.00	137.22

Small log

Table A.5 Small log: Descriptive statistics of activity duration and frequency.

Activity	μ	σ	Frequency
Activity A	-	-	2,000
Activity B	3,600.00	-	2,000
Activity C	3,600.00	-	2,000
Activity D	3,600.00	-	2,000
Activity E	3,600.00	-	2,000
Activity F	3,600.00	-	2,000
Activity G	1,186.20	1,692.54	2,000
Activity H	3,600.00	-	2,000
Activity I	558.00	1,303.18	2,000
Activity J	628.20	1,366.68	2,000
Activity K	1,186.20	1,692.54	2,000
Activity L	1,186.20	1,692.54	2,000
Activity M	1,227.60	1,706.99	2,000
Activity N	1,227.60	1,706.99	2,000

Table A.6 Small log: Descriptive statistics of case duration.

Stat	Value in seconds	Value in days
Mean	28,800.00	0.33
Standard deviation	-	-
Min	28,800.00	0.33
25th percentile	28,800.00	0.33
50th percentile	28,800.00	0.33
75th percentile	28,800.00	0.33
Max	28,800.00	0.33

Large log

Table A.7 Large log: Descriptive statistics of activity duration and frequency.

Activity	μ	σ	Frequency
Activity A	-	-	15,000
Activity B	3600.00	-	15,000
Activity C	1797.36	1800.06	15,000
Activity D	1802.64	1800.06	15,000
Activity E	3600.00	-	15,000
Activity F	3600.00	-	15,000
Activity G	3600.00	-	5,073
Activity H	3600.00	-	4,994
Activity I	3600.00	-	4,933
Activity J	3600.00	-	15,000

Table A.8 Large log: Descriptive statistics of case duration.

Stat	Value in seconds	Value in days
Mean	21,600.00	0.25
Standard deviation	-	-
Min	21,600.00	0.25
25th percentile	21,600.00	0.25
50th percentile	21,600.00	0.25
75th percentile	21,600.00	0.25
Max	21,600.00	0.25

Appendix B

Scatter Plot of Anomalous Time

In this appendix, we will show the simulated anomalous time attribute graphically.

BPI 2013 Challenge

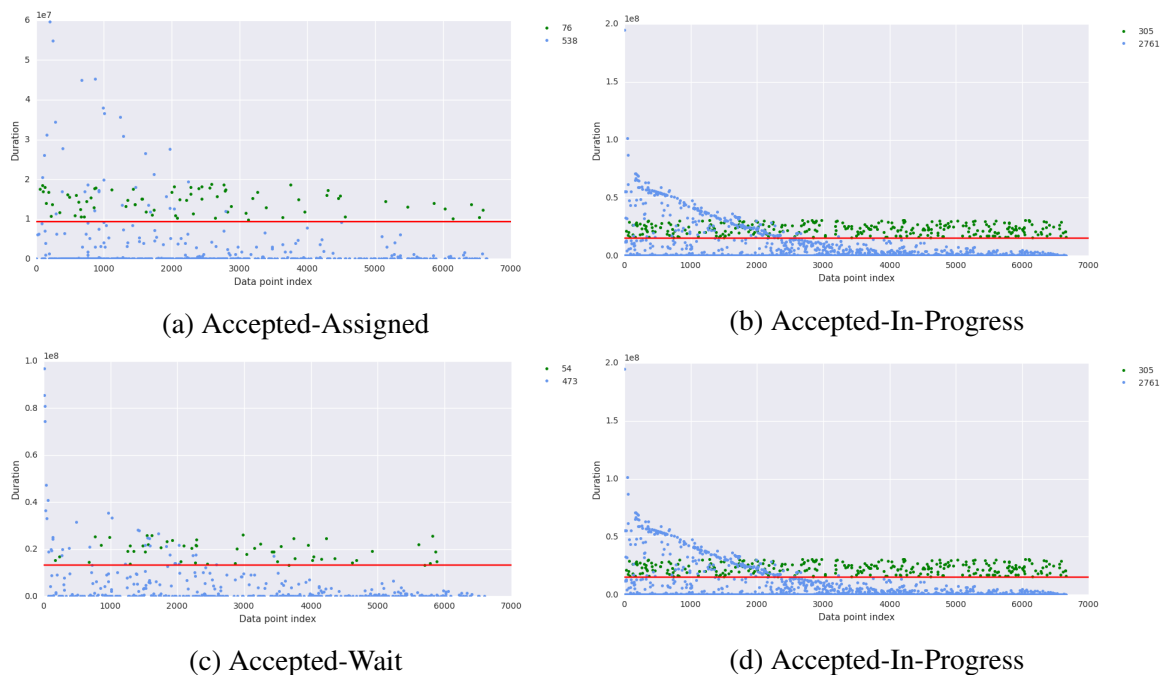


Fig. B.1 BPI 2013 log: Scatter plot of selected activity duration. The green dot denotes for anomalous data, blue dot denotes for the normal data and the red solid line is the borderline.

BPI 2012 Challenge

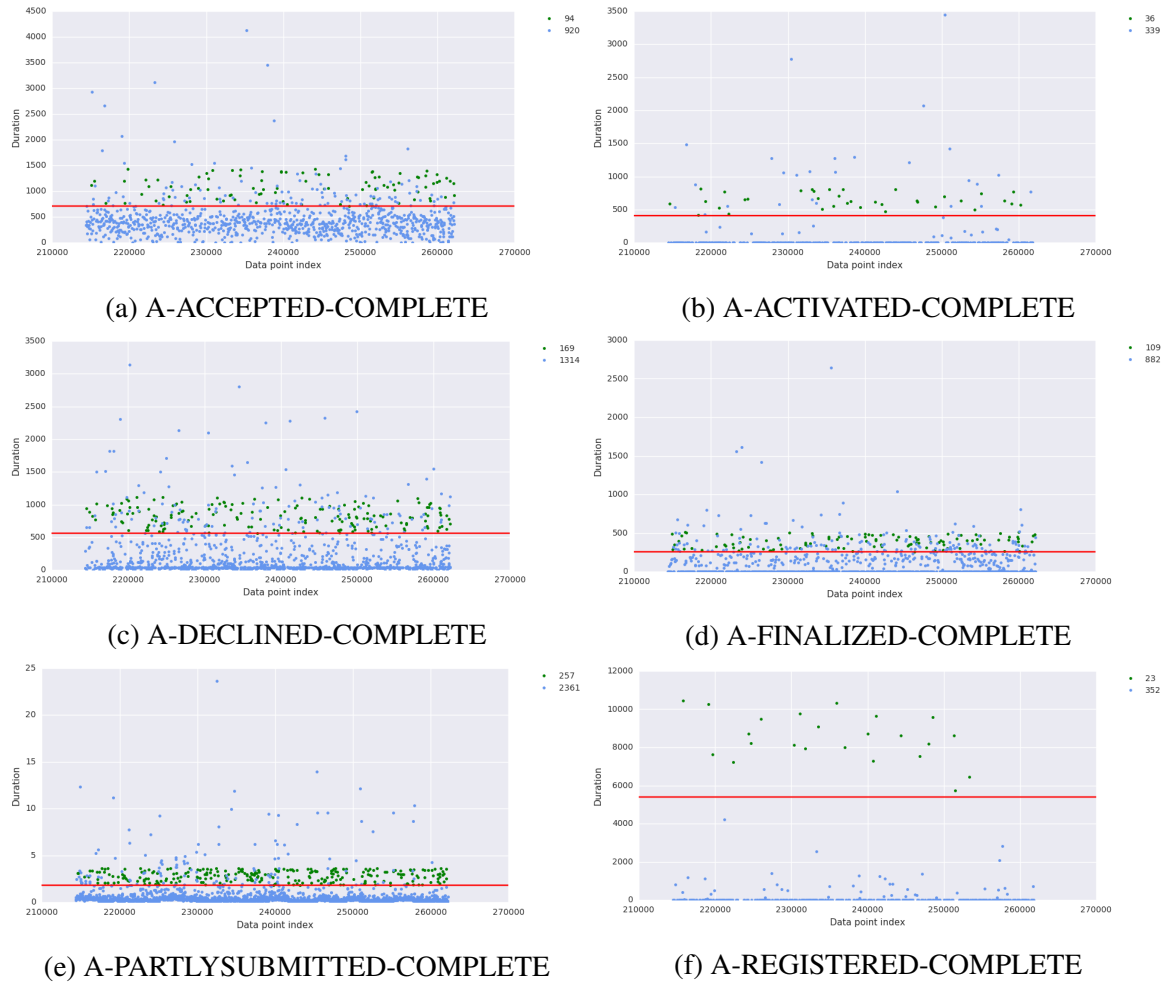


Fig. B.2 BPI 2012 log: Scatter plot of selected activity duration. The green dot denotes for anomalous data, blue dot denotes for the normal data and the red solid line is the borderline.

Small Log

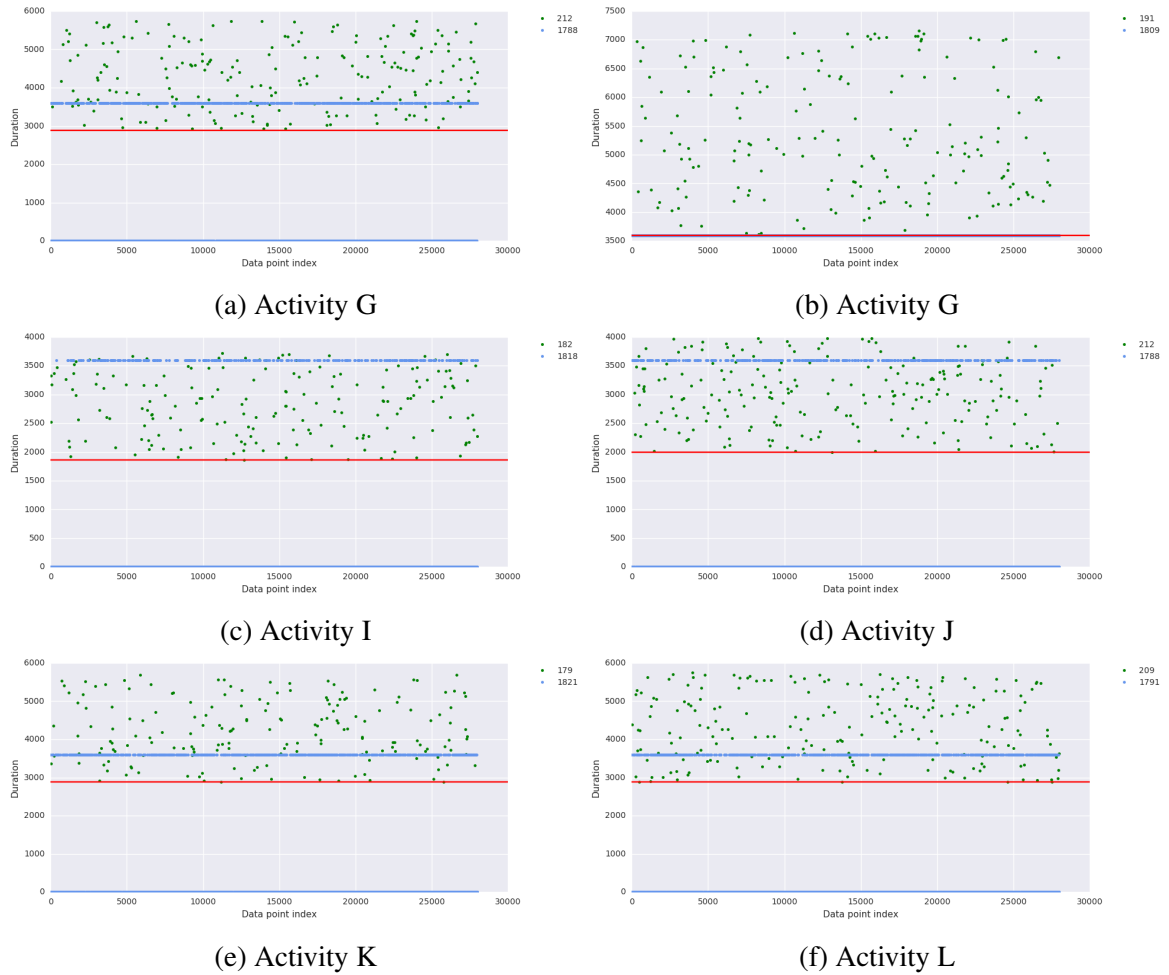


Fig. B.3 Small log: Scatter plot of selected activity duration. The green dot denotes for anomalous data, blue dot denotes for the normal data and the red solid line is the borderline.

Large Log

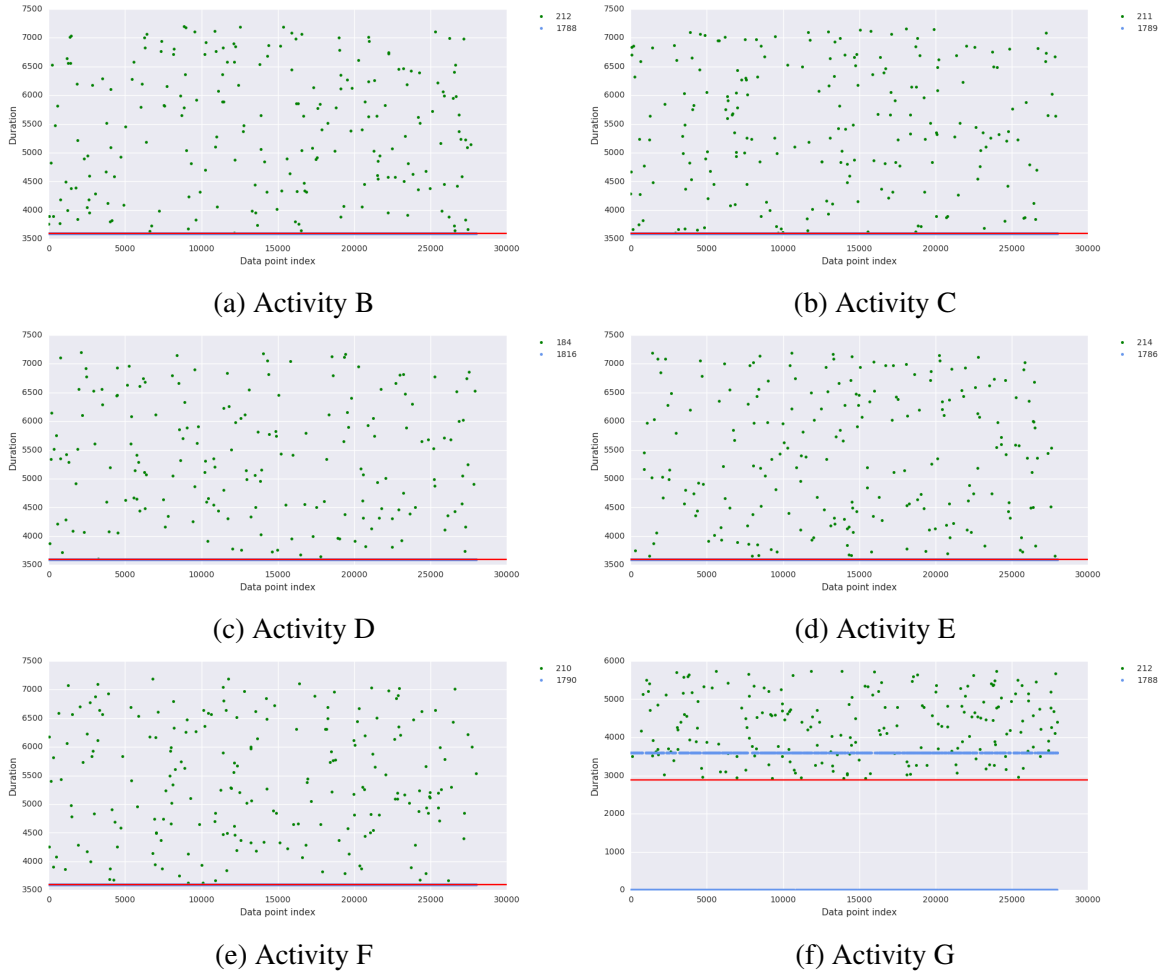


Fig. B.4 Small log: Scatter plot of selected activity duration. The green dot denotes for anomalous data, blue dot denotes for the normal data and the red solid line is the borderline.

Appendix C

Receiver Operating Characteristic Curve

In this appendix, we provide the visualisation of ROC curves obtained from the experiments in Chapter 4.

ROC Curve of Time attribute

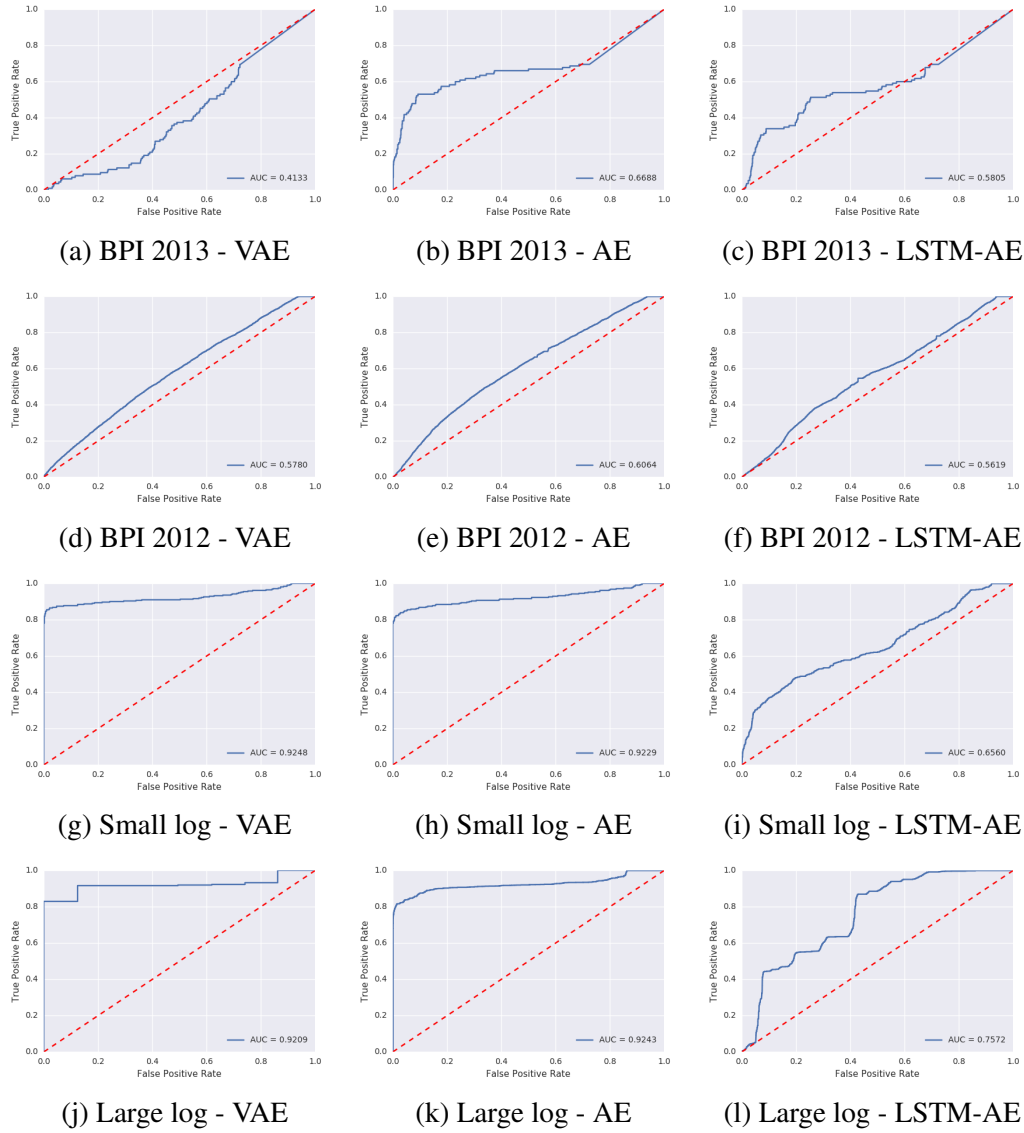


Fig. C.1 Receiver Operating Characteristic of Time attribute.

ROC Curve of Activity attribute

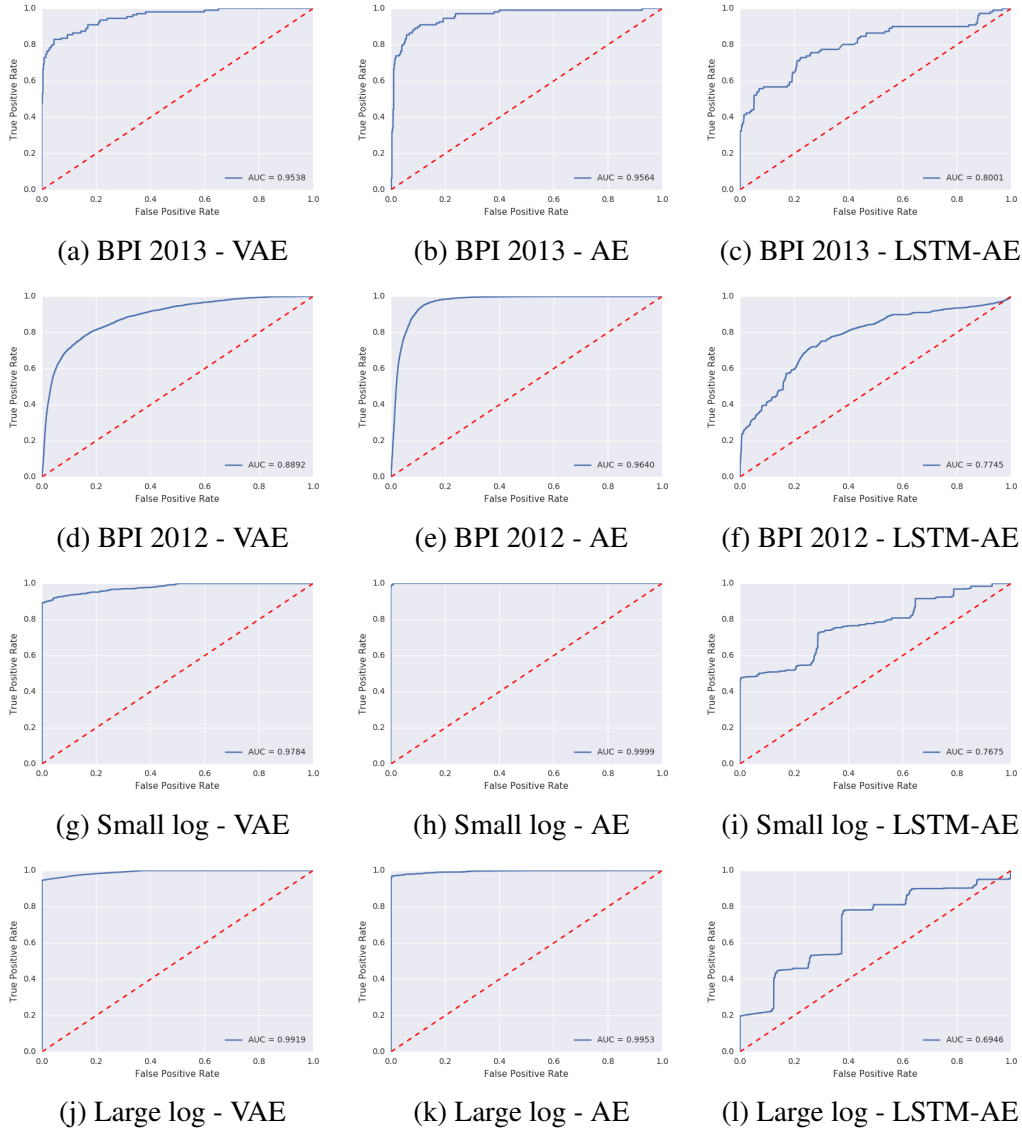


Fig. C.2 Receiver Operating Characteristic of Time attribute.

Appendix D

Implementation

In this appendix, we give a short instruction on how to reproduce the results shown in this thesis.

Event Log Reconstruction

1. Source code: <https://github.com/hoangnguyen3892/event-log-reconstruction>
2. Directory structure:

Event log Reconstruction

Directory structure:

```

event-log-reconstruction
|
|  README.md
|  requirement.txt
|
|--- data: original dataset
|   |
|   |  bpi_2013.csv
|   |  bpi_2012.csv
|   |  small_log.csv
|   |  large_log.csv
|
|--- data_preprocessing
|   |
|   |  induce_missing_data.py
|   |  preprocess_variables.py
|   |  real_log_preprocessing.sh
|
|--- utils
|   |
|   |  utils.py
|   |  models.py
|
|--- input: preprocessed data
|
|--- experiment
|   |
|   |  output
|   |  AE.ipynb
|   |  VAE.ipynb
|   |  LSTMVAE.ipynb
|
|--- base_model
|   |
|   |  dummy_imputation.ipynb
|   |  statistical_description.ipynb
|
|

```

Fig. D.2 Event Log Reconstruction: Directory structure.

3. Clone the repository into a new private directory:

```
$ git clone https://github.com/hoangnguyen3892/event-log-reconstruction
```

4. Follow README.md to install requirements and run the kernels.

