Lehrstuhl Angewandte Informatik IV
Datenbanken und Informationssysteme
Prof. Dr.-Ing. Stefan Jablonski

Institut für Angewandte Informatik
Fakultät für Mathematik, Physik und Informatik
Universität Bayreuth

UNIVERSITÄT
BAYREUTH

Exploring the Potential of Embeddings for Predicting the Next Activity of a Process

Christian Gebhardt

November 20, 2023
Version: Final

Universität Bayreuth

Fakultät Mathematik, Physik, Informatik
Institut für Informatik
Lehrstuhl für Angewandte Informatik IV

Masterarbeit

Exploring the Potential of Embeddings for Predicting the Next Activity of a Process

Christian Gebhardt

1. Reviewer	Prof. Dr.-Ing. Stefan Jablonski Fakultät Mathematik, Physik, Informatik Universität Bayreuth
2. Reviewer	Dr. Lars Ackermann Fakultät Mathematik, Physik, Informatik Universität Bayreuth
Supervisor	Msc. Martin Käppel

November 20, 2023

Christian Gebhardt
Exploring the Potential of Embeddings for Predicting the Next Activity of a Process
Masterarbeit, November 20, 2023
Reviewers: Prof. Dr.-Ing. Stefan Jablonski and Dr. Lars Ackermann
Supervisor: Msc. Martin Käppel
Universität Bayreuth
Lehrstuhl für Angewandte Informatik IV
Institut für Informatik
Fakultät Mathematik, Physik, Informatik
Universitätsstrasse 30
95447 Bayreuth
Germany

Kurzfassung

In den letzten Jahren haben Methoden zur Erlernung von Vektordarstellungen von Wörtern, welche semantische und syntaktische Eigenschaften erfassen, große Erfolge erzielt. Diese Vektordarstellungen, allgemein als Word Embeddings bekannt, können die Leistung für verschiedene Downstream Tasks im Natural Language Processing (NLP) Bereich verbessern, wenn diese als Eingabe verwendet werden (z. B. Word Similarity Tasks [1, 2], POS-Tagging [3] oder auch komplexere Aufgaben, wie Language Modeling [4]). Aufbauend auf einer früheren Arbeit von [5] haben wir die Anwendung dieser Techniken im Bereich des Process Mining untersucht, insbesondere im Kontext von Prozessaktivitäten.

Dafür haben wir zunächst einen Systematic Literature Review über bestehende Word und Sentence Embedding Techniken aus dem NLP Bereich durchgeführt, um einen Überblick zu erhalten und geeignete Techniken für unsere späteren Experimente zur Next Activtivity Prediction auszuwählen. Wir haben die Interpretierbarkeit der Embeddings für Aktivititäten untersucht, welche durch die Skip-Gram Architektur aus [1] auf synthetischen und realen Event Logs trainiert wurde, indem wir die Embeddings auf einen 2-dimensionalen Raum mit Principal Component Analysis (PCA) projizierten. Nach näheren Untersuchungen der Embeddings, haben wir unsere Erkenntnisse mit dem Learning Objective des Skip-Gram Modells in Bezug gesetzt und kamen zu dem Schluss, dass die resultierenden Aktivitätsvektoren tatsächlich Informationen zum Kontrollfluss der zugrunde liegenden Prozesse beinhalten.

Infolgedessen haben wir Experimente durchgeführt, um die Leistung von Embeddings und einfachen One-Hot-kodierten Aktivitäten als Eingabe für neuronale Netze bei der Next Activity Prediction gegenüberzustellen. Die Experimente zeigten jedoch bei keinem der drei verwendeten Event Logs signifikante Unterschiede in der F1-Score-Performance zwischen Modellen, die Embeddings bzw. One-Hot-kodierte Aktivitäten als Eingabe erhielten. Anschließend liefern wir potenzielle Gründe für diese Erkenntnisse und zeigen die Grenzen unserer Experimente auf.

Abstract

In recent years, methods for learning vector space representations of words, that capture word semantics and syntax, showed great success. These representations, commonly referred to as word embeddings, can improve performance on various NLP downstream tasks, when they are utilized as input (e.g. word imiliarity tasks [1, 2], POS-tagging [3] or even more complex tasks like language modeling [4]). Building upon previous research from [5], we investigated the application of these techniques in process mining, specifically focusing on process activities.

Therefore, we first provide a systematic literature review on existing word and sentence embedding techniques from Natrual Language Processing (NLP) to gain perspective and select appropriate techniques for our later experiments on next activity prediction. We explore the interpretability of activity embeddings learned by the CBOW and Skip-Gram models from [1] for synthetic and real world event logs, by reducing the representation to 2-dimensional space with Prinipical Component Anaysis (PCA). After closer inspection of the embeddings, we relate our findings to the Skip-Gram model objective and come to the conclusion that the activity vectors are indeed enriched with control flow information of the underlying processes.

Consequently, we conduct experiments to assess the performance of activity embeddings and simple one-hot encoded activities as input for neural network models on the next activity prediction task. However, the experiments revealed no significant difference in F1-Score performance between models utilizing embeddings and those employing one-hot encoded activities as input for any of the three event logs. Afterward, we present potential reasons to explain this finding and show the limitations of our experiments.

Contents

1 Introduction **1**

1.1 Business Process Management 1

1.1.1 Concepts and Terminology 1

1.1.2 Process Mining 5

1.1.3 Next Activity Prediction 9

1.2 Embedding Techniques 10

1.2.1 Word Embeddings: Capturing Word Meaning 11

1.2.2 Applying Embedding Techniques in Process Mining 14

1.3 Research Objectives and Thesis Structure 15

1.4 Related Work 16

2 Systematic Literature Review on Word Embedding Techniques **19**

2.1 Methodology 19

2.1.1 Research Questions 20

2.1.2 Search Queries 20

2.1.3 Study Selection Process 22

2.2 Classification of Identified Studies 23

2.2.1 Word Embedding Techniques 24

2.2.2 Sentence and Paragraph Embedding Techniques 34

2.3 Selection of Techniques 36

3 Interpretability of Activity Embeddings **39**

3.1 Synthetic Event Logs 40

3.2 Real World Event Log 44

4 Evaluation of Activity Embeddings in Next Activity Prediction **53**

4.1 Experimental Setup 53

4.1.1 Embedding and Encoding Techniques 53

4.1.2 Neural Network Architectures 54

4.1.3 Datasets 58

4.1.4 Training and Testing Details 60

4.2 Hypotheses 63

4.3 Results 64
4.4 Discussion 75

5 Conclusion 79

Bibliography 81

Introduction 1

In the following chapter, we give an introduction to Business Process Management (BPM), Process Mining and Word Embeddings as background for later chapters in section 1.1 and 1.2. We will outline the structure of this work and present our three main research objectives in section 1.3, before discussing related research in section 1.4.

1.1 Business Process Management

This section provides a brief overview of Business Process Management (BPM) and Process Mining, its main concepts and terminology. We explore the process mining task of next activity prediction in more depth, since it will be the focus of our experiments in chapter 4.

1.1.1 Concepts and Terminology

Business Process Management (BPM) plays an important role in modern organizations by offering a structured approach to enhancing operational efficiency and overall performance of the organization. It is crucial to represent processes digitally, to be able to integrate existing information systems in processes that are otherwise manually performed by employees, to keep track of the execution of these processes and to analyze them for improvement. Before defining the term Business Process Management, we should define the term Business Process first. We use the definitions from Mathias Weske in his book “Business Process Management Architectures” [6, p. 5]:

> “A *business process* consists of a set of activities that are performed in coordination in an organizational and technical environment. These activities jointly realize a business goal. Each business process is enacted by a single organization, but it may interact with business processes performed by other organizations.” [6, p. 5]

And the term *Business Process Management* is then defined as follows:

> "*Business process management* includes concepts, methods, and techniques to support the design, administration, configuration, enactment, and analysis of business processes." [6, p. 5]

So in general, Business Process Management helps organizations to formalize, execute and monitor the processes that are vital to the organization. To formalize business processes, we present them as business process models, which act as a kind of "execution plan" for the process. Here is a formal definition of a business process model together with a business process instance, which is another important term in this context:

> "A *business process model* consists of a set of activity models and execution constraints between them. A *business process instance* represents a concrete case in the operational business of a company, consisting of activity instances. Each business process model acts as a blueprint for a set of business process instances, and each activity model acts as a blueprint for a set of activity instances." [6, p. 7]

So far, the definitions might appear rather abstract to the reader. To give a better practical understanding, we may look at an example business process formalized in an according business process model in the business process modeling notation (BPMN), depicted in figure 1.1.

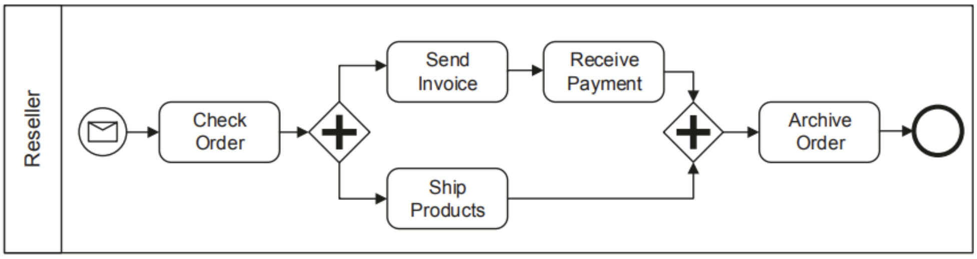


Figure 1.1: An example business process model from [6, p. 7] representing a simple ordering process of a reseller in BPMN.

The process model gives a formal representation of an ordering process of a reseller in BPMN. A message start event (mail symbol) triggers the start of the process, when there is an incoming order. Subsequently, the activity "Check Order" is performed, indicated as a rounded rectangle in BPMN. After the completion of the activity, the process flow encounters a parallel split, represented as a diamond with a plus sign. Here, the activities along the two branches, are executed in parallel. But note, that the activity "Send Invoice" has to be performed before the activity "Receive Payment". After the execution of these activities, the parallel split is joined again to one process branch and the process terminates after the activity “Archive Order” is executed, indicated by the circle with a thick black border.

The control flow determines, which sequences of process activities (trace variants) conform to the process model. Although there might be different variants of the order, in which the activities are performed, due to the parallel split. The activity “Ship Products” on the one parallel branch might be executed before, between, or after the two activities “Send Invoice” and “Receive Payment” on the other parallel branch. In general, the activities may be executed by resources of the reseller company, either employees or automated by a system. Since multiple business process instances of the same business process model can exist, there is a one-to-many relationship between the process model and the process instances belonging to the process model.

For more information, a business process model, representing the activities performed from the perspective of the buyer and a complete model representing the interaction of the models from the reseller and buyer perspective, can be found in [6, pp. 8-9]. Also note, that the Business Process Modeling Notation offers many more elements to model much more complex business processes than the one presented here. But the simple model from the reseller perspective should be sufficient as background for this work. Again, for more detail, the official documentation of BPMN can be found on the BPMN website[1].

Today, there are software systems, called Business Process Management Systems (BPMS) to perform the core activities needed for successful Business Process Management. BPMS are defined as follows in [7]:

> “A *Business Process Management System* (BPMS) is a system that supports the design, analysis, execution, and monitoring of business processes on the basis of explicit process models.” [7, p. 344]

The core BPM tasks (design, analysis, execution, and monitoring) might be thought of as an iterative lifecycle model with different phases, rather than separate tasks. One variant of the BPM lifecycle was proposed in the book [8, p. 31] and is depicted in figure 1.2 on the next page. It should be recognized, that the process lifecycle may not be run only once, but rather multiple times, where changes in each phase are made according to an outcome from the preceding phases.

Moving forward, we take a closer look at the various phases comprising the BPM lifecycle, starting with the diagnosis/requirements phase. In this phase, relevant business processes are identified and the requirements for the process are defined (or changed in later cycles, when necessary), e.g., by interviewing domain experts. Typically, the requirements outline, which information needs to be incorporated

[1] `https://www.bpmn.org/`

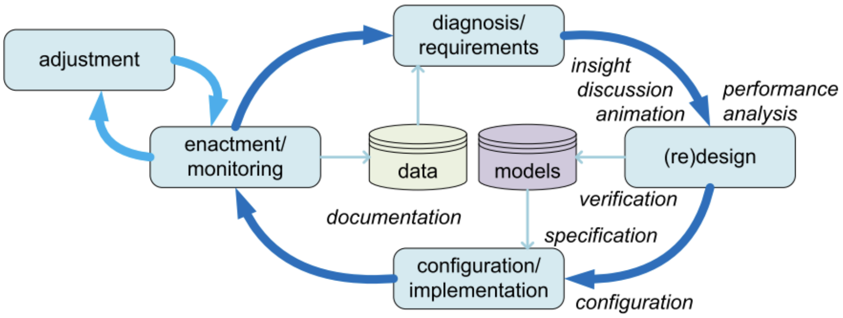


Figure 1.2: The BPM lifecycle from [8, p. 31] showing the different phases of Business Process Management.

in the process in form of activities, data, users, technology/software and further syntactical information, such as control flow, constraints, and rules. In the next phase, the (re-)design phase, the process model is first designed (or modified) according to the defined requirements in an appropriate modeling language, for example BPMN.

In the configuration/implementation phase, the process model needs to be transformed, such that it is executable by a business process management system. This includes the configuration of the interaction of employees/users with the BPMS, as well as integration of the necessary software systems for the process, which might need further implementation work [6, p. 13]. After the processes are completely executable by a BPMS, the enactment/monitoring phase begins. In this phase the processes are monitored, and the execution logs are evaluated in terms of some key performance indicators (KPIs), e.g., execution time or execution cost. In the adjustment phase, some of these changes are handled by reconfiguration, but note, that no new software is implemented, and the process is not redesigned yet [8, p. 31].

In a new iteration of the process life cycle, requirements are redefined as needed, if there is a change in the environment of the business process. Finally, if there is need for changes due to poor performance (e.g., bottlenecks during the execution), recurring errors or deviation from the intended process behavior [7, p. 24], then these changes should be reflected when redesigning the process. This cycle should be reiterated as often as required to continuously improve the process and its performance in execution.

1.1.2 Process Mining

This section gives a high-level overview of the research field of *Process Mining*. Process Mining, is described as the missing link between the two areas of data science and process science in [8, pp. 17-18], because it can utilize insights from observed behavior to discover new processes, perform conformance checking of process models or enhance existing processes. Thus, process mining can help to improve processes with data-driven techniques, in the relevant phases of the BPM lifecycle. The three activities listed above are the main subfields of process mining, namely *Process Discovery*, *Conformance Checking* and *Process Enhancement* will be explained briefly in this section. We will also look at yet another subfield called *Process Prediction* or *Predictive Business Process Monitoring*, since it will be of interest in later experiments.

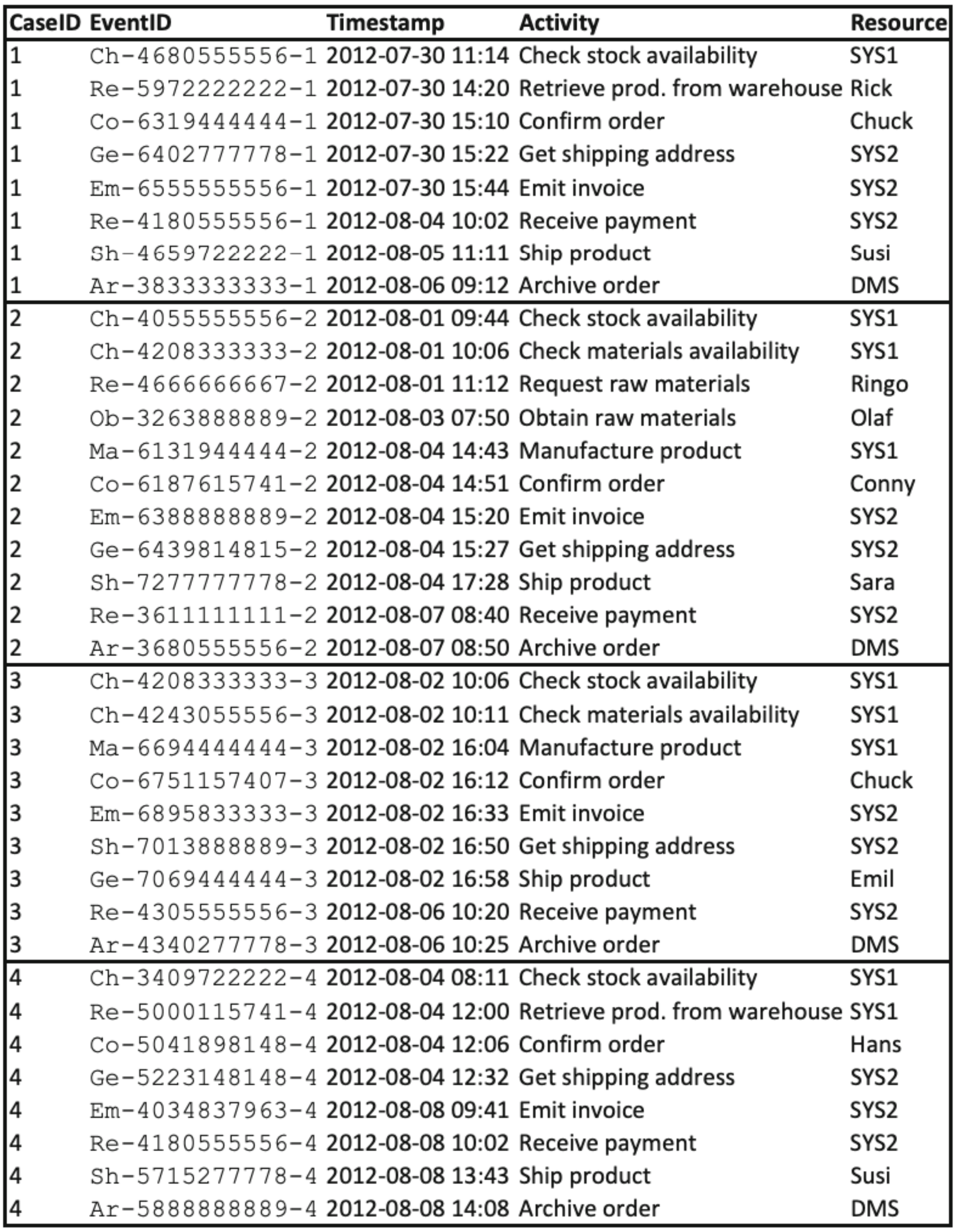
Event Logs

But first, to gain insights from observed process behavior, the process execution data has to be gathered and stored in the form of *event logs*. This data may already be available as execution logs from the business process management system or from other information systems used in the organization. Event logs provide information about *one process* and can contain *multiple cases* of this process. These cases consist of multiple events that were performed in one particular order, where one event can only belong to one process case (i.e., business process instance). Each event refers to one activity (one well-defined step of the process) [8, p. 32]. Together with the activity name, additional information about the event can be stored, such as the resource, that executed the event (person, software, etc.), the timestamp or other data elements that are linked to the event (e.g., items of an order) [8, p. 33]. An example event log of an order-to-cash process is depicted in figure 1.3.

This simple tabular event log could be interpreted as a Comma Separated Value (CSV) file, with events as rows. Meanwhile, the five columns represent the attributes of the event, whereas the “CaseID” is an identifier for the specific process case the event belongs to, the “EventID” is a unique identifier for the specific event, the other three attributes describe additional information of the event, such as the timestamp, the name of the corresponding activity and the resource that executed the event. For event logs, where each event can have a variable number of attributes (e.g., additional data elements), the commonly used XES[2] format might be more suited.

There also exists one sequence of events ordered by their timestamp for each case, called *trace*. For example, in the event log from figure 1.3 the trace for the case

[2] https://xes-standard.org/



CaseID	EventID	Timestamp	Activity	Resource
1	Ch-4680555556-1	2012-07-30 11:14	Check stock availability	SYS1
1	Re-5972222222-1	2012-07-30 14:20	Retrieve prod. from warehouse	Rick
1	Co-6319444444-1	2012-07-30 15:10	Confirm order	Chuck
1	Ge-6402777778-1	2012-07-30 15:22	Get shipping address	SYS2
1	Em-6555555556-1	2012-07-30 15:44	Emit invoice	SYS2
1	Re-4180555556-1	2012-08-04 10:02	Receive payment	SYS2
1	Sh-4659722222-1	2012-08-05 11:11	Ship product	Susi
1	Ar-3833333333-1	2012-08-06 09:12	Archive order	DMS
2	Ch-4055555556-2	2012-08-01 09:44	Check stock availability	SYS1
2	Ch-4208333333-2	2012-08-01 10:06	Check materials availability	SYS1
2	Re-4666666667-2	2012-08-01 11:12	Request raw materials	Ringo
2	Ob-3263888889-2	2012-08-03 07:50	Obtain raw materials	Olaf
2	Ma-6131944444-2	2012-08-04 14:43	Manufacture product	SYS1
2	Co-6187615741-2	2012-08-04 14:51	Confirm order	Conny
2	Em-6388888889-2	2012-08-04 15:20	Emit invoice	SYS2
2	Ge-6439814815-2	2012-08-04 15:27	Get shipping address	SYS2
2	Sh-7277777778-2	2012-08-04 17:28	Ship product	Sara
2	Re-3611111111-2	2012-08-07 08:40	Receive payment	SYS2
2	Ar-3680555556-2	2012-08-07 08:50	Archive order	DMS
3	Ch-4208333333-3	2012-08-02 10:06	Check stock availability	SYS1
3	Ch-4243055556-3	2012-08-02 10:11	Check materials availability	SYS1
3	Ma-6694444444-3	2012-08-02 16:04	Manufacture product	SYS1
3	Co-6751157407-3	2012-08-02 16:12	Confirm order	Chuck
3	Em-6895833333-3	2012-08-02 16:33	Emit invoice	SYS2
3	Sh-7013888889-3	2012-08-02 16:50	Get shipping address	SYS2
3	Ge-7069444444-3	2012-08-02 16:58	Ship product	Emil
3	Re-4305555556-3	2012-08-06 10:20	Receive payment	SYS2
3	Ar-4340277778-3	2012-08-06 10:25	Archive order	DMS
4	Ch-3409722222-4	2012-08-04 08:11	Check stock availability	SYS1
4	Re-5000115741-4	2012-08-04 12:00	Retrieve prod. from warehouse	SYS1
4	Co-5041898148-4	2012-08-04 12:06	Confirm order	Hans
4	Ge-5223148148-4	2012-08-04 12:32	Get shipping address	SYS2
4	Em-4034837963-4	2012-08-08 09:41	Emit invoice	SYS2
4	Re-4180555556-4	2012-08-08 10:02	Receive payment	SYS2
4	Sh-5715277778-4	2012-08-08 13:43	Ship product	Susi
4	Ar-5888888889-4	2012-08-08 14:08	Archive order	DMS

Figure 1.3: An example event log of an order-to-cash process from [7, p. 423]

with “CaseID” 1 would be described as follows (since this is the chronological order of events for the case): <*Check stock availability, Retrieve prod. from warehouse, Confirm order, Get shipping address, Emit invoice, Receive Payment, Ship product, Archive order*>. One process can have multiple *trace variants*, if the set of executed activities or the execution order of the traces differ. But two traces are of the same trace variant, if their set of executed activities match, and if their execution order is identical.

Process Mining Techniques

Now that we have explored the fundamentals of event data, we can see how process mining techniques use the information about observed process behavior, that this data contains. The discussed techniques are generally adopting other techniques and algorithms from the data mining and machine learning domain to solve problems in the business process management domain. Figure 4 illustrates the position of the three main process mining techniques in business process management and the relationship of process model and event log in the respective process mining technique by the red arrows. These relationships will also become clearer, when explaining the three main types of process mining in more detail, in the following paragraphs. Also, a fourth technique, *Predictive Business Process Monitoring*, will be introduced.

The first process mining technique is *Process Discovery*, which takes an event log as input and automatically generates a business process model as output. The produced business process model should match the as-is behavior seen in the event log as closely as possible [7, p. 420]. The automatically discovered model can then be used as a new model of the process or can be compared with an existing model for improvement, speeding up the otherwise time-consuming modeling process.

The second technique is *Conformance Checking*, which compares an existing process model with the event log of the corresponding process, giving a list of differences as output. Now, this list of differences can be used to detect, locate and explain deviations and also to measure how closely the process behavior observed in the event log matches the process model [8, p. 33]. The detected deviations from the process model can be sometimes attributed to an error, or more commonly, to an exception, an alternative trace variant, that is not accounted for in the process model [7, pp. 420-421].

The third main process mining technique is *Process Enhancement*. Here, we also take an existing process model and an event log as input, and now try to improve or extend the existing model by the process information of the event log. Two common tasks of process enhancement are repair, i.e., changing the process model to better

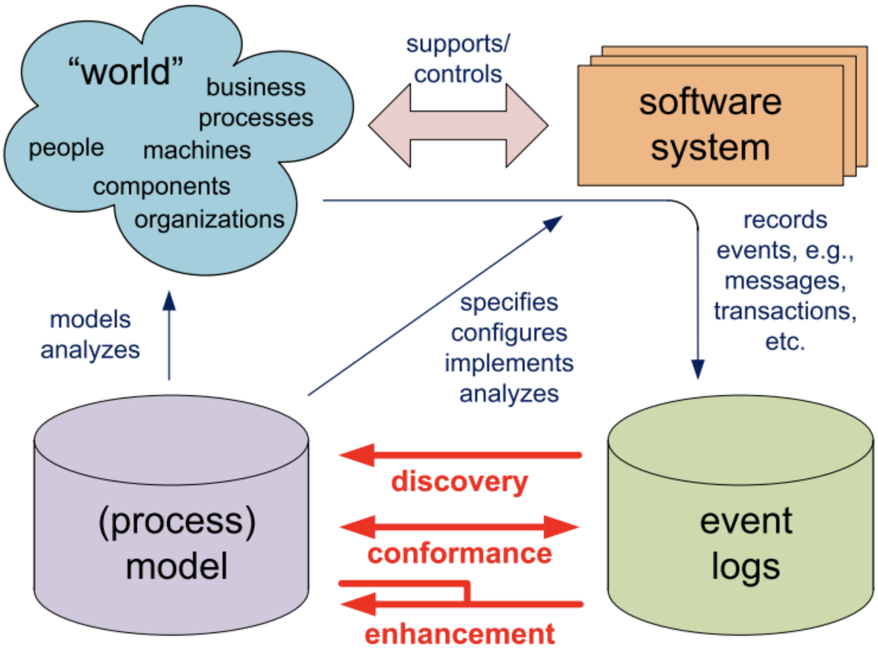


Figure 1.4: The three main types of Process Mining and their role in Business Process Management from [8, p. 32]. The relationship between the process model and event log in the respective Process Mining technique is indicated by the red arrows.

conform to observed behavior, and extension of the process model, i.e., to display performance data, resources, or other useful information along with the process model [8, p. 33].

Another process mining technique is *Predictive Business Process Monitoring* (also called *Process Prediction*). While the other techniques presented so far, are used in an offline setting and help to improve the process over a longer period of time, Predictive Business Process Monitoring is a form of online learning and can give real-time predictions for a running process instance. As a result, it can provide operational support to the workers and decision-makers involved in the process. For example, given an incomplete trace of an ongoing process instance, we can make different predictions e.g., the outcome of the process as the cost of the process, the final event or the remaining time for the process to finish, derived from past event log data of the process [8, p. 311]. We can also predict the probabilities of the next activity occurring, giving valuable information to workers managing the process ahead of time. This allows them to anticipate future process behavior and proactively prepare solutions to potential problems.

1.1.3 Next Activity Prediction

This section investigates *Predictive Business Process Monitoring*, in particular *Next Activity Prediction*, more thoroughly. At the end of the last section, some common problems of predictive business process monitoring were explained already, but in general every task that aims to make a prediction ahead of time for a running, incomplete business process instance, also called a *partial trace*, can be thought of as predictive business process monitoring [8, pp. 305, 311].

This can include for example predicting the outcome of a partial trace, the remaining time, or the probability that a certain activity will be next in the trace. Before we start with the definition of the task, we want to point out, that we separate between the two terms event and activity. An event is part of an event log and has several attributes, e.g., case, activity, timestamp, resource, and other data elements about the event, while an activity simply represents the operation, the task action [9]. Other literature may describe the task of Next Activity Prediction as Next Event Prediction, since they are either using the terms event and activity interchangeably, or they in fact predict more information than just the next activity (e.g., timestamp). However, we only focus on predicting the next activity and no further information about the event.

In the context of *Next Activity Prediction*, the task is to predict the next activity, denoted as a_{n+1}, in a partial trace T based on a given prefix of activities $a_{n-k}, a_{(n-k)+1}, ..., a_n$, where k is a positive integer satisfying $0 < k \leq n$, where n represents the size of the partial trace T.

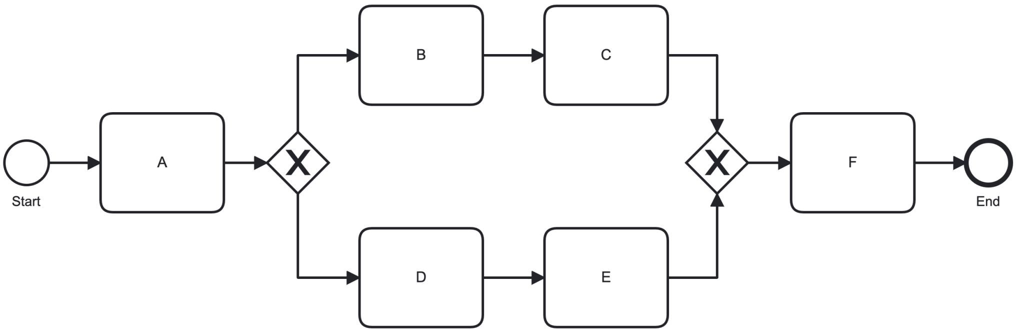


Figure 1.5: An example process model for next event prediction. The process is split after activity A by an exclusive gateway, which requires the execution of exactly one of the two branches, either B, C or D, E. The process is then finished after the completion of activity F.

To learn which activity will be next in the trace, we can analyze historical data from event logs of this specific business process. This assumes, that the process will have a similar future control flow for similar partial traces. If we look at the example process model from figure 1.5, a partial trace <*A*, *B*> might be given, and our task is to predict the following activity. If, for simplification, all process instances from the event log conform completely to the process model, meaning that there is no violation of the control flow, then we can be sure that for <*A, B*> activity C has to follow. However, if only the partial trace <*A*> is given, the next activity can be both B or D after the exclusive gateway. At this point our only option is to check the historical data, which of the two activities occurred more frequently after A, and take this activity as our prediction. Note, that in real-world applications, we often do not have access to a process model, that was created manually beforehand. We might discover a model from the event log with process discovery techniques (see [10]), but real-world process behavior can deviate from the model, which makes the task of next activity prediction even more difficult.

We can refine our predictions by including more information from the event log, such as the resource, the timestamp or additional data elements of the events. If someone applies for a credit in a loan process, not only the control flow (i.e., the sequence of events in partial traces) can have an impact on the next events, but also for example the initially requested amount of the credit. Also, the timestamp of events could potentially influence future events, i.e., if for one trace, the relative time between events is higher. This could imply, that the financial institution assigns lower priority to this process instance and the request may be handled differently.

At first, Hidden Markov Models [11, 12] and Probabilistic Finite Automatons [13] were used as modeling techniques for Next Event Prediction, but more recently, deep learning approaches gained more attention (e.g., [14, 15, 16]). The deep learning techniques take the sequence of preceding events as input in a neural network architecture and output a probability distribution, representing the probability for each possible activity. For our task of predicting the next activity, we might only pick the one activity with the highest probability as our final prediction. In chapter 4, we see how neural networks can be used for next activity prediction, and we explore two neural network architectures in more detail, the *Feed-Forward Neural Network* (FNN) and the *Long-Short-Term-Memory Neural Network* (LSTM).

1.2 Embedding Techniques

In the following section, we will present the concept of word embeddings from Natural Language Processing (NLP). In this context, we will see how the utilized models exploit one important linguistic observation, the *Distributional Hypothesis*, to capture word meaning in the learned embeddings and show the advantages of word embeddings compared to traditional encoding techniques (section 1.2.1).

Furthermore, we will show, how the advantages of these modern methods, might be also applicable in the process mining domain (section 1.2.2).

1.2.1 Word Embeddings: Capturing Word Meaning

Natural Language Processing (NLP) is located in the fields of Computer Science and Linguistics, and it examines how machines can process, understand and generate language. Some common applications of NLP include part-of-speech tagging (POS tagging) of words, machine translation from one language to another, or even question answering, which forms the basis for modern chatbots. All these tasks require an appropriate encoding method of text input (words in particular), such that a machine, usually some form of machine learning algorithm (e.g., a neural network) can process and interpret it.

To find a representation of words, we need to convert the word strings to numbers, so that a neural network can perform the necessary numeric operations for their learning procedure. In one simple approach, we may represent the words as index numbers, where each index is mapped to exactly one word from the vocabulary, which includes all words occurring in the processed text or document. The disadvantage of this method is that it imposes an ordering on the words, which is in reality non-existent. This might cause problems in the learning of the neural net, since it might infer that a word is more similar to one word than another word, since their distance of indices is smaller.

With this in mind, we could use another method, called *One-Hot Encoding*, to represent our words. In one-hot encoding, each word with index i is represented by a binary vector of length n, with n being the size of the vocabulary, in which all elements are zero, except for one element at the position i. This technique does not impose an ordering on the words, because all word vectors are equidistant to each other. But this comes with the cost of high dimensionality, since documents or texts may have big vocabularies (high number of unique words). This leads to inefficient space usage, as the one-hot vectors are sparse, and is also computationally more expensive, as the required input parameters of the neural network grow [17]. However, One-Hot Encoding is commonly used to represent categorical data and therefore is also widely used as standard technique to represent activities for deep learning in process mining.

Another important aspect, which both of the two previous methods lack, is encoding semantics in the word representations. In fact, words have semantic relationships to each other, as for example synonyms like “warm” and “hot” have very similar meaning. These relationships should also be somehow reflected in their encoding. *Word Embeddings* solve this problem by representing words as vectors in multidimensional vector space, where word vectors with similar semantics are close to each other in distance or point in similar directions.

We can learn these vectors by applying the *Distributional Hypothesis* formulated in [18], stemming from the following idea: *"you shall know a word by the company it keeps"* [19]. This important concept of linguistics, asserts that semantically similar words tend to occur in similar contexts. We can use this underlying idea of similarity by word-word co-occurrence to construct word vectors, by either applying deep learning or matrix factorization techniques on large text corpora, that can contain millions of words. As an example, consider the two words "warm" and "hot" in the following two sentences:

1. "It was very *warm* during the summer months this year, but the temperature declined in September."
2. "We have a *hot* summer this year, with some weeks experiencing extremely high temperatures in July."

The words "warm" and "hot" are synonyms to each other and semantically very similar. If we look at their respective contexts (words that appear along with them in the same sentence), the important terms "summer" and "temperature" are also in both of the sentences. But also some less relevant words, commonly referred to as stop words, appear in the sentences, which we might give less attention. This includes for example determiners ("the", "a/an"), conjunctions ("and", "or", "but", etc.), prepositions ("at", "for", "during", etc.) and other word types with little semantic information.[3] Imagine a situation where we have text corpora with thousands of sentences, including the words "warm" and "hot". It is very likely that in a modest fraction of the sentences "temperature" or "summer" is also present. In another fraction of the sentences might contain some other important context terms like "climate", "weather" or "sun".

Word Embedding techniques exploit this linguistic property on large training datasets, and ultimately are able to learn semantically-rich word vectors. We will take a closer look at how these techniques learn these word vectors in the systematic literature review on word embeddings in chapter 3.

After leveraging huge text corpora as training data for the learning techniques, the resulting word vectors allow simple computation of semantic similarity of two words. Furthermore, they show interesting semantic relationships to each other, which can be expressed by algebraic vector operations. For example, by taking the vector representation of the word "king", subtracting "men" and adding "woman", a word

[3] see `https://nlp.stanford.edu/IR-book/html/htmledition/dropping-common-terms-stop-words-1.html` for more information.

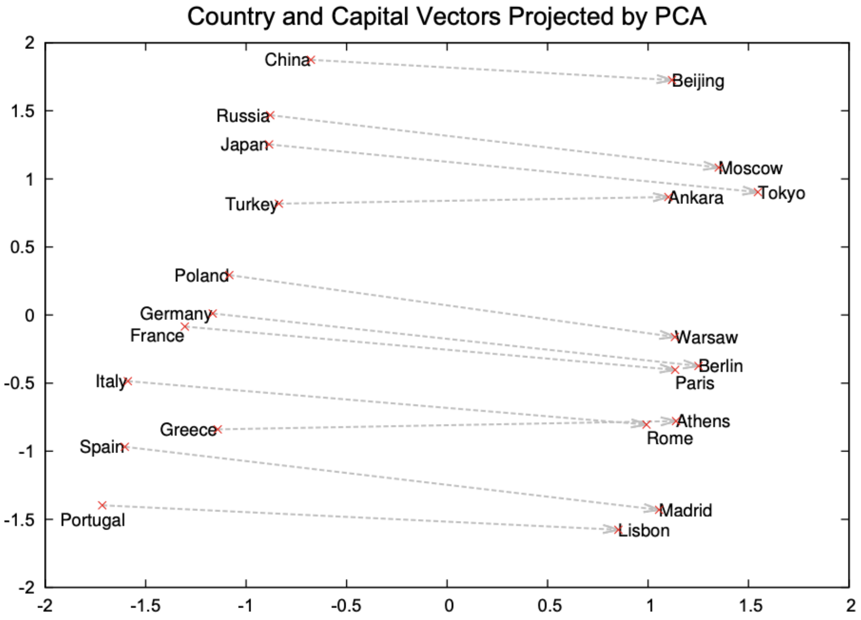


Figure 1.6: Word Embeddings with dimension 1000 of countries and their corresponding capitals reduced to two dimensions with the dimensionality reduction technique PCA from [2]. This illustrates that word embeddings internally capture the concepts of countries and capitals and learns their semantic relationships.

vector is computed that is close in distance to the vector for the word “queen” [1]. Another example, which showcases how word embeddings are able to distinguish between the two concepts of countries and capitals and represent their semantic relationship, is illustrated in figure 1.6.

All together, word embeddings have the advantage, that they capture semantics and meanwhile they do not have the memory and computational cost of higher dimensionality for large vocabularies V with size $|V|$, since they are vectors of fixed length d. The vector dimension d can be selected in the training process of the embeddings, but typically $d << |V|$. After pre-training the word embeddings, the resulting word vectors can be used as representations for words that occur in text corpora of NLP downstream tasks, such as POS-tagging, Chunking, etc. This is one of the main reasons why word embeddings are so powerful. They can be pre-trained once on large text corpora as training sets to learn a semantically-rich representation of the words, and then be reused on a wide-range of NLP tasks [3]. This approach can lead to significant performance improvements, surpassing earlier state-of-the-art methods, e.g., in semantic and syntactic language modeling tasks [1].

1.2.2 Applying Embedding Techniques in Process Mining

In the previous section, we introduced the concept of word embeddings and already touched on some benefits, that word embedding techniques offer in the NLP domain. These advantages include the reusability of pretrained embeddings and, most importantly, the potential to capture semantics in fixed length word vectors, while not suffering from the cost of high dimensionality in contrast to one-hot encoding. As a result, performance gains in various NLP tasks were achieved, by utilizing the pre-trained word embeddings later in their own training [1].

This naturally raises the question, if these embedding techniques are limited to the NLP domain or if we could adapt them to learn representations enriched with semantics in other domains as well. The work from [5] explores this interesting idea to learn embeddings for activities, traces, logs, and process models in the field of process mining. In the case of activities, the authors argue, that we can view activities in a trace similarly to words in a sentence (or paragraph), where the corpus equals the unique activities in the event log. The Distributional Hypothesis might also hold true for activities, as the context of activities could determine their semantic relationship, for example their relative position in the underlying process model. Consequently, one could leverage the already existing techniques from the NLP domain to learn embeddings for activities in the same way as for words.

The authors from [5] adopt techniques from the word2vec[4] package to learn such embeddings for activities. The package includes the Skip-Gram and CBOW architectures from [1, 2], which are also applied in later experiments of this work and are covered in more detail in the systematic literature review in chapter 3.

In their discussion, the researchers point out several future areas of research, including the interpretability of the learned activity embeddings. Like in the aforementioned “King and Queen” example in section 1.2.1, which shows the existence of semantic relationships between learned word vectors, it is yet to determine if we can observe any semantic relationships in activity embeddings as well. The authors also propose Predictive Business Process Monitoring as a potential application area of activity embeddings. This entails investigating how using embeddings as input for neural networks affects task performance, in comparison to conventional encoding techniques. We will primarily focus on the research of these two aspects of activity embeddings, the interpretability, and the influence on task performance in predictive business process monitoring.

[4] https://code.google.com/archive/p/word2vec

1.3 Research Objectives and Thesis Structure

This section will define the research objectives and will outline a structure of the thesis. The main three research objectives and scientific contributions of this master thesis are:

Objective 1: Systematic Literature Review on Word and Sentence Embedding Techniques

Objective 2: Interpretability of Activity Embeddings

Objective 3: Impact of Activity Embeddings on Next-Event Prediction Performance

The first objective is to give a comprehensive summary and categorization of existing word, sentence, and paragraph embedding techniques in the NLP domain by performing a systematic literature review (chapter 2). The literature review should also give the reader a high-level understanding of the techniques, how they are applied in the NLP domain, and finally support the selection process of embedding techniques for later research objectives.

The second objective is to explore the interpretability of the learned activity embeddings (chapter 3). The interpretability of embeddings was also highlighted as potential future research in the paper [5], but this area has not been widely investigated yet. This study will focus on inspecting potential semantic relations between the learned activity embedding vectors in hyper-dimensional space. For example, the learned vectors may contain control flow information of the process model, e.g., the position of an activity relative to other activities in the process model. For our analysis on interpretability, we compare the vectors by one similarity metric and visualize the embeddings with principal component analysis (PCA) in 2-dimensional space. Eventually we will determine, if semantic relationships exist between activities, and what kind of process information they represent.

The third objective aims to assess the effects on performance of activity embeddings in a process mining task (chapter 4). The work [5] brings up predictive process monitoring as a potential application area of activity embeddings, among others. Therefore, we picked the next activity prediction task from this field to examine the effects of activity embeddings on relevant performance metrics, compared to one-hot-encoding as baseline. The source code of our experiments can be found on GitHub[5] to reproduce our results, and it can also be reused for experiments with other neural network models, embedding methods or datasets. Some studies already used event embeddings as input for neural networks in predictive business process

[5] https://github.com/Christian-Gebhardt/act2vec_evaluation

monitoring and next activity prediction (e.g., [16, 14]). But no study extensively compared pretrained embeddings learned by modern NLP techniques with traditional encoding techniques of activities in a deep learning setup and how this change of input representation might impact performance on next activity prediction.

1.4 Related Work

In this section, we want to discuss research, that is related to our three research objectives, as defined in the previous section. The studies [20] and [21] aim to provide overviews on word embedding techniques in NLP, similar to the systematic literature review conducted within this work (objective 1). The survey on word embeddings from [20] selected techniques based on citation count and impact on newer techniques. The researchers categorize the techniques similarly in prediction and count-based techniques, but do not include contextual embedding techniques or sentence embedding techniques in their survey. Moreover, the work does not follow a systematic approach like our review. The study [21] follows a systematic approach, but does not have the same scope as our review, as the researcher only searched three scientific databases with 150 results in total (we complemented our search with Google Scholar, yielding around 3900 results). Again, the study did not include contextual and sentence embedding techniques. Consequently, we have a wider scope in our review and additionally include studies that are not present in their review. Thus, our systematic literature review adds a new and broad perspective on existing word and sentence embedding techniques.

One very important study, related to our research, is [5]. Here, the researcher introduced the idea of learning representations for process activities, traces, logs, and models with embedding techniques based on neural networks from NLP and evaluated the representations on trace clustering and process model comparison tasks, showing promising results. Additionally, they highlighted potential areas of future research, which inspired our research on the interpretability of activity embeddings and the application on next activity prediction. To learn representations of activities, they propose architectures based on the two predictive CBOW and Skip-Gram models from [1, 2], but without evaluating the embeddings on downstream tasks. Other studies that used embedding techniques from NLP to learn activity or event representations are [22] and [23]. In [22] word2vec models are applied to an "event-connection graph" to learn event representations in the health care domain. The quality of the learned representations is then evaluated on how well the representations of events from different, predefined categories are distinguishable. They also explore the interpretability of the event embeddings, by asking domain experts to rate the similarity of events (related to our research objective 2). The study [23] employs activity embeddings learned by an approach similar to the Skip-Gram model on the task of remaining process time prediction.

In a recent systematic literature review on state-of-the-art deep learning methods for process prediction [24], the researchers identified 15 studies that employ neural network models for next-event-prediction. Among these, six studies [25, 16, 26, 14, 27, 28] use embeddings to encode the events instead of one-hot encoding (some also use additional event information other than the activity name). Notably, only one study [16], pretrained the embeddings for both activity and resource information of events in a separate neural network, by providing the network with positive and negative examples of attribute associations as input. In contrast, the remaining studies learned embeddings jointly during the task, without pretraining. This leaves the evaluation of pretrained activity embeddings by NLP techniques on the task of next activity prediction (objective 3) as an open area of research.

2 Systematic Literature Review on Word Embedding Techniques

Finding representations of words, that capture both syntactic and semantic meaning, is one major challenge in the domain of natural language processing (NLP). Learning dense vector representations of words in a multidimensional semantic space as input features for downstream tasks has proven successful for a wide range of tasks such as part-of-speech tagging (POS-tagging), named entity recognition (NER), word analogy tasks or even question answering.

These word representations, also known as *word embeddings*, follow the distributional hypothesis: *"you shall know a word by the company it keeps"* [19], stating that words with similar semantics should occur in similar contexts. Inspired by this hypothesis, researchers developed methods to learn word embeddings by word co-occurrences, exploiting the distributional property to capture word meaning in multidimensional vector space. Ultimately, the learned word vectors with similar semantics should also have similar vectors. One interesting property of these representations is, that they allow simple algebraic vector operations to compute new word representations and explore semantic regularities. For example, by taking the vector representation of the word "king", subtracting "men" and adding "woman", a word vector is calculated that is very close to the vector for the word "queen" [1].

The following systematic literature review was conducted to give an overview of existing techniques to learn word and sentence embeddings and to discuss, how these techniques may be applied to the domain of process mining. At first, we summarize our methodology of the review in section 2.1. We then categorize and present the selected studies (section 2.2), before we decide, which embedding techniques we will utilize for our later experiments (section 2.3).

2.1 Methodology

To obtain a wide and objective overview of the research on learning techniques for word and sentence embeddings, this systematic literature review applies the methods described in [29] for planning, conducting and reporting a systematic literature review. Beginning with the formulation of the research questions in subsection 2.1.1, we state the objectives of the review. We then define our search queries for the literature search within scientific databases (2.1.2). Consequently, we present our protocol based on inclusion and exclusion criteria, which we applied to the search results for study selection (2.1.3).

2.1.1 Research Questions

The goal of this systematic literature review is to identify existing research on word and sentence embedding techniques and classify them based on their characteristics. Furthermore, a set of techniques should be selected, that will be used to learn activity embeddings that will be evaluated on next activity prediction in chapter 4. These objectives can be broken down in following three research questions:

1. **RQ1**: "What word and sentence embedding techniques exist in the natural language processing domain?"
2. **RQ2**: "By which characteristics can the identified techniques be classified?"
3. **RQ3**: "Which of the identified techniques look most promising to learn semantic representations, when applied to activities and traces in the process mining domain?"

RQ1 and RQ2 mainly aim at identifying existing research on word and sentence embeddings techniques and providing a sensible overview of different approaches within the research. Finally, the goal of RQ3 is to discuss which of the identified techniques may be the most successful learning meaningful activity and trace embeddings when applied to event logs of business processes, instead of words and sentences in text corpora.

2.1.2 Search Queries

The main literature search was performed with Google Scholar, since Google Scholar is indexing most common scientific literature databases[1]. Additionally, we supplemented our search with two common computer science literature databases (ACM Digital Library, IEEE Explore). Initially, ACL Anthology was also included to supplement the Google Scholar search, as it is one very important literature database in the NLP domain. But due to a lack of exportation options for search results and the fact that studies from ACL Anthology should also be indexed by Google Scholar, we decided to not include ACL Anthology additionally.

[1] For more details, see `https://scholar.google.com/intl/en/scholar/inclusion`

Table 2.1: Summary of Search Results

Database	Search Query	Number of Results
Google Scholar	(intitle:"word embeddings" OR intitle:"word embedding") AND "semantic" AND "vector space" AND ("natural language processing" OR "nlp")	2960
Google Scholar	(intitle:"word representations" OR intitle:"word representation") AND "semantic" AND "vector space" AND ("natural language processing" OR "nlp")	617
Google Scholar	(intitle:"word vectors" OR intitle:"word vector") AND "semantic" AND "vector space" AND ("natural language processing" OR "nlp")	387
Google Scholar	(intitle:"distributed representations" OR intitle:"distributed representation") AND "semantic" AND "vector space" AND ("natural language processing" OR "nlp")	311
ACM	[Title: "word representations" OR Title: "word embeddings" OR Title: "distributed representations" OR Title: "word vectors" OR Title: "word representation" OR Title: "word embedding" OR Title: "distributed representation" OR Title: "word vector"]AND All: "semantic" AND All: "vector space" AND [All: "natural language processing" OR All: "nlp"]	130
IEEE	[Title: "word representations" OR Title: "word embeddings" OR Title: "distributed representations" OR Title: "word vectors" OR Title: "word representation" OR Title: "word embedding" OR Title: "distributed representation" OR Title: "word vector"]AND All: "semantic" AND All: "vector space" AND [All: "natural language processing" OR All: "nlp"]	52

In table 2.1 the search queries, the database on which the query was performed and the number of results of the query, are listed. All search queries were executed on May 23, 2023. Note that the search queries include a title search term with the singular and plural of the terms “word embedding”, “word representation”, “word vector” and “distributed representation”. This was a necessary step to decrease the total number of results and increase the share of relevant results of a naive initial search query without title term. The Google Scholar search did not permit long enough search queries, so the query had to be split into four queries, with singular and plural of one title word in each query. To enhance the comprehensiveness of the search process and achieve a more comprehensive solution, *one step backward-search* (snowballing) was employed on the final selected studies in retrospect. The exact procedure is described in the next section.

2.1.3 Study Selection Process

In order to filter search query results and isolate studies pertinent to Research Question 1 (RQ1) the following inclusion and exclusion criteria are defined:

- **IQ1**: The study introduces a novel word or sentence embedding technique or enhances an existing technique with a concept that significantly diverges from established approaches.
- **EQ1**: The study introduces a representation learning technique, that is not applied to words and sentences.
- **EQ2**: The study introduces a representation learning technique, that integrates external knowledge beyond unlabeled text corpora (e.g., word definitions or sentiment labels) in the learning process.
- **EQ3**: The study introduces a representation learning technique, that largely exploits language-specific characteristics (e.g., word substructure) or merely focuses on improving cross-lingual word embeddings.
- **EQ4**: The study enhances existing techniques only by optimizing hyperparameters of existing techniques.
- **EQ5**: The study enhances existing techniques only by applying post-processing methods, and does not propose a new approach regarding the learning technique.

If IQ1 is fulfilled and none of the exclusion criteria are violated, a study will be included. The study selection process was divided in three phases to efficiently filter out irrelevant work from the large amount of studies from the search queries. Before the selection process, duplicates among queries were eliminated, resulting in 3887 results from Google Scholar, 31 from IEEE and 2 from ACM (that were not among the results from Google Scholar).

1. **Phase 1**: Study selection based on **title**
2. **Phase 2**: Study selection based on **abstract**
3. **Phase 3**: Study selection based on **full-text**

In phase 1 a first screening of study titles was applied to filter out titles that were clearly not related to IQ1, therefore irrelevant. In cases where uncertainty arose, the studies were retained in the selection process, progressing to the next phase. Subsequently, the remaining 316 studies were then filtered based on study abstract in phase 2 and finally from the remaining 83 studies, 25 were selected based on a comprehensive examination of their full-text in phase 3.

To augment the comprehensiveness of the search process, a one-step backward search was conducted on the 25 selected studies. This involved examining the studies that were referenced within the selected studies, employing the same three-phase screening process as before. As a result, a total of 594 studies were identified, accounting for the removal of duplicates and studies already included in the initial database search. Additionally, a criterion was applied whereby a study had to be cited multiple times (more than once) within the resulting collection, based on the heuristic that studies of greater relevance would tend to receive more citations within similar work. As a result, 130 studies remained. After applying the three-phase protocol described earlier to the 130 remaining studies, this resulted in 12 additional studies that were included. In total, together with the initial database search, 37 studies were selected (25 initial search, 12 backward search).

2.2 Classification of Identified Studies

This section aims to address both Research Question 1 (RQ1) and Research Question 2 (RQ2) by presenting a comprehensive classification of the selected studies based on various criteria, providing an overview of the existing word and sentence embedding techniques.

To begin with, the selected studies are categorized into two main groups: word embedding techniques and sentence embedding techniques. Word embedding techniques focus on learning vector representations for individual words, while sentence embedding techniques aim to capture vector representations for entire

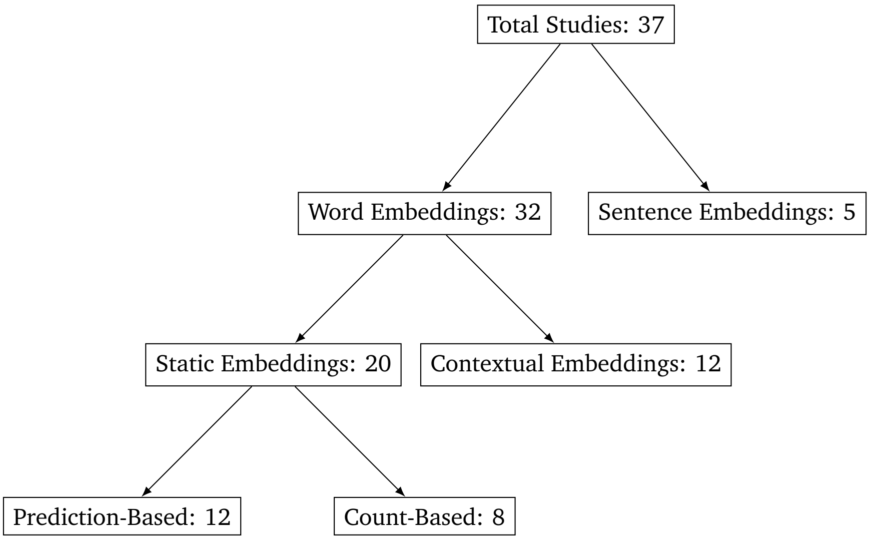


Figure 2.1: Categorization of the identified studies

sentences or paragraphs. Within the category of word embedding techniques, further subgroups are established based on the underlying approaches employed. This includes prediction-based approaches and count-based approaches, which represent static word embeddings. Additionally, contextual word embeddings are considered, and all these groups are organized in chronological order to provide a historical perspective. The tree illustrated in figure 2.1 summarizes the categorization of the selected studies.

Furthermore, studies that propose improvements to already existing techniques are grouped together with the respective existing technique, ensuring a coherent classification of the literature. By applying these classification criteria, a comprehensive analysis is performed on the selected studies, providing a structured and systematic exploration of the diverse landscape of word and sentence embedding techniques. A high-level overview of selected studies can be found in table 2.2 and table 2.3 on the next pages.

2.2.1 Word Embedding Techniques

All word embedding and sentence embedding techniques covered in this work, operate on text corpora, e.g., documents containing words grouped in sentences (separated by dots) and optionally paragraphs (separated by newline or indentation characters). They do not rely on further labelling of text such as POS-tagging, sentiment information or external knowledge about words from dictionaries or lexical databases (e.g., WordNet). As a result, all the discussed techniques are forms of unsupervised learning. In this section, prediction- and count-based techniques for

Study	Categories							
	Word	Sentence	Prediction	Count	Static	Contextual	New Technique	Improvement
Bengio et al. (2003) [30]	✓		✓		✓		✓	
Collobert et al. (2008) [31]	✓		✓		✓		✓	
Collobert et al. (2011) [3]	✓		✓		✓			✓
Mikolov et al. (2013a) [1]	✓		✓		✓		✓	
Mikolov et al. (2013b) [2]	✓		✓		✓			✓
Mnih et al. (2013) [32]	✓		✓		✓			✓
Ling et al. (2015) [33]	✓		✓		✓			✓
Cohen et al. (2018) [34]	✓		✓		✓			✓
Wang et al. (2019) [35]	✓		✓		✓			✓
Song et al. (2018) [36]	✓		✓		✓			✓
Ling et al. (2015) [37]	✓		✓		✓			✓
Sonkar et al. (2020) [38]	✓		✓		✓			✓
Rhode et al. (2006) [39]	✓			✓	✓		✓	
Lebret et al. (2013) [40]	✓			✓	✓		✓	
Lebret et al. (2015) [41]	✓			✓	✓			✓
Pennington et al. (2014) [42]	✓			✓	✓		✓	
Levy et al. (2014) [43]	✓			✓	✓		✓	
Salle et al. (2016) [44]	✓			✓	✓			✓
Stratos et al. (2015) [45]	✓			✓	✓		✓	
Dhillon et al. (2015) [46]	✓			✓	✓		✓	

Table 2.2: Comparison of Word Embedding Techniques in Selected Papers

Study	Categories							
	Word	Sentence	Prediction	Count	Static	Contextual	New Technique	Improvement
Reisinger et al. (2010) [47]	✓					✓	✓	
Reisinger et al. (2010) [48]	✓					✓	✓	
Dhillon et al. (2011) [49]	✓			✓		✓	✓	
Huang et al. (2012) [50]	✓		✓			✓	✓	
Tian et al. (2014) [51]	✓		✓			✓	✓	
Neeklakantan et al. (2015) [52]	✓		✓			✓	✓	
P. Liu et al. (2015) [53]	✓		✓			✓	✓	
Y. Liu et al. (2015) [54]	✓		✓			✓		✓
Bartunov et al. (2016) [55]	✓		✓			✓	✓	
Sun et al. (2017) [56]	✓			✓		✓	✓	
Li et al. (2020) [57]	✓		✓			✓	✓	
Meng et al. (2020) [58]	✓		✓			✓	✓	
Le et al. (2014) [59]		✓	✓		✓		✓	
Li et al. (2014) [60]		✓	✓		✓		✓	
Kiros et al. (2015) [61]		✓	✓		✓		✓	
Hill et al. (2016) [62]		✓	✓		✓		✓	
Pagliardini et al. (2017) [63]		✓	✓		✓		✓	

Table 2.3: Comparison of Word Embedding Techniques in Selected Papers

learning static word embeddings are covered first, followed by techniques to learn contextual word embeddings.

Prediction-based Techniques

One group of word embedding techniques are *prediction-based* techniques. They aim to generate word representations by predicting target words from their context (or vice versa), by feeding this contextual information (the surrounding words) as input to a neural network architecture. Therefore, the words are first one-hot-encoded and fed to the input layer, before they are mapped to their corresponding d-dimensional vectors in a projection layer. The number of dimensions d of the vectors can be selected arbitrary. Finally, the neural network is trained on (large) text corpora, and during the training process, the vectors in the projection layer, are updated to optimize the objective function, which learns the contextual dependencies between words.

By considering the context of words in a defined window size, these techniques capture the distributional properties of words by their co-occurrence and represent them as dense vectors in a high-dimensional semantic space, where the learned vectors of similar words tend to be also similar. After the learning process, the learned weight vectors for each word in the projection layer can be directly used as word embeddings.

A popular early model applying a neural network setup to a language modeling task is proposed in [30] as the *Neural Probabilistic Language model* (NPLM). It predicts the probability of a word given its context, taking into account a fixed number of previous words. The NPLM uses a hidden layer with non-linear activation functions to capture complex dependencies of previous words influencing the probabilities of the next word. Again, the learned vectors of the words after training the network, can be directly used as word embeddings, which represent the dependencies between (left) context of preceding words, to target word (the next word).

Later work, as in [31] and [3] show the effectiveness of pretrained word embeddings as inputs on a variety of natural language processing tasks. They use a complex convolutional network setup to learn word embeddings by leveraging “fake” center words. Here the center word of a context window of predefined length is replaced with a random word and then the neural network predicts if the sequence is correct (original text) or not (“fake” center word). The learned weights of the neural net (word embeddings) are then shared for several NLP tasks.

Arguably, the two most influential models for learning high quality semantic word embeddings were proposed in the paper [1], the *Continuous Bag-of-Words*

(CBOW) and the *Skip-Gram* model, which both are implemented in the popular *Word2Vec*[2] software package introduced by Google. Both models use target words and surrounding (left and right) context words of the target word in the learning process. Interestingly, the feedforward neural network architecture does not contain a non-linear hidden layer, significantly lowering computational cost and enabling training with large corpora. The two architectures of the CBOW and Skip-Gram model are visualized in figure 2.

The CBOW model uses the average of the context word vectors of the target word as input and predicts the probability of target words as output in a feedforward neural network setup. Contrary to the CBOW model, the Continuous Skip-Gram model uses the target word as input and predicts the left and right context words as output, but only one context word at a time, so we have $2l$ predictions of (target, context)-pairs, when l is the size of the symmetric window. The Skip-Gram model performs essentially the inverse task of the CBOW model. Both models assign a probability to each word in the vocabulary (words seen in the corpus) in their prediction, using a hierarchical version of softmax in the output layer.

The input for the softmax is the dot product of the context vector v_c (averages of all surrounding context vectors) and the target word vector v_w for CBOW. For Skip-Gram, it is simply the dot product of the target word vector and the context vector of the one word, we are trying to predict at the time. Fundamentally, the learning objectives of CBOW and Skip-Gram are maximizing the dot product between frequently occurring context-target pairs, as this also maximizes their softmax probabilities.

Therefore, their vectors are shifted, so that they point in a similar direction (remember, the dot product of two vectors is maximal, when they point in the same direction, and zero, when they are orthogonal), when updating them by backpropagation in neural network training. Ultimately, these objectives model the distributional hypothesis, since words sharing similar contexts, also have similar word vectors.

The first improvement of Skip-Gram was published by the authors of the original paper themselves in [2], using subsampling of frequent words and negative sampling as an alternative to the softmax layer, resulting in better quality word embeddings and faster training times. Subsampling discards potentially less semantically meaningful high frequent words (e.g., “the”) with a higher probability in the training process, while negative subsampling randomly selects words that do not appear within the context of the target word as negative samples, reducing the training objective to only classify a word as negative (not in context) or positive (in context) given the target word. This significantly lowers the training cost, as we do not

[2] `https://code.google.com/archive/p/word2vec/`

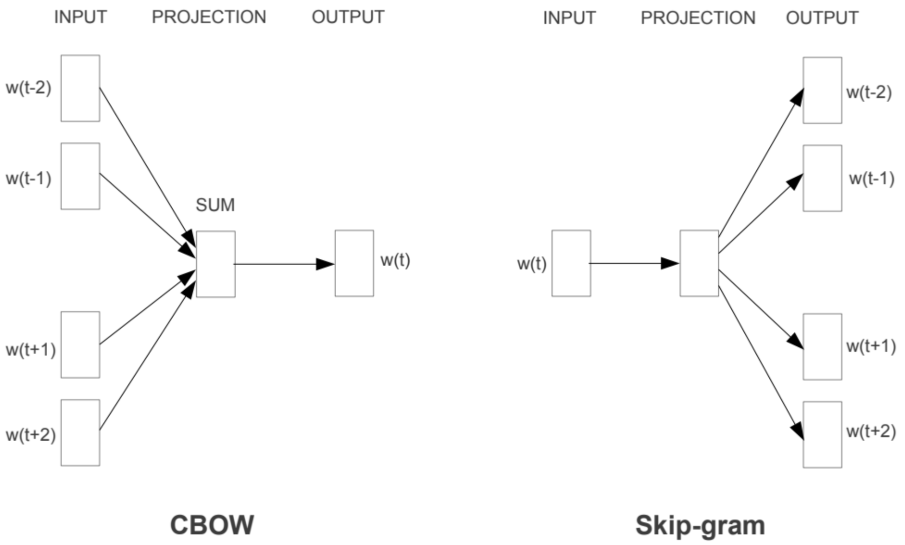


Figure 2.2: The CBOW and Skip-Gram architecture, original figure from [1]. The CBOW architecture is taking the context words as input and computes the sum of the context vectors to output the target word. In contrast, the Skip-Gram architecture takes the target word as input and predicts the surrounding context words.

have to compute probabilities in an expensive soft max layer for every word in the vocabulary. Another interesting approach called noise-contrastive estimation (NCE) comparable to the negative sampling procedure was explored by [32] with the proposed inverse log-bilinear language model (ivLBL), which is also closely-related to the Skip-Gram model.

Since then, other researchers also proposed several improvements to the original versions of CBOW and Skip-Gram models. The naive models neglect word order, treating every context word the same way, regardless of their relative position to the target word, which results in a loss of relevant information for learning word embeddings. In [33], [34], [35] this information is included in advanced models using different approaches. Moreover, explicitly distinguishing between left and right context words by an additional direction has also shown potential improvements ([36]). The work from [37] and [38] apply weighting of context words by their relevance for the prediction of the target word in the CBOW model. The weighting of context words is determined by a relevance score given by an attention model, similar to the one first introduced for machine translation in [64]. In [65] pre-trained contextual word embeddings, as described in a later section, are leveraged to learn static embeddings in a modified version of CBOW.

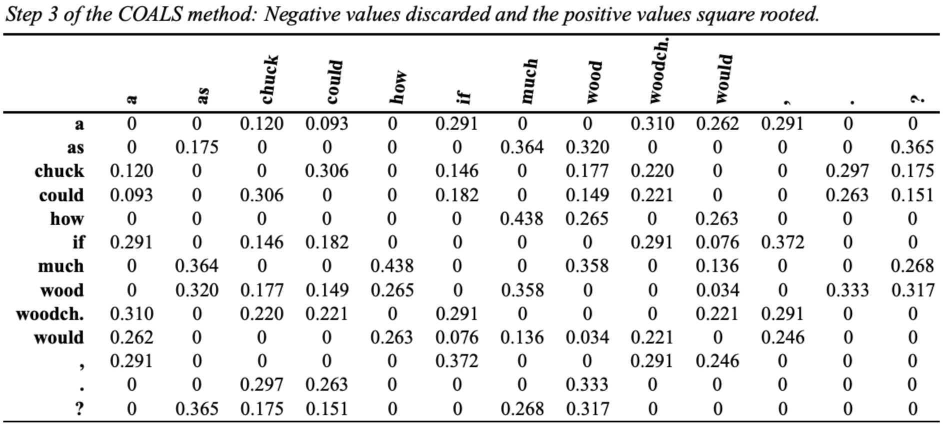
Count-based Techniques

Count-based techniques offer an alternative approach for acquiring word embeddings. These techniques generally derive word embeddings by applying matrix factorization methods to a global word-word co-occurrence matrix, leveraging statistical information about the corpus. The rows and columns from the co-occurrence matrix correspond to the words from the vocabulary, therefore the matrix is quadratic with shape $|V| \times |V|$, where $|V|$ is the number of words in the vocabulary. The entries of the matrix represent how often a word occurs in the context of another word, so for example the matrix at row w_i and column w_j, denotes how often w_j occurs in the context of w_i. The context is given by a local context window, with predefined size. Matrix factorization then decomposes the original word-word co-occurrence matrix into low-rank matrices, that capture the most relevant features describing the observed co-occurrence patterns. The extracted feature values can subsequently serve as embeddings for each word.

Count-based techniques have an extensive history, in contrast to the prediction-based techniques that rather emerged more recently. The first model that is considered here, is the *Correlated Occurrence Analogue to Lexical Semantic* (COALS) model [39], which is based on two popular early techniques for deriving word embeddings from word co-occurrence statistics, the *Hyperspace Analogue to Language* (HAL) [66] model and *Latent Semantic Analysis* (LSA) [67], [68].

The four (simplified) main steps are as follows: the COALS model builds a word-word co-occurrence matrix like the HAL model (and as described in the beginning of this section), but with a weighting factor, assigning closer words in proximity to the target word a higher weight. The low-frequency word columns are eliminated (due to high noise) and the high frequency words (like "the") are normalized in respect to their frequency. Then the word-word frequencies are converted to Spearman correlations, setting negative correlations to 0 and squaring positive values. An example matrix after this step can be found in figure 2.3. Finally, Singular Value Decomposition (SVD), a matrix factorization method and dimensionality reduction technique, can be applied to the matrix (as in LSA), which captures the most relevant features by variance. Again, the resulting feature values for each word can then be interpreted as their embeddings. In other techniques [40], [41] word embeddings are extracted by performing Principal Component Analysis (PCA) on the word-co-occurrence matrix, while minimizing the reconstruction error based on the Hellinger Distance between words.

In [42] the *Global Vectors for Word Representation* (GloVe) model was introduced, combining ideas from the well-established count-based models and their recently developed prediction-based models, to further enhance the quality of semantic word



Step 3 of the COALS method: Negative values discarded and the positive values square rooted.

	a	**as**	**chuck**	**could**	**how**	**if**	**much**	**wood**	**woodch.**	**would**	**,**	**.**	**?**
a	0	0	0.120	0.093	0	0.291	0	0	0.310	0.262	0.291	0	0
as	0	0.175	0	0	0	0	0.364	0.320	0	0	0	0	0.365
chuck	0.120	0	0	0.306	0	0.146	0	0.177	0.220	0	0	0.297	0.175
could	0.093	0	0.306	0	0	0.182	0	0.149	0.221	0	0	0.263	0.151
how	0	0	0	0	0	0	0.438	0.265	0	0.263	0	0	0
if	0.291	0	0.146	0.182	0	0	0	0	0.291	0.076	0.372	0	0
much	0	0.364	0	0	0.438	0	0	0.358	0	0.136	0	0	0.268
wood	0	0.320	0.177	0.149	0.265	0	0.358	0	0	0.034	0	0.333	0.317
woodch.	0.310	0	0.220	0.221	0	0.291	0	0	0	0.221	0.291	0	0
would	0.262	0	0	0	0.263	0.076	0.136	0.034	0.221	0	0.246	0	0
,	0.291	0	0	0	0	0.372	0	0	0.291	0.246	0	0	0
.	0	0	0.297	0.263	0	0	0	0.333	0	0	0	0	0
?	0	0.365	0.175	0.151	0	0	0.268	0.317	0	0	0	0	0

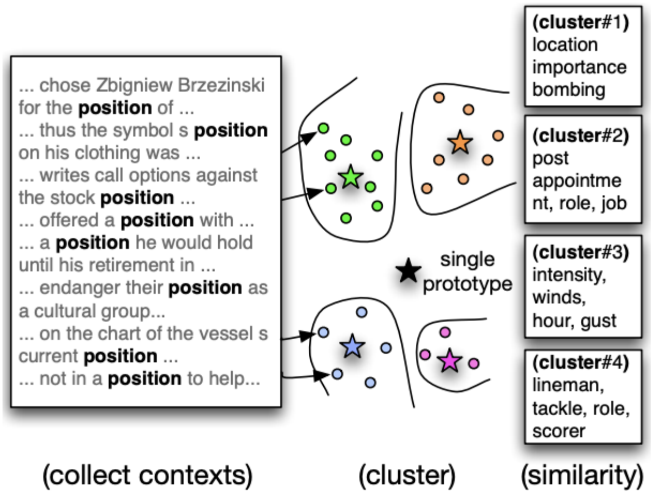
Figure 2.3: The COALS matrix from [39] after converting the weighted word-word co-occurrences to Spearman correlations, squaring positive values and eliminating negative values. To obtain word embeddings of smaller dimension, one can apply Singular Value Decomposition (SVD) to this matrix.

embeddings. The objective of the GloVe model is to minimize a proposed weighted least squares cost function. The cost function mainly consists of the difference between the dot product of the word vector of w_i and the context vector of w_j and the logarithm of the observed co-occurrence ratios between w_i and w_j for all words. Since the dot product is a measure of similarity between two vectors in the multidimensional space, the similarity of two word vectors will align with their observed global co-occurrence ratios, as the cost function is optimized.

The work of [43] bridges the gap between prediction-based and count-based techniques, by showing that the objective of Skip-Gram with Negative Sampling (SGNS) from [2] is implicitly factorizing a shifted Pointwise Mutual Information (PMI) matrix (derived from the co-occurrence matrix). The researchers also come to a similar result for the NCE technique from [32]. Following their observations, they propose a novel sparse Shifted Positive PMI word-context matrix, as a measure of association between words and utilize SVD as described earlier to extract word embeddings. Building on this study, in [44] a loss function is proposed that weights errors on frequent words more heavily, in contrast to SVD, and also incorporates negative co-occurrences in the learning process, inspired by SGNS. In [45] the SVD-based technique Canonical Correlation Analysis (CCA), is applied on a weighted bi-gram word co-occurrence matrix, while another study [46] performs a two-step CCA on left and right context matrices of respective center words to extract word embeddings.

Contextual Techniques

One disadvantage of static word embeddings is, that they are unable to capture multiple senses of words. For example, the polysemous word “mouse” could be interpreted as a small animal in the sentence “The mouse is eating a piece of cheese” or as a computer input device in the sentence “My computer screen is frozen, I can’t move the mouse cursor”. The previously discussed static techniques only learn one embedding per word, and therefore they are unable to distinguish between multiple word meanings. To overcome this issue, multiple embeddings for polysemous words are necessary, where the most appropriate word embedding will then be selected, depending on the context of the word. This concept is employed in *contextual word embeddings*.


... chose Zbigniew Brzezinski for the **position** of ...
... thus the symbol s **position** on his clothing was ...
... writes call options against the stock **position** ...
... offered a **position** with ...
... a **position** he would hold until his retirement in ...
... endanger their **position** as a cultural group...
... on the chart of the vessel s current **position** ...
... not in a **position** to help...

single prototype
(cluster#1) location importance bombing
(cluster#2) post appointme nt, role, job
(cluster#3) intensity, winds, hour, gust
(cluster#4) lineman, tackle, role, scorer
(collect contexts)
(cluster)
(similarity)

Figure 2.4: The multi-prototype approach applied for the single target word “position”, original figure from [47]. The four cluster centroids are used as prototype vectors for the corresponding context occurrences to discriminate word sense. Note, that for static, single prototype embeddings, the vector would be the centroid of all occurrences and could not distinguish the various word senses.

In the study [47], for each word, context vectors are determined by word sense discovery, as described in [69]. Subsequently, these context vectors are clustered, with the idea in mind, that for polysemous words, a cluster group of similar contexts for each sense will emerge. An example from [47] for the word “position” is depicted in figure 2.4. In this technique, the average vector of each cluster group will be included in a set of sense-specific prototype vectors for each word (the contextual embeddings). Finally, the prototype vector with the minimum distance to the vector of the specific context the word appears in, will be selected as word embedding.

With the extension of the basic model with tiered clustering, the same researchers further improve the model for highly polysemous words by including context-independent features from a background model for the prototype vectors [48]. This adjustment is based on the observation, that for highly polysemous words like "line" or "run" the senses for the words have a common metaphoric background and thus might also share some context-independent features. In an earlier study [49] from the authors of [46], the authors explored how concatenation of left-context, right-context and target word embeddings derived from CCA can be used to generate contextual embeddings.

Another interesting approach integrating both global context and local context for a word is explored in [50]. Here, the global context is encoded in a document embedding, which is computed as a weighted average of all word embeddings occurring in the document. The words are weighted by their importance by Inverse Document Frequency (IDF). Following the approach of [31], the training objective is to differentiate the correct last word from a sequence of words from random word replacements, given both local context embeddings and the global embedding of the document. In order to learn contextual embeddings, the approach from [47] is adapted, relabeling all words to associate them with their respective cluster group. The learning procedure can then be applied without further changes, as all instances of one word labeled with the sense, form one entity with its own embedding.

Since the introduction of the Skip-Gram model in [1], [2], several studies extended the original static embedding models to learn contextual embeddings. In [51] a finite mixture model, capturing the different senses of a word, is utilized to express the probability a context word is observed, given the target word and its prototype (word sense) probabilities. Again, by using one embedding for each word prototype, this model will learn both the probability of each word sense of the target word and also, like in the original Skip-Gram model, the embedding by predicting the context of the target word, by the weighted average of its word prototype probabilities (high probability word prototypes have more weight).

Combining the clustering word sense discovery approach with the Skip-Gram model, the authors of [52] propose a non-parametric version, where the number of clusters (word prototypes) is not fixed. In this technique, first the word prototype is predicted given its context words and then the word embedding for the word prototype is learned. New word prototypes are created, when the context vector representation distance to all existing clusters (word senses) is above a certain hyperparameter threshold. A closely related non-parametric Bayesian approach can be found in [55].

The concept of Topical Word Embeddings (TWE) is presented in [53], where the authors first learn topic assignments for each word depending on their contexts, following the Latent Dirichlet Allocation (LDA) from [70]. The authors take the word “apple” as an example, because it can have the meaning of a fruit under the topic “food” or the meaning of the famous IT company under the topic “information technology”. With this additional topic information, they propose different variants to learn contextual word embeddings with the Skip-Gram model. A comparable approach is employed in [54], while [57] utilizes the CBOW model with an attention mechanism, to determine relevance of context words. In [58] CBOW and Skip-Gram variants are proposed, that extend the objective functions to include a loss term corresponding to the global context, modeled as document embedding.

A very simple contextual embedding technique is explored in [56], where static embeddings are learned first, before contextual embeddings for a target word are computed as a weighted sum of the static base embeddings of each context word, where the weighting factor is derived from a normalized co-occurrence matrix.

In recent years, sophisticated deep neural network architectures, so-called *Large Language Models* (LLMs), were developed by both independent researchers and prominent technology companies and gained wide popularity, due to their huge advancements in the field of NLP. Popular examples are the Bidirectional Encoder Representations from Transformers (BERT) model from Google [4] and the Generative Pre-trained Transformer (GPT) model from OpenAI [71]. These transformer-based models also learn contextual word embeddings in their pre-training phase by utilizing the attention mechanism of [72] to determine the relevance of surrounding words for the prediction of a target word (in a masked word prediction task for BERT and next word in a sequence prediction task for GPT). While these LLMs introduce many new interesting concepts for learning contextual word embeddings and therefore are worth to mention briefly, they are not explored further in the scope of this master thesis. This decision stems from their immense complexity and that they often rely on language specific information in their techniques (splitting words in subwords, stemming, etc.), which violates EQ3.

2.2.2 Sentence and Paragraph Embedding Techniques

This section provides an overview of existing sentence or paragraph embedding techniques. The goal is not to learn vector representations for a single word like in the last section, but rather vector representations for an entire sentence or paragraph. While a few related ideas of the word embedding techniques can be applied as well here, there are still adjustments to be made to learn high-quality embeddings. Again, the studies are presented in chronological order.

In 2014, one year after the introduction of the CBOW and Skip-Gram models, Le and Mikolov published another paper [59], this time focusing on vector representations of text with variable length, e.g., sentences, paragraphs, or even entire documents. To learn the embeddings, a neural network setup is proposed, which objective is to predict the next word, given the concatenation or average of the vectors of the preceding context word and an additional paragraph vector (see figure 2.2). The context words and target words are generated by sliding a window over the entire paragraph and thus are varied, meanwhile the paragraph vector does not change, except for other paragraphs. The idea to use the same paragraph vector for different contexts in the same paragraph, is that the paragraph vector should remember and represent the respective topic or meaning of the paragraph after some learning steps. Therefore, the model is called *Distributed Memory Model of Paragraph Vectors* (PV-DM).

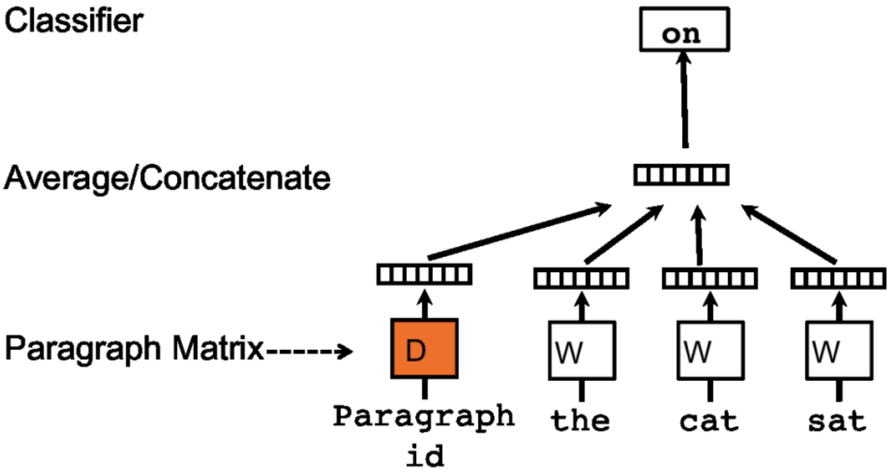


Figure 2.5: The PV-DM model depicted in the study [59] from Le and Mikolov. To predict the next word in a sequence, the concatenation or average of preceding context word vectors is computed with an additional paragraph vector. The paragraph vector contributes paragraph information, that is shared in the prediction task among different contexts in the same paragraph.

Another interesting approach is presented in [60], where a sentence representation is calculated in a recurrent neural network setup. Starting with the first word in a sentence, the sentence embedding is calculated by combining the initial sentence vector with the current word. These sentence embeddings are then used as input for a feedforward neural network in a clique coherence task to predict whether the sentences are coherent (order taken from the original text) or are not coherent (random generated permutations from any sentence of the original text).

In the *Skip-Thought Vector* model from [61] the learning objective is to predict the preceding s_{i-1} and the following sentence s_{i+1} for a given source sentence s_i. Again, a recurrent neural network is utilized to “encode” the words from the source sentence in a single sentence representation and then “decode” the surrounding sentences. For each word in the target sentence (either s_{i-1} or s_{i+1}) a probability is calculated in a softmax function over all words from the vocabulary and the cost is then the sum of the negative log-likelihood of each correct (original word from the corpus).

The study [62] proposes two new sentence embedding techniques based on earlier techniques. The first technique, *Sequential Denoising Autoencoders* (SDAE), tries to recover an original sentence from a corrupted sentence, where words from the original sentence are either deleted or swapped with predefined probabilities. After the learning process, the sentence representation should hold the features of the sentence that explain its important factors of variation ([73]). The objective of the second technique, *FastSent*, is to predict the words from the adjacent sentences s_{i-1} and s_{i+1} with the softmax function given a source sentence s_i, similar to the Skip-Thought Vector model. But here, in contrast to the Skip-Thought Vector model, a simple Bag-of-Words representation is used to represent the source sentences, which is just the sum of the embeddings from the words in the source sentence. This adjustment leads to faster training times compared to the costly gradient backpropagation in the recurrent network setup from the Skip-Thought Vector model, opening up the possibility of training on larger corpora. They also propose another variant, *FastSent+AE* where the encoded source sentence representation also predicts its own words in addition to the words of the adjacent sentences.

In the work [63] another technique called *Sent2Vec* was introduced. It has a similar objective as the CBOW model, as *Sent2Vec* predicts a target word given context words for a sentence (or paragraph). In addition to the context and target embeddings, the model also learns n-gram embeddings of words, therefore taking into account different word compositions. The final sentence embedding is then simply defined as the average of the word and n-gram embeddings of the sentence. They also utilize a shallow feed-forward neural network architecture instead of a complex deep architecture. Ultimately, this leads to faster training times, when combined with negative sampling to approximate the otherwise computational expensive softmax probabilities of words, as shown in [2].

2.3 Selection of Techniques

After giving an overview of existing word and sentence embedding techniques, the final goal of this systematic literature review is to select embedding techniques, that will later be used for the exploration of interpretability of the learned activity embeddings and for the following performance evaluation of activity embeddings in next activity prediction. We decided to use the CBOW and Skip-Gram models from [1] for the following reasons: simplicity, effectiveness, and context of evaluation.

Firstly, the CBOW and Skip-Gram models are both simple in their architecture, as they both only use shallow neural networks architectures (architectures with no additional hidden layers) to learn embeddings. Also, their objective, the prediction of one target word from context words for CBOW or vice versa, the prediction of context words from one target words for Skip-Gram, is very straightforward. It is easy to relate these learning objectives to the distributional hypotheses for words of [19] (words with similar context, are similar in meaning) and to understand how they produce word vectors inline with this linguistic property. Also, there are preexisting implementations of these techniques, such as the Word2Vec functionality of GenSim [74]. We used this library for our work, because it is fast, due to highly optimized C routines, provides an easy-to-use Python interface (the language we used for our code as well) and, most importantly, it is adaptable to learn embeddings for process activities.

Another important reason was their already proven effectiveness in the NLP domain. They largely improved the performance on semantic relationship tasks and showed solid performance on syntactic task for words ([1], [2]). Even though they do not incorporate word order (e.g., position of the context word relative to the target word), in contrast to other techniques that leverage Recurrent Neural Networks (RNN) or Long Short-Term Memory Networks (LSTM), this did not seem to make a major difference when comparing their performance on syntactic tasks. Since embedding techniques are still very new in the field of process mining and to the best of our knowledge, there has not been a real comparison of activity embedding quality for different embedding techniques, this makes them a solid starting point for initial experiments.

Finally, the goal of this master thesis is not to give an extensive comparison of embedding techniques, but to explore the interpretability of activity embeddings per se, and evaluate how they might affect performance, when they are used as an activity encoding technique for a process mining task. We are aware, that the usage of different embedding techniques or the adaptation of embedding techniques to the process mining domain, may produce different results in performance evaluation. But for the scope of this master thesis the two techniques we selected (CBOW and Skip-Gram), should be sufficient for our analysis.

3 Interpretability of Activity Embeddings

In the last chapter, we gained perspective of the wide range of existing word and sentence embedding techniques from NLP, and we selected the two models, CBOW and Skip-Gram, to learn activity vectors for our empirical experiments on next activity prediction in chapter 4. But first, we want to examine the interpretability of the learned activity embeddings in this chapter. Eventually we will see, if the techniques are able to capture process semantics in the activity vectors to the same degree, as they are able to capture language semantics in word vectors in NLP.

In the NLP domain, the quality of the learned embeddings can be measured as performance in word similarity tasks, for example WordSim-353 [75] or SimLex-999 [76]. These tasks often compare the calculated similarity of two embedding vectors with human assigned similarity scores of the two corresponding words.

When we want to measure similarity for activity embeddings, we may also first need to think about, what kind of activities should be similar to each other (and which should be not). Since there is very little research on learning embeddings in process mining, there is no gold standard task for evaluation of the learned embeddings like in the NLP domain. For words, we have a common understanding of semantic similarity, since we can relate the concepts the words describe, or we have word associations. For example synonyms (e.g., "cup" and "mug"), very similar words (e.g., "alligator" and "crocodile") or just somehow related words (e.g., "car" and "crash" are often associated with each other) are similar in their embeddings, since frequently appear in similar contexts, as implied by the distributional hypothesis [19]. We often share common beliefs about word concepts and associations, which is reflected in high agreement and low variance in the results of the human rated similarity scores.

For activities, we could think of similarity in terms of their shortest path in the process model or their shortest path to a common activity, if they are on separate branches (e.g., for exclusive or parallel splits). We might learn activity embeddings with this property, because in order to appear in similar context windows, the activities need to be close enough together in distance by shortest path to a common third activity, such that the activity occurs in both context windows. To explore the interpretability of activity embeddings, we look at three simple synthetic examples in section 3.1 and one real world event log of the Business Process Intelligence (BPI) Challenge 2017 in section 3.2.

3.1 Synthetic Event Logs

We constructed three synthetic event logs based on three simple business process models, the sequential model is depicted in figure 3.1, the exclusive model in figure 3.2, and the parallel model in figure 3.3. We learned embeddings for all six present activities in the event logs with the Skip-Gram model [1] from the Word2Vec package of GenSim [74] with window size 2. All the event logs were generated with equal proportion to each possible trace variant, with the log generation framework purple from [77], freely accessible at [78].

For the process models, we wanted to inspect how the embeddings change when introducing the two main types of process gateways, exclusive and parallel gateways. For illustration, we reduced the dimension from 8, the original embedding dimension, to two dimensions by Principal Component Analysis in figures, 3.4, 3.5 and 3.6. Before we start examining the embeddings in two-dimensional space, we should note that some information is lost inevitably to dimensionality reduction by PCA. The Principal Components (PC) 1 and 2 can only account for their summed percentage of explained variation (around 70% for the three different embeddings). As a measure of similarity, we take the commonly used cosine similarity (see e.g., [1, 79, 76] for similarities of word pairs) of two activities with vectors u and v, defined as their normalized dot product:

$$\text{cos(u, v)} = \frac{u \cdot v}{\|u| \cdot \|v\|} \tag{3.1}$$

The cosine similarity computes the angle of two vectors (in other words, the similarity of vector directions), where the similarity value is between -1 (exactly opposite) and 1 (exactly same). A similarity value of 0 can be interpreted as completely unrelated (orthogonal vectors). The metric is invariant to the length of the vectors, which would not be the case if we would simply use the Euclidean distance, for example [79].

When we now look at the embeddings of the sequential example in figure 3.4, the hypothesis that activities close together in their shortest path should cluster together, does not appear to be true. In fact, there do not seem to be any trivial patterns, that can relate the position of the embeddings in vector space to their position in the process model. For example F and C are somewhat close together (cosine similarity of 0.323), however F should never appear in the context window of C and vice versa, since their distance 3 is greater than the context window 2.

Meanwhile, D is the most dissimilar to C (cosine similarity of -0.610), even though they are direct neighbors. This can be a result of the indifference of relative position in the Skip-Gram model, because there is no higher weighting of context, that is closer in distance to the target (e.g., when predicting context for C, B and D with distance 1 are weighted equally like A and E with distance 2). Also, we should remember, that two activities that appear in the context of each other (and therefore are also close in distance) are not guaranteed to cluster, but rather only if they share a large proportion of context. E.g. C and D with distance 1 would appear both in the context of each other, but their context overlap is only of size 2, {B, E}. When we compare B and E with distance 3, their context overlap would also be of size 2, {C, D}. This leaves the Skip-Gram model with very little discriminative information for the activities in the sequential example. We will try to formalize this context overlap in more detail in the next section for the real world event log.

If we look at the embedding of the exclusive example in figure 3.5 however, we can observe that embeddings for C and E on the last position of the two exclusive branches appear similar (cosine similarity of 0.282). This holds true for most embeddings of activities of the two opposite exclusive branches (or at least they share similar distance relations), since they appear in similar contexts. A and F are always part of the two exclusive trace variants [A, B, C, F] and [A, D, E, F] and inside the context window for B, C, D and E. Therefore the Skip-Gram target contexts have higher overlap, and they are moved closer together in vector space. We also make a similar finding for the embeddings of the parallel example in figure 3.6, although there are now 6 possible trace variants. But in all variants, A is first before either B or C occur, and F has to follow after D and E.

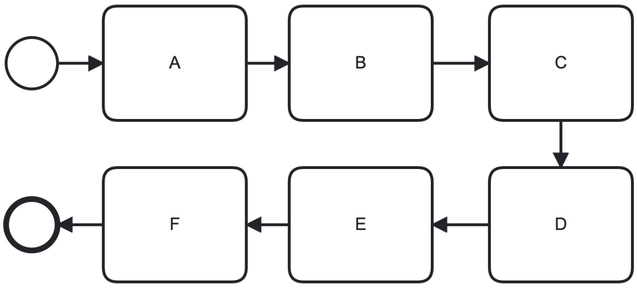


Figure 3.1: An example model of a simple sequential process

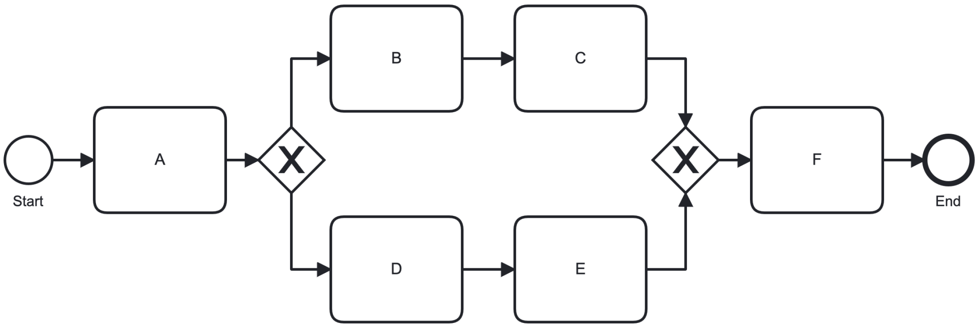


Figure 3.2: An example model of a process with an exclusive gateway

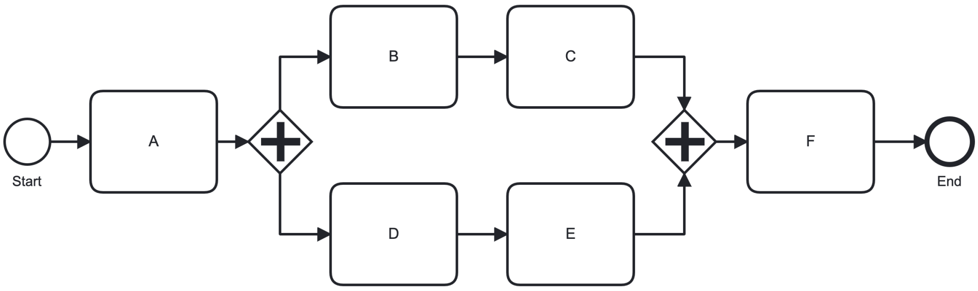


Figure 3.3: An example model of a process with a parallel gateway

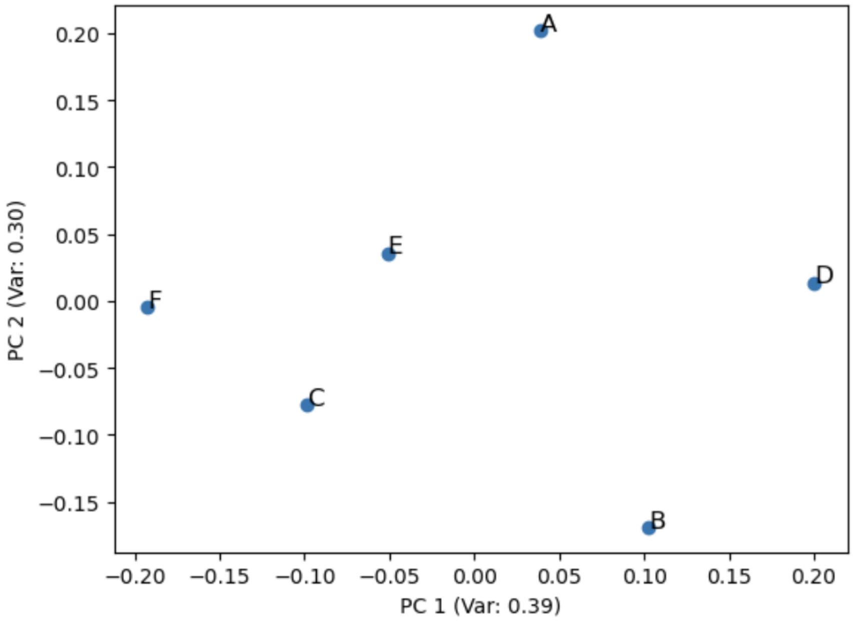


Figure 3.4: Activity embeddings for the sequential event log with dimension 8, reduced to two dimensions by PCA

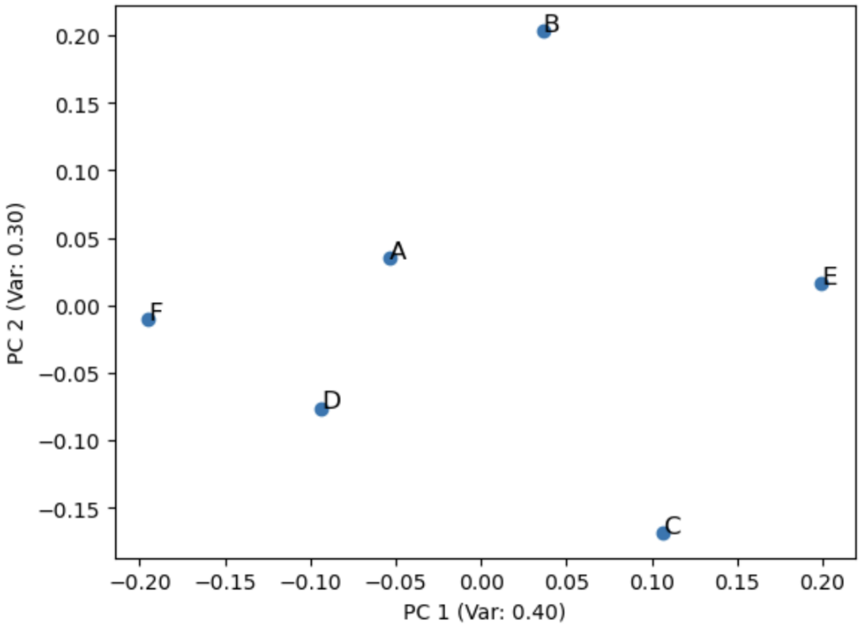


Figure 3.5: Activity embeddings for the exclusive event log with dimension 8, reduced to two dimensions by PCA

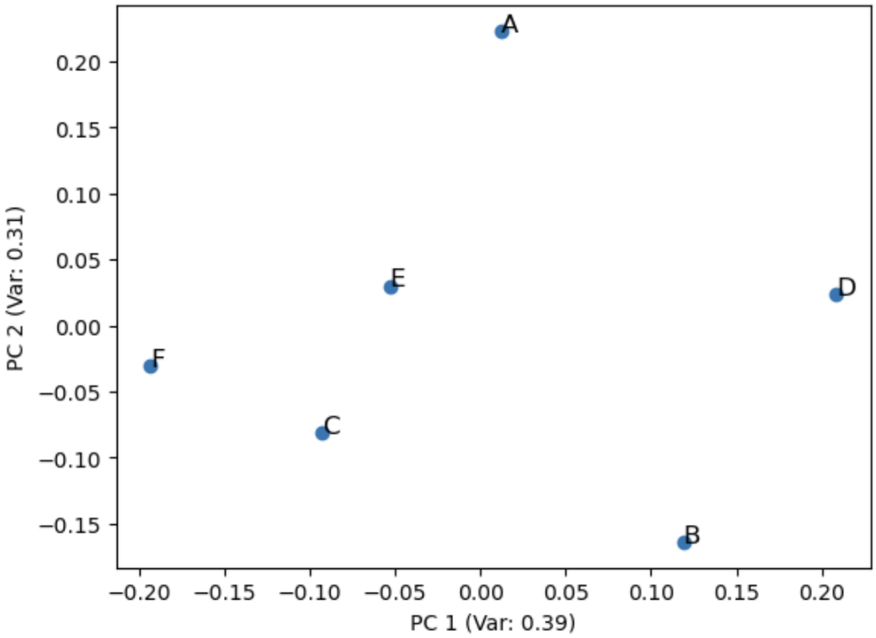


Figure 3.6: Activity embeddings for the parallel event log with dimension 8, reduced to two dimensions by PCA

3.2 Real World Event Log

Following the first insights on the synthetic example event logs, let us consider the Skip-Gram embeddings learned on a real world event log of the BPI Challenge 2017 from [80] now, which contains traces of a loan application process from a Dutch Financial Institution. Although we do not need any domain knowledge about the process for our purposes (only the control flow information depicted in later figures), we refer to the work of [81] or [82] for a detailed analysis of the event log and the underlying process.

We imported the event log in the process mining tool Disco [83] (available under [84]) for analysis. The process map depicted in figures 3.9, 3.10, 3.11 and 3.12 at the end of the section was automatically discovered by Disco and gives an overview of the loan application process and the absolute frequencies of activities and activity connections. Activity names starting with "A_" correspond to activities regarding the state of the application, names starting with "O_" to activities regarding the state of the offer and names starting with "W_" to activities regarding the state of the work item in the loan application process.

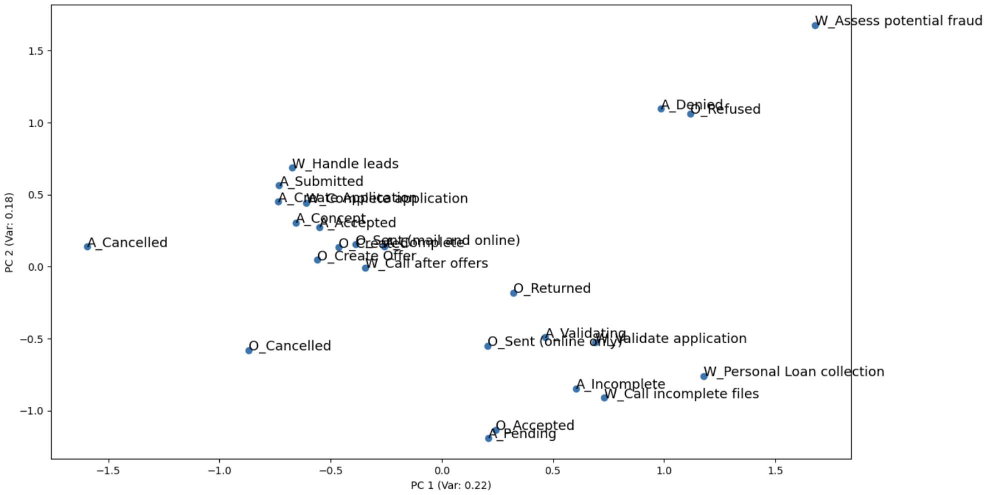


Figure 3.7: Activity embeddings for the BPI 2017 event log with dimension 32 learned by the Skip-Gram model from the Word2Vec package of Gensim [74] and reduced to two dimensions by PCA

Again, we learned embeddings for the activities with the Skip-Gram model from the Word2Vec package of Gensim as before, but we set the window size to 5 and the embedding dimension to 16. The activity vectors were reduced to two dimensions by PCA. The results can be found in figure 3.7. Additionally, we build a heatmap representing the cosine similarity matrix of the activity embeddings, which is shown in figure 3.8. The colors of activity pairs indicate their degree of similarity. White to yellow are indicating a high similarity score for a pair of activities, orange to light red are indicating medium to low similarity scores, and dark red to black are indicating similarity scores close to zero or even negative.

When looking at figure 3.7, we see a clustering of activities in the middle left from the “starting” branch (activities “A_Create Application” to “O_Sent (mail and online)” in figure 3.9 and 3.10) and activities of the left branch that follows after “O_Sent (mail and online)” (figure 3.11). Unfortunately, the vectors are very close together in figure 3.7, so that we can not always clearly read the annotated activity names. For these cases, we refer to the heatmap in figure 3.8 for a more detailed and readable view of the similarities of densely clustered activities.

But if we look closely, we can see that the activities of the “starting” branch (upper left of the cluster) are a little separate from the left following branch after “O_Sent (mail and online)” (bottom right of the cluster). We link the clustering of these two branches with the very frequent direct path from “W_Complete Application” to “W_Call after offers” and the fact that after the activity “O_Created” the activity “O_Sent (mail and online)” (left branch) follows more frequently than “O_Sent (online only)” (right branch).

The cluster formed by the activities from the right branch (figure 3.12), which follow after the activity “O_Sent (online only) from the “starting” branch, or after the activity “W_call after offers” from the left branch, is less dense, but some activities of subgraphs are still close together (e.g., “O_Accepted” and “A_Pending” or “A_Denied” and “O_Refused”). In general, we can observe two trends i) activities, that are close together in direct path distance or indirect distance to a third common activity in the process map and ii) activities that are connected through paths with higher relative frequencies, also tend to have vectors with higher cosine similarity.

To explain these two findings, we may again briefly review the objective of the Skip-Gram model introduced in section 2.1.1, which was used to learn the embeddings. The Skip-Gram model takes a target, activity a_i, and predicts the surrounding context, activities $a_{i-l}, ...a_{i-1}$ to the left of the target and $a_{i+1}, ..., a_{i+l}$, $2l$ activities in total with symmetric window size l. This objective models the distributional hypothesis by maximizing the dot product of frequently occurring target-context pairs and ultimately activities, which share similar contexts (and therefore meaning) are clustered together.

This objective already explains finding i), because for two activities a_i, a_j to share context, a third activity a_k, the shared context activity, a_k needs to be in both the context window of a_i and a_j. We define the distance $d(a_i, a_j)$ of two activities a_i, a_j, as their shortest path in the graph. Then two activities a_i, a_j share the context activity a_k, when $d(a_i, a_k), d(a_j, a_k) \leq l$, for window size l. Generally, activities close together in distance have a higher potential for sharing more context activities (since

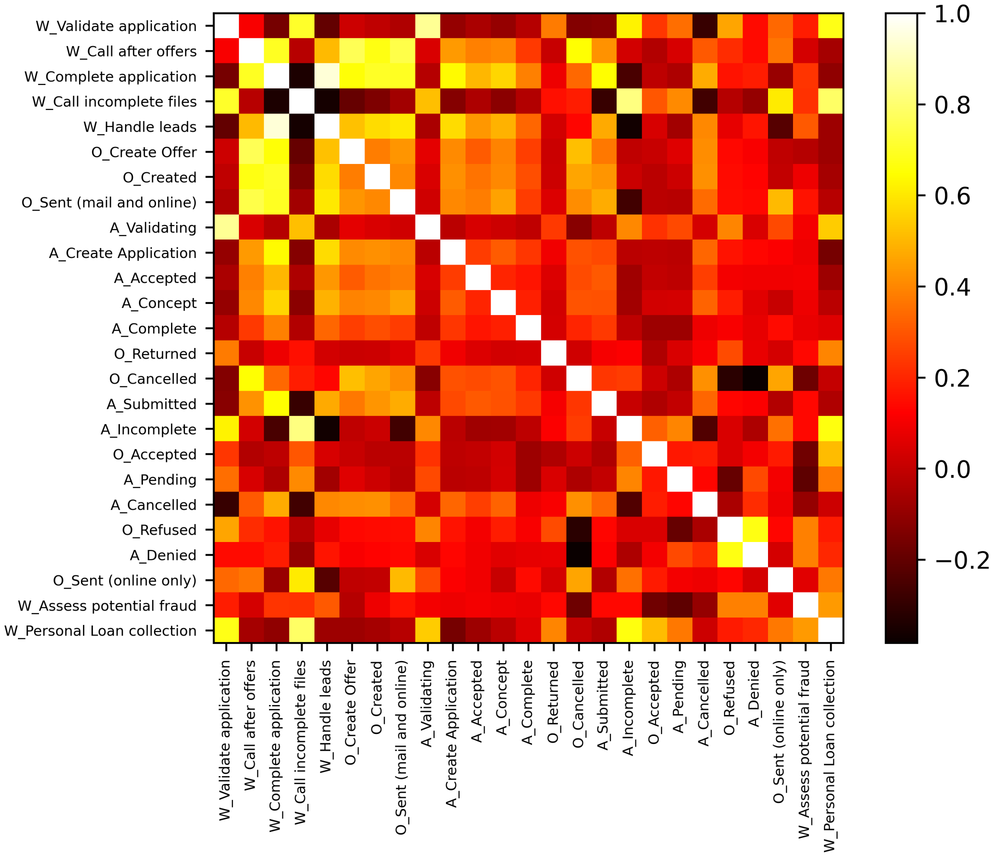


Figure 3.8: A heatmap for the cosine similarity matrix of activity pairs from the BPI 2017 event log. The corresponding activity names are on the x and y-axis and the color bar to the right indicates the similarity value for the color spectrum.

their windows largely overlap), leading to more similar vectors according to the Skip-Gram objective (consider "W_Handle leads" and "A_Submitted" or "A_Denied" and "O_Refused" for example).

In [43] it was shown, that Skip-Gram implicitly factorizes a word-context matrix M, whose cells are the pointwise mutual information PMI(w, c), into two smaller matices, word matrix W and context matrix C, such that we can reconstruct M, with $W \cdot C^T = M$ (approximately). We can view the PMI(w, c) as a measure of association strength of a target word w and context word c, as expressed in equation 3.1.

In the following, we will reuse some mathematical formulations of the study. The Skip-Gram model assumes a corpus of words $w \in V_W$ and their contexts $c \in V_C$, where V_W and V_C are the word and context vocabularies. In the equation, D denotes the collection of word-context pairs, $\#(w, c)$ is the number of times the pair (w, c) appears in D. The number of times w occurred in D overall is $\#(w) = \sum_{c' \in V_C} \#(w, c')$ and for c it is $\#(c) = \sum_{w' \in V_W} \#(w', c)$.

$$\text{PMI}(w, c) = \log \frac{\#(w, c) \cdot |D|}{\#(w) \cdot \#(c)} \tag{3.2}$$

So the PMI(w, c) is essentially the log ratio of the joint probability $\#(w, c)$, the frequency w and c occurring together, to the product of the marginal probabilities $\#(w)$ and $\#(c)$, the frequency of w and c occurring independently (the word-context collection size $|D|$ is constant for all pairs). In summary, the objective of Skip-Gram is to factorize the PMI matrix of word-context pairs M in a $V_w \times d$ matrix W and a $V_c \times d$ matrix C, such that the reconstruction loss of $W \cdot C^T = M$ is minimal, assuming that dimensionality d of W and C is not high enough to allow for perfect reconstruction (remember we typically choose $d << V_w$). Typically, in NLP, the rows of the matrix W are the final word embeddings, while matrix C is disregarded.

Let us consider finding ii) again, that activities that are on paths with higher relative frequencies tend to cluster together. We define T_{wc} as the subset of all traces T, where activity w and c are present. Then we can define the number of joint occurrences $\#(w, c)$ of an activity w as target and activity c as context with window size l and their distance in the specific trace $d(t, w, c)$ as follows:

$$\#(w, c) = \sum_{t \in T_{wc}} X(t, w, c), X(t, w, c) = \begin{cases} 1, & \text{if } d(t, w, c) \leq l \\ 0, & \text{otherwise} \end{cases} \tag{3.3}$$

This equation can also be applied to calculate $\#(w)$ and $\#(c)$ over all traces T, as defined previously. From this formulation of $\#(w, c)$, we can follow that, when two activities w, c are connected by one or more paths of high relative frequency (relative to overall independent occurrences of $\#(w)$ and $\#(c)$ in all traces T), where their distance is shorter than window size l, they also have a high PMI, thus a higher level of association.

This also explains the observation that activity vectors from the “starting” branch (activities “A_Create Application” to “O_Sent (mail and online)” in figure 3.7 and 3.8) and activities of the left branch that follows after “O_Sent (mail and online)” (figure 3.9), are more similar than activities from the “starting” branch are to activities that follow “O_Sent (online only) from the right branch (figure 3.10). Ultimately, they have higher relative path frequencies, especially for the two activities that make the left and right branch (“O_Sent (mail and online)” with path frequency 35,812 to “O_Sent (online only)” with path frequency of only 1,895).

With these two findings, we can conclude that the resulting activity embeddings from the Skip-Gram model indeed capture control flow information about the underlying process. We learn embeddings, that encode information about i) the distances between two activities in traces (thus their possible relative positions in the underlying process model) and ii) the relative frequency of their co-occurrences in shared paths, as their strength of association.

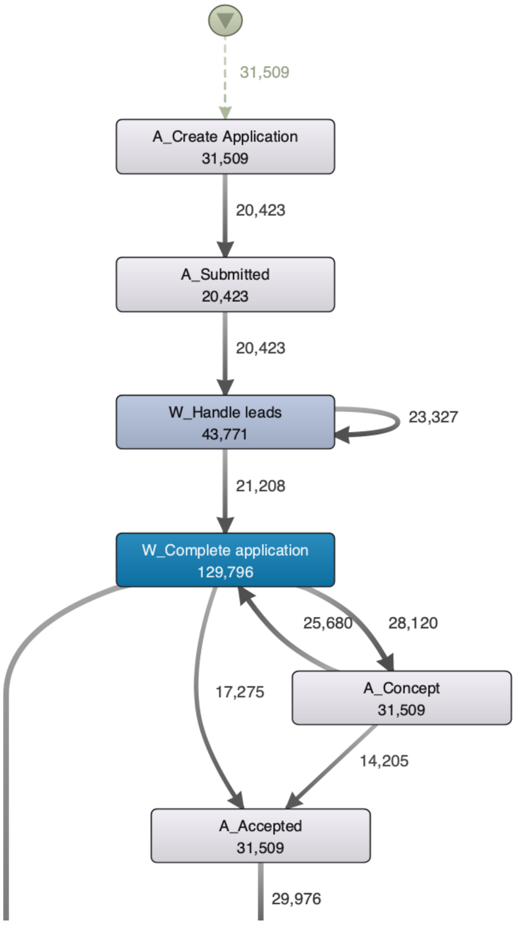


Figure 3.9: The discovered process map from Disco of the BPI Challenge 2017 real world event log of a loan application process of a Dutch Financial Institution (Part 1)

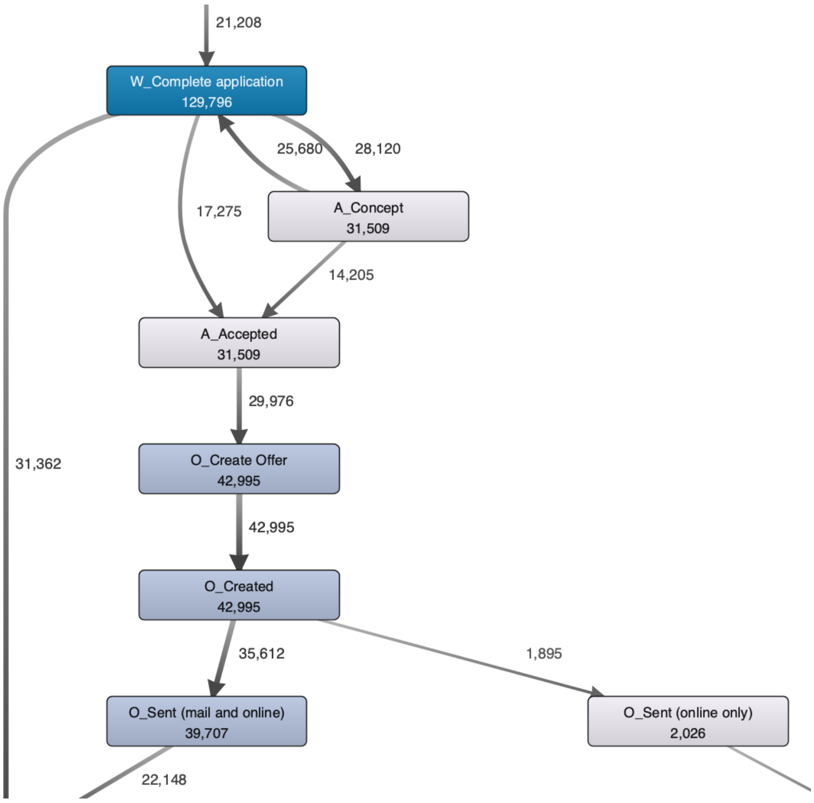


Figure 3.10: The discovered process map from Disco of the BPI Challenge 2017 real world event log of a loan application process of a Dutch Financial Institution (Part 2)

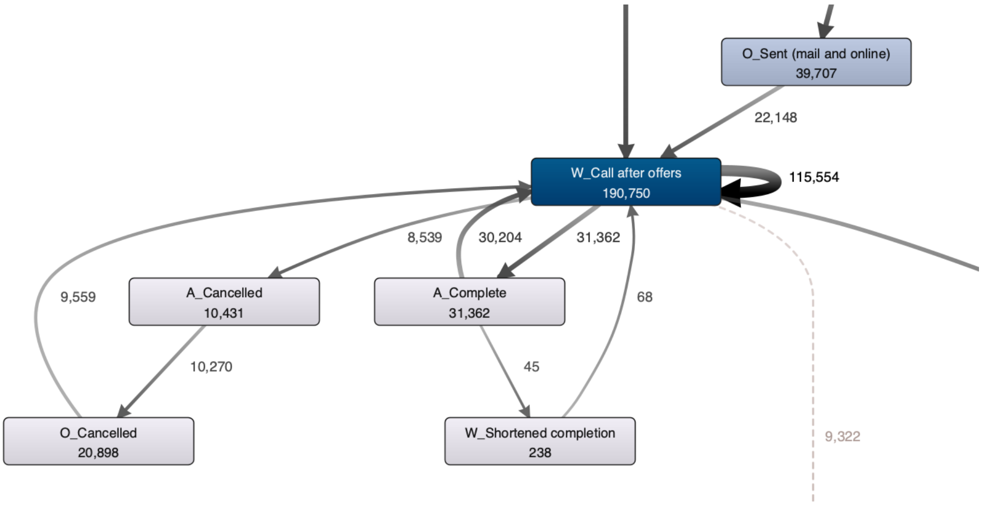


Figure 3.11: The discovered process map from Disco of the BPI Challenge 2017 real world event log of a loan application process of a Dutch Financial Institution (Part 3)

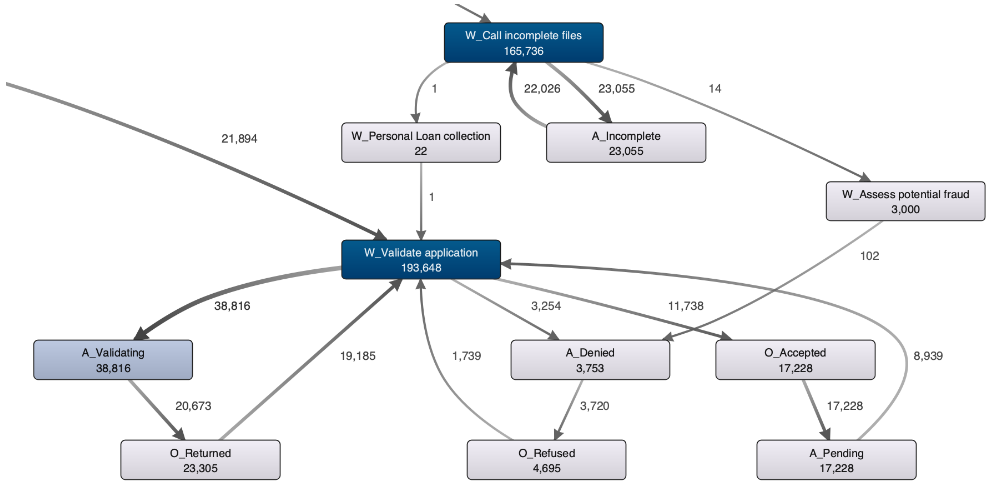


Figure 3.12: The discovered process map from Disco of the BPI Challenge 2017 real world event log of a loan application process of a Dutch Financial Institution (Part 4)

4 Evaluation of Activity Embeddings in Next Activity Prediction

In this chapter, we will first present the setup of our experiments on next activity prediction (section 4.1). We will once again discuss the reasons why we think that activity embeddings might be superior to conventional one-hot encoding as neural network input, and formulate the null hypothesis for subsequent statistical tests (section 4.2). Following that, we present the results of our experiments and the outcome of statistical tests for pairs of embedding and one-hot models (section 4.3). Surprisingly, the statistical tests show, that there is no significant difference when utilizing embedded activities compared to one-hot encoded activities as input for our neural network models for next activity prediction. Ultimately, we come up with potential reasons to explain these results, and show limitations of our conducted experiments (section 4.4). The source code for the experiment setup can be found on this GitHub repository[1] and may be reused for further experiments by other researchers.

4.1 Experimental Setup

The subsequent section will give more details on our experimental setup for evaluation performance of embedded and one-hot encoded activities on next activity prediction. We introduce the activity embedding and encoding techniques that we used in 4.1.1 and give a short overview of the two employed neural network architectures and their implementation in 4.1.2. In subsection 4.1.3 we present the five datasets for our experiments and in the last subsection 4.1.4, we summarize our protocols for input preparation, training and testing.

4.1.1 Embedding and Encoding Techniques

For evaluation of performance, we selected the CBOW and Skip-Gram neural network architectures from [1] as embedding techniques. Both techniques are implemented

[1] `https://github.com/Christian-Gebhardt/act2vec_evaluation`

in the Word2Vec model from the GenSim library [74], which we used to train activity embeddings for each dataset. Essentially, the shallow feed forward neural networks models are trained on two independent tasks: predicting a target word, given surrounding context words (for the CBOW model) and predicting the surrounding words given a target word (for the Skip-Gram model). The architecture of the models and their procedure for learning high quality word vectors was already covered in more detail in the systematic literature review of word embeddings in section 2.2.1. As baseline for comparison, we also generated random embeddings by using the exact embedding dimension as we used for the other two embedding models, but filled the vectors with uniform random values in the interval $[-1, 1]$.

The mentioned embedding techniques are compared with one-hot Encoding when assessing the performance of each input encoding technique on next activity prediction. One-Hot Encoding is one standard technique for encoding nominal features like activities as input for a neural network setup, and is also commonly used as encoding technique in other studies on next activity prediction (see the review of [24]). In this method, each activity is assigned an index and is represented by a binary vector of length n, with n being the number of unique activities in the event log. To represent the activity with index i, all elements are zero in the vector belonging to the activity, except for one element at the position i.

4.1.2 Neural Network Architectures

The learned activity embeddings (or one-hot encoded activities) will be the input for neural network architectures, that are trained and evaluated on the task of next activity prediction. A typical neural network architecture is composed by an input layer, an output layer and an arbitrary number of hidden layers between them (see figure 4.1). Each layer contains an arbitrary number of neurons, which calculate an activation function, that has the summation of the output of the activation function of interconnected nodes, together with their respective connection weights and biases as input. First, the input layer passes the input features multiplied by their weight of the connection to the subsequent layer. If there are any hidden layers, they take the incoming feature values of their connections, calculate the activation function and pass the output to their connected neurons. This step is repeated until the output layer is reached. The neurons of the output layer again take the input of connected neurons of preceding layer(s) and calculate the activation function, which will be the final output of the neural network.

When training, neural networks are provided with training samples, where each sample consists of a set of input features and a known target value. By providing the neural network with examples and the expected output, it optimizes its weights and biases for each connection and therefore changes the output of the activation functions of each neuron, to ultimately minimize the error between the generated output value and the actual target value. The learning of optimal weights is realized by error backpropagation, where each weight is adjusted by an update term, that is based on the gradient of the error function. For more information on neural networks and their learning procedure, we recommend reading the relevant chapters of the book [85].

Feed Forward and Long Short-Term Memory

For the experiments, we used two types of neural network architectures, Feed-Forward Neural Networks (FNN) and Long Short-Term Neural Networks (LSTM). The FNN architecture is characterized by its forward flow of information between the layers, this means at each step the output of the activation function of the neurons only flows to the subsequent layer, not backwards. The network is therefore an acyclic graph, where information is passed first to the input layer, then optionally from one hidden to the next, and finally to the output layer. An example architecture of a FNN is depicted in figure 4.1.

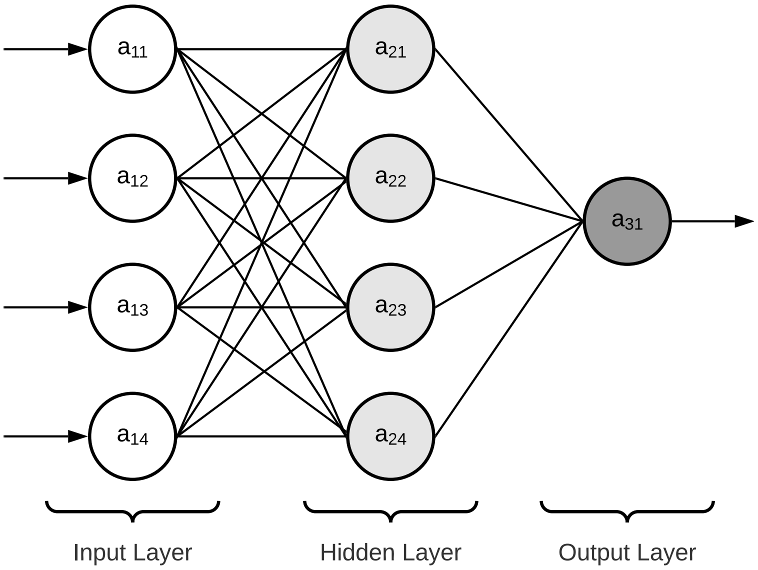


Figure 4.1: The architecture of a Feed Forward Neural Network, with four neurons in the input layer, four neurons in the subsequent hidden layer and one neuron in the final output layer. Each layer is fully connected with its previous layer.

One limitation of FNNs is, that they only take current input X_t and no previous input $X_{0:t-1}$ of the neural network into consideration for their calculation of the current output [86, 87]. Nonetheless, in time-dependent tasks like next-activity-prediction, having information about distant past activities and understanding their influence on the current output may be vital.

Long Short-Term Memory Networks, that were first introduced in [88], overcome this issue by “remembering” earlier input in a cell state. LSTMs are a form of recurrent neural network (RNN) architecture, but their advantage over basic recurrent neural networks, is that they do not suffer from vanishing or exploding gradient updates, which can cause problems when learning long-range dependencies [87]. This is achieved by integrating a mechanism for long-term memory with constant error flow [88]. In Figure 4.2, the structure of an LSTM cell is illustrated, depicting the interactions among the input gate, forget gate, cell state, and output gate. In contrast to FNNs, where one unit only consist of a single neuron, the LSTM cell uses multiple neurons.

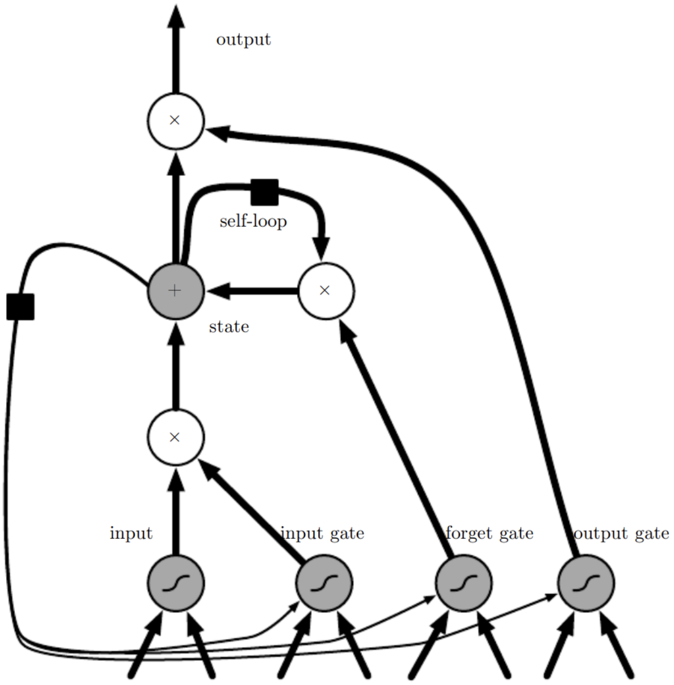


Figure 4.2: The structure of a Long Short-Term Memory cell from [85, p. 411]. The cell state is effected by the activation of the input gate (new information) and the activation of the forget gate through a self-loop with an earlier cell state (long-term memory). The output gate controls the flow of outgoing information, that was computed as final output by the LSTM cell (short-term memory). Optionally, the cell state can also be added as input for the activation functions of the three gateways.

The cell state is regulated by the input, an input gate and a forget gate. The input is calculated with a regular neuron, but the relevance of the input and the influence of this new information on the cell state is determined by the sigmoid activation of the input gate. The state has an inner self loop delayed by one time-step, therefore the previous state can alter the current state (the state *unrolls* in time). This acts as the long-term memory of the cell, where the weight of the memory on current computation is calculated by the sigmoid activation of the forget gate.

Finally, the output gate controls the resulting output of the current LSTM cell state, which is the input for subsequent cells (short-term memory). Note, that earlier cell state can also be added as input for the three gating units. For a more formal and detailed description of RNNs, their gradient problem and LSTMs, we again refer to the excellent book [85].

Implementation

The FNN and LSTM networks were implemented with the Python Machine Learning Library *TensorFlow* [89]. To have better control over the architectures and to be able to modify them to our needs and the task at hand, we chose to implement them on our own, instead of using preexisting ones from other literature. Also, by using our own, rather simple neural network architectures instead of a specific or more complex architecture, we reduce the risk that model performance is predominantly influenced by architectural choices, instead of input encoding, which is our focus of research.

There are two variants for both networks, a variant for one-hot-encoded activities and a variant for embedded activities, since embeddings need an additional embedding layer. The architecture of the FNN, that uses embedded activities, has one input layer, an embedding, two dense (fully connected) hidden layers and one softmax output layer. We opted for two hidden layers, since using only one hidden layer might lead to poor generalization of the network or unnecessary high complexity (number of neurons) to reach the desired performance [85, p. 198], [90].

The input layer receives k embedding indices of activities, where k denotes the window size (number of activities preceding to the activity, that we want to predict) and passes them to the embedding layer. The embedding layer translates the embedding indices into their corresponding embedding vectors. These embedding vectors are then passed as input to the first hidden layer. After that, each neuron in the hidden layers, computes its activation function and passes the output to the input of the subsequent layer. The units in both dense layers use the Rectified Linear Unit (ReLU) activation function from equation 5.1, with input z from the previous layer as argument.

$$\text{ReLU}(z) = \max(0, z) \tag{4.1}$$

Ultimately, the output layer is a softmax layer with n neurons, where n is the number of unique activities in the event log. Here, the $i - th$ neuron computes the softmax activation function to obtain the probability for the $i-th$ activity, as seen in equation 5.2. Naturally, the network will select the activity with the highest probability of the n activities as its final prediction.

$$\sigma(z)_i = \frac{e^{z_i}}{\sum_{j=1}^{n} e^{z_j}} \tag{4.2}$$

The LSTM model follows the same structure as the FNN in principle, but the two dense hidden layers are replaced by two LSTM layers with LSTM cells as units. Stacking LSTM layers has shown great performance on several tasks, including next activity prediction [91] (although it is not completely theoretically understood why [92]). In contrast to the dense FNN hidden layers, the LSTM layers are not densely connected. Instead, the cells have a one-to-one connection, so each cell c_i1 in the first layer is connected to one subsequent cell c_i2 in the second layer.

The variants with one-hot encoded activities as neural network input are fundamentally similar, but do not use an additional embedding layer. Instead, they take the one-hot-encoded activities directly as input. For the experiments, we build the FNN with 32 units (single neurons) and the LSTM with 32 units (LSTM cells with multiple neurons) in each of the two hidden layers. So there are 64 hidden units in total for both neural network architectures.

4.1.3 Datasets

The following section presents the datasets of the conducted experiments. The choice of the datasets was based on the systematic literature review on deep learning methods for process prediction from [24]. All five datasets mentioned here have been previously employed in scientific articles related to process prediction, specifically in the area of next activity prediction, and are freely accessible. Thus, it is easier to relate the experiments to other studies within this research area.

In the first experiment, the objective was to find a sufficient embedding dimension as parameter for the subsequent main experiment, for which we used the *Helpdesk* and *Hospital* dataset for model training and testing. As we can see in table 4.1, the *Helpdesk* [93] and the *Hospital* [94] dataset are quite different in event log statistics, including average trace length and the number of unique activities of the event log. Consequently, we can evaluate the embedding dimension on i) large and small context size (trace length) and ii) large and small vocabulary size (unique activities).

For the final evaluation of the impact of the different encoding techniques on next activity prediction performance, we used the three datasets from table 4.2. If we look at the number of traces and average trace length, the *BPI Challenge 2012* [95], the *BPI Challenge 2017* [80] and the *Sepsis* [96] dataset deviate vastly from each

Dataset Summary	Helpdesk	Hospital
Num. traces	4580	1143
Avg. trace length	4.66	131.49
STD trace length	1.18	202.53
Min. trace length	2	1
Max. trace length	15	1814
Num. unique activities	14	624
Num. total events	21348	150291

Table 4.1: Summary of the Helpdesk and Hospital datasets including the number of traces, average trace length, standard deviation of trace length, minimum and maximum trace length, number of unique activities, and total events.

Dataset Summary	BPI 2012	BPI 2017	Sepsis
Num. traces	13087	31509	1050
Avg. trace length	20.04	38.16	14.49
STD trace length	19.94	16.72	11.47
Min. trace length	3	10	3
Max. trace length	175	180	185
Num. unique activities	24	26	16
Num. total events	262200	1202267	15214

Table 4.2: Summary of the Business Process Intelligence (BPI) Challenge 2012, the BPI Challenge 2017, and the Sepsis dataset including the number of traces, average trace length, standard deviation of trace length, minimum and maximum trace length, number of unique activities, and total events.

other. As a consequence, the over all number of samples available for learning varies among the datasets, allowing us to classify them as small (Sepsis), medium (BPI 2012) and large (BPI 2017) datasets. By incorporating datasets of varying sizes, we are trying to ensure a comprehensive coverage of test cases in our evaluation.

4.1.4 Training and Testing Details

In this section, we describe our input preparation, training and testing processes in more detail. Our input preparation transformed event log data in the XES format to activity sequences (individual traces) that can be fed as input for our neural network models. The training details include training setup, neural network optimizer and hyperparameter choices, and the hardware we utilized during training. Lastly, we summarize our cross validation protocol for model evaluation and the statistical method that we chose to test our hypotheses.

Input Preparation

Prior to training the neural networks, the data needs to be transformed into a format suitable for neural network input. The event logs, that we used as datasets for training and testing, came initially in the *XES* format [97]. We read in the XES files with functionality from the python process mining library *PM4Py* [98] and transformed it into a data frame, to make rows (events) and feature columns easily accessible. Subsequently, we converted the activities from the events to traces, where one trace represents all the activities of the corresponding case in chronological order. Only the activity name was used as feature to represent the activity, no further features, such as timestamp, resource, etc. were regarded.

After this step, the traces were padded at the start with $k-1$ dummy activities, where k is the window size. This step is necessary, so that the FNN can start predicting with activity prefix greater or equal to one as input. Otherwise, it would only be able to start predicting at activity a_{k+1}, since it needs at least k input activities as prefix. This is artificially avoided by giving the neural net $k-(i-1)$ dummy inputs (together with the $(i-1)$ real prefix activities) when predicting the activity a_i in the trace with insufficient prefix length. Now, to prepare the traces as input for the two neural network variants, the activities are either one-hot encoded or embedded with the random method or the CBOW and Skip-Gram techniques that were already mentioned in section 4.1.1.

Training

The following training setup was used in all conducted experiments. The window size was set to 10, which gives the network the 10 preceding activities $a_{i-10}, a_{i-9}, ..., a_{i-1}$ to the target activity a_i as input for prediction. For the learning of optimal weights, the *Adam* Gradient Descent Optimizer from [99] was used, which takes advantage of an adaptive learning rate. The batch size was set to 128, and the early stopping method was used to dynamically determine the number of epochs needed for the learning to converge, as recommended in [100]. This takes the burden of experimentally finding a good number of epochs. Also, it avoids overfitting, by halting, as soon as the performance stagnates on validation data [100]. Here, 5 epochs were used as "patience" parameter for early stopping. Consequently, the network will stop training and restore previous optimal weights, when the prediction accuracy is not improving on validation data for 5 consecutive training epochs.

The experiments were performed on an NVIDIA A40 GPU with 48 GB GDDR6 memory and 798 GB/s memory bandwidth. We utilized the named GPU in the TensorFlow GPU version with CUDA [101] instead of the CPU, since GPUs allow for fast and parallel matrix multiplication, which is a fundamental operation in neural network learning. Additionally, they do not suffer from memory buffer bottlenecks, when storing and updating parameters through gradient descent, due to their high memory bandwidth compared to CPUs [85, pp. 444-446]. Consequently, this results in faster training times.

Testing

When comparing different machine learning models, we want to perform statistical tests for our hypotheses. Following the recommendation for hypothesis testing from [102], we chose 5×2-fold cross validation with a modified paired Student's t-test, to test if there is a significant difference in performance between the two models.

For one iteration in the 5×2-fold cross validation, the samples from the dataset are randomly split into one training set, containing 50% of the samples and one test set, containing the other 50% of the samples. The model is then trained once on the training set and evaluated once on the test set. Afterward, the training and test set are swapped, for another training and test cycle. By dividing in only 2-folds and swapping them, we ensure that no sample will be used twice for training (disjoint training sets), which would otherwise violate the assumption of sample independence required for the paired t-test and ultimately leads to higher Type I errors [102]. Using the standard 10-fold cross validation, for example, would break this assumption, because training samples are shared among splits. The cross validation procedure is repeated five times, leading to ten independent performance measurements in total. To ensure fair comparison between models, the random splits are the same for all models.

After we take the ten independent measurements, we can apply the modified paired t-test for 5×2 cross validation proposed in [102] to calculate the t-statistic and p-value. If the p-value is smaller than the chosen significance level α, we reject the null hypothesis h_0 and accept that there is a significant difference in the two models. We used the common choice $\alpha = 0.05$ for the significance level in our statistical tests.

The classification metrics that are measured in evaluation include accuracy (equation 4.3), macro-average precision (equation 4.4), macro-average recall (equation 4.5) and top-3 accuracy, which should provide a broad overview of classifier performance. In the equations we denote true positives with *tp*, true negatives with *tn*, false positives with *tp* and false negatives with *fn*, and the number of classes as *l*. Macro-averaging takes the precision and recall per class and averages the scores, which results in equal class weights.

The accuracy, precision, and recall metrics are standard metrics, commonly used to assess performance in classification tasks. We chose the top-3 accuracy in addition, to give a more fine-grained picture of classifier performance. The metric evaluates how frequently the correct class (activity) is within the top three predicted classes based on their probabilities. We chose n=3, because for all event logs we considered the number of unique activities is significantly larger than 3. But we should note that for event logs with few next activity options (e.g., gateways with less than or equal 3 options), the top 3 choices might be obvious. We did not perform statistical tests for all the named metrics. Instead, we calculated the (macro-average) F-score from equation 4.6.

$$\text{Accuracy} = \frac{tp + tn}{tp + fn + fp + tn} \tag{4.3}$$

$$\text{Precision (macro)} = \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fp_i}}{l} \tag{4.4}$$

$$\text{Recall (macro)} = \frac{\sum_{i=1}^{l} \frac{tp_i}{tp_i + fn_i}}{l} \tag{4.5}$$

$$\text{F-score} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{4.6}$$

The (macro-average) F-score has the advantage, that it combines precision and recall in one metric. By taking the macro-averages of the two metrics, this prevents undervaluing rare classes in imbalanced datasets, where unequal proportions of target classes are present. This ensures a fair evaluation, in contrast to the simple accuracy metric, which would favor target classes with higher proportions (e.g., higher frequent activities in our case) [103, 104]. We additionally measured the training time for each model and the number of required epochs, before the early stopping patience threshold was reached (see training details).

4.2 Hypotheses

In this section, we will discuss important points of consideration from our research on embedding techniques and why activity embeddings might be superior to one-hot encoded activities, when assessing their performance next activity prediction. Also, we will formulate the null hypotheses h_0 and alternative hypotheses h_1 in context of our statistical tests on the experiment results of the performance evaluation. As to why using embedding techniques might lead to an increase in performance in next activity prediction, we identified two major factors:

1. **F1**: Reduction of dimensionality
2. **F2**: Inclusion of semantic information

Dimensionality reduction (F1) in our concrete case means, that compared to one-hot encoding, where the vector size is equal to the number of unique activities in the event log, embedding techniques offer the chance to only use a noticeably smaller vector size. A smaller input dimension, also will lower the number of trainable parameters for the network, which reduces memory requirements and might lead to faster training times and computational efficiency. Most importantly, embedding techniques learn to represent the words (or in our case activities) in a low-dimensional vector space, while keeping the most important factors of variation from the co-occurrence matrix (see also [43]).

These low-dimensional dense representations might be superior in performance to high-dimensional sparse representations, since this reduces the risk of the “curse of dimensionality” phenomena, which states that for effective density estimation, the required number of needed training samples increases exponentially with increasing dimensionality [105, 106]. This applies to machine learning models in the sense, that with high-dimensional input, the data is more sparse, making it more difficult to generalize the model accurately. Also, learning low-dimensional input representations will lead to a model with fewer parameters, decreasing the risk of overfitting the training data[85, p. 114], [107].

When learning word embeddings in the NLP domain, the resulting vectors provably contain semantic information, as we saw earlier in section 1.2.1. We hypothesize that similar inclusion of semantic information (F2) might also occur when these embedding techniques are applied to activities in the process mining domain. However, the concept of semantics might differ here. Fundamentally, word embeddings are learned by exploiting word co-occurrence statistics, where similar context results in similar vectors. For words, the vectors typically allow for comparison of word meaning (e.g., the synonyms “hot” and “warm” might have very similar vectors), since the distributional hypothesis implies that word co-occurrence statistics can be taken as measure for their relatedness in word meaning.

If we apply this measure to activities, the “meaning” or semantics in the learned vectors might represent their relatedness to positional information or control flow context (see our previous chapter on interpretability). With this additional encoded information in the input, we could see an increase in model performance, since it could help the neural network to better understand process control flow. Again, for one-hot-encoding all activities are equidistant and therefore no information about their semantic relatedness is given.

In our first experiment, we first attempt to find a sufficient embedding dimension. In our second experiment, we employ this embedding dimension for the learned activity vectors and compare the performance on the F-score metrics of model variants, which use activity embeddings as input, against model variants, which use one-hot encoded activities as input. For significance testing, we formulate the null hypothesis H_0 and the alternative hypothesis H_1 for comparing two model variants as follows (see section 4.1.4 for more details on the procedure):

- H_0: The F-scores of the two compared models are equal
- H_1: The F-score of one model is significantly different from the F-score of the other model

We chose $\alpha = 0.05$ as significance level for the modified paired Student’s t-test, as described earlier. When $p \leq \alpha$ we reject H_0 and conclude that there is probably a difference in F-score performance between the two models as stated by H_1. Therefore, the model with the higher F-score is superior to the other model with the lower F-score. However, if $p > \alpha$, we fail to reject H_0 and conclude that there is no significant difference in F-score performance between the two models.

4.3 Results

In this section, the results of the two experiments are presented. The goal of the first experiment was to find an appropriate embedding dimension for the activity vectors, with respect to next activity prediction performance. This embedding dimension will then be used for the activity vectors in the final experiment, where the effect of embeddings on next activity prediction performance is compared to one-hot encoding.

In the first experiment, the embedding dimensions 8, 32, 128 and 256 were evaluated with a 5×2-fold cross validation for the embedding techniques CBOW and Skip-Gram on two datasets. We added random embeddings as baseline, generated with the procedure we described earlier. The helpdesk dataset has a relatively low number of activities and a small average trace length, while the Hospital dataset has many activities and the average trace length is high (see section 4.1.3 on datasets). As a result, the performance was compared for two contrary datasets to give a more general recommendation for the embedding dimension. The results of the comparison on the Helpdesk and Hospital dataset can be found in table 4.3 and table 4.4, respectively. The best performance for each embedding dimension is highlighted in bold and underlined.

The average metric values, show minimal differences (rarely >0.005) for varied embedding dimensions (8, 32, 128, 256) and for different embedding techniques (Skip-Gram, CBOW, Random) or model variants (FNN and LSTM). Surprisingly, the random embeddings do not seem to be inferior to the activity vectors learned by the Skip-Gram or CBOW embedding techniques. As there were no notable differences in model performance, we examined the training time and the number of training epochs needed to meet the early stopping criteria (refer to table 4.9 and table 4.11 at the end of the section).

Moreover, against our expectations, the average training time (in seconds) and the number of required epochs for early stopping did not show an increase with higher embedding dimensions. Instead, there was a significant decrease in training time observed for higher embedding dimensions, when learning the FNN and LSTM models. We also observe higher training times for LSTMs in general, when compared to FNNs. Possible reasons for these differences in training time, are that LSTMs hard to parallelize on the GPU and are slowed down by the temporal dependency between units (backpropagation through time) [108, 109].

We further experimented with even higher dimension (512, 1028) to see when there is no more decrease in training time. The results can be found at the end of the section in table 4.10 and 4.12, respectively. When we look at the training times for dimension 512 and 1024 for the Helpdesk and the Hospital dataset, we can observe, that the training times dropped again for FNNs on the Helpdesk and the Hospital dataset, while they increased for the LSTMs. Therefore, we chose 256 as embedding dimension for our final experiment, since the experiments showed very similar results for all the embedding dimensions that we tried as parameter, but up to the dimension 256, the training times for FNNs and LSTMs both decreased.

Technique (Dim=8)	Accuracy	Precision	Recall	Top-3 Accuracy
RANDOM (FNN_WV)	**0.795**	0.800	0.789	0.979
CBOW (FNN_WV)	0.794	0.802	0.783	0.981
SGNS (FNN_WV)	0.790	0.798	0.780	0.979
RANDOM (LSTM_WV)	0.790	0.797	0.779	0.980
CBOW (LSTM_WV)	0.789	0.798	0.773	0.979
SGNS (LSTM_WV)	0.788	0.798	0.767	0.978
Technique (Dim=32)				
RANDOM (FNN_WV)	0.794	0.806	0.776	0.980
CBOW (FNN_WV)	0.794	0.802	0.783	0.981
SGNS (FNN_WV)	**0.795**	0.802	0.784	0.980
RANDOM (LSTM_WV)	0.790	0.798	0.775	0.980
CBOW (LSTM_WV)	0.788	0.804	0.659	0.957
SGNS (LSTM_WV)	0.788	0.798	0.771	0.978
Technique (Dim=128)				
RANDOM (FNN_WV)	**0.795**	0.802	0.785	**0.981**
CBOW (FNN_WV)	0.795	0.803	0.782	0.980
SGNS (FNN_WV)	0.795	0.802	0.783	0.980
RANDOM (LSTM_WV)	0.793	0.799	0.784	0.979
CBOW (LSTM_WV)	0.788	0.803	0.764	0.960
SGNS (LSTM_WV)	0.790	0.806	0.761	0.961
Technique (Dim=256)				
RANDOM (FNN_WV)	0.795	0.802	0.786	0.980
CBOW (FNN_WV)	0.777	0.863	0.827	0.963
SGNS (FNN_WV)	0.795	0.803	0.782	0.980
RANDOM (LSTM_WV)	**0.795**	0.802	0.786	**0.981**
CBOW (LSTM_WV)	0.774	0.859	0.823	0.960
SGNS (LSTM_WV)	0.790	0.723	0.690	0.961

Table 4.3: Comparison of average performance on the 5×2-fold cross validation of different embedding techniques for embedding dimensions 8, 32, 128 and 256 on the **Helpdesk** dataset. The highest accuracy scores of the models are bold and underlined for each embedding dimension (top-3 accuracy, if accuracy is equal).

Technique (Dim=8)	Accuracy	Precision	Recall	Top-3 Accuracy
RANDOM (FNN_WV)	0.653	0.769	0.554	0.847
CBOW (FNN_WV)	0.643	0.763	0.539	0.844
SGNS (FNN_WV)	0.650	0.766	0.547	0.849
RANDOM (LSTM_WV)	0.669	0.787	0.568	0.854
CBOW (LSTM_WV)	**0.672**	0.784	0.576	0.860
SGNS (LSTM_WV)	0.671	0.783	0.575	0.858
Technique (Dim=32)				
RANDOM (FNN_WV)	0.664	0.766	0.579	0.856
CBOW (FNN_WV)	0.659	0.769	0.568	0.855
SGNS (FNN_WV)	0.667	0.768	0.583	0.860
RANDOM (LSTM_WV)	**0.678**	0.787	0.586	0.860
CBOW (LSTM_WV)	0.677	0.784	0.585	0.862
SGNS (LSTM_WV)	0.675	0.785	0.581	0.859
Technique (Dim=128)				
RANDOM (FNN_WV)	0.665	0.764	0.584	0.856
CBOW (FNN_WV)	0.666	0.76	0.59	0.857
SGNS (FNN_WV)	0.669	0.767	0.587	0.86
RANDOM (LSTM_WV)	**0.680**	0.785	0.591	0.862
CBOW (LSTM_WV)	0.677	0.783	0.586	0.862
SGNS (LSTM_WV)	0.677	0.782	0.587	0.861
Technique (Dim=256)				
RANDOM (FNN_WV)	0.664	0.761	0.587	0.855
CBOW (FNN_WV)	0.664	0.761	0.586	0.856
SGNS (FNN_WV)	0.669	0.765	0.589	0.859
RANDOM (LSTM_WV)	**0.678**	0.786	0.584	0.86
CBOW (LSTM_WV)	0.676	0.781	0.585	0.862
SGNS (LSTM_WV)	0.677	0.784	0.585	0.861

Table 4.4: Comparison of average performance on the 5×2-fold cross-validation of different embedding techniques for embedding dimensions 8, 32, 128 and 256 on the **Hospital** dataset. The highest accuracy scores of the models are bold and underlined for each embedding dimension (top-3 accuracy, if accuracy is equal).

For our final evaluation, we compared the embeddings generated by the techniques CBOW and Skip-Gram and the random embeddings (all using embedding dimension 256) with one-hot encoded activities as model input for the next activity prediction task on three datasets (BPI Challenge 2012, BPI Challenge 2017 and Sepsis), using the two model variants (FNN, LSTM) as before. The results are presented in tables 4.6, 4.7 and 4.8 at the end of the section.

Like in the last experiments, the performance on the different metrics is very similar for all techniques, this time including one-hot encoding. Only for the Sepsis dataset, the metrics show greater variance, but we may attribute this instability to the small size of the dataset. We employ the modified paired Student's t-test on the F-score performance for pairs of model variants (FNN and LSTM), which use one-hot-encoded input, vs. the model variants, which use either the CBOW or Skip-Gram (SG) embeddings as input. For fair comparison, we only employed the test for same model variants. Table 4.5 contains the 12 test results for the three datasets and for each combination.

Dataset	Model 1	Model 2	Mean F-Score 1	Mean F-Score 2	T-Stat	P-Value
BPI 2012	FNN_OH	FNN_WV / SG	0.852	0.85	0.052	0.96
BPI 2012	FNN_OH	FNN_WV / CBOW	0.852	0.853	-0.044	0.967
BPI 2012	LSTM_OH	LSTM_WV / SG	0.852	0.853	-0.441	0.677
BPI 2012	LSTM_OH	LSTM_WV / CBOW	0.852	0.853	-0.083	0.937
BPI 2017	FNN_OH	FNN_WV / SG	0.873	0.874	0.289	0.784
BPI 2017	FNN_OH	FNN_WV / CBOW	0.873	0.873	0.13	0.902
BPI 2017	LSTM_OH	LSTM_WV / SG	0.875	0.875	0.274	0.795
BPI 2017	LSTM_OH	LSTM_WV / CBOW	0.875	0.875	-0.146	0.89
SEPSIS	FNN_OH	FNN_WV / SG	0.593	0.591	-0.101	0.923
SEPSIS	FNN_OH	FNN_WV / CBOW	0.593	0.588	0.045	0.966
SEPSIS	LSTM_OH	LSTM_WV / SG	0.543	0.615	-0.002	0.999
SEPSIS	LSTM_OH	LSTM_WV / CBOW	0.543	0.605	0.009	0.993

Table 4.5: Results of the modified paired Student's t-test from [102] for the comparison of performance on next activity prediction when using one-hot encoding (OH) vs. the CBOW and Skip-Gram (SG) embedding techniques as input encoding method for the neural network models.

All 12 tests fail to reject the null hypothesis (the F-score of the two compared models are equal), since the p-values of all paired t-tests are greater $\alpha = 0.05$. The high p-values suggest, that there is no significant difference between F-score performance in next-activity prediction on any dataset or any model variant when using one-hot encoding as input for the model variant vs. any of the two employed embedding techniques, CBOW and Skip-Gram. Despite apparent F-score differences in the LSTM pairs for the Sepsis dataset, they are likely due to high variance and a small sample size. Thus, their calculated p-value remains high, indicating insufficient evidence to reject the null hypothesis.

Technique	Accuracy	Precision	Recall	F-Score	Top-3 Accuracy
OH (FNN_OH)	0.628	0.761	0.486	0.593	0.895
CBOW (FNN_WV)	0.619	0.736	0.489	0.588	0.887
SGNS (FNN_WV)	0.611	0.731	0.495	0.591	0.89
RANDOM (FNN_WV)	0.626	0.74	0.518	0.609	0.902
OH (LSTM_OH)	0.603	0.775	0.418	0.543	0.889
CBOW (LSTM_WV)	0.636	0.782	0.493	0.605	0.91
SGNS (LSTM_WV)	0.636	0.787	0.505	0.615	0.914
RANDOM (LSTM_WV)	0.64	0.784	0.516	0.622	0.914

Table 4.6: Comparison of average performance on the 5×2-fold cross-validation of the **Sepsis** dataset. Embedding techniques used a vector dimension of 256.

Technique	Accuracy	Precision	Recall	F-Score	Top-3 Accuracy
OH (FNN_OH)	0.848	0.906	0.804	0.852	0.98
CBOW (FNN_WV)	0.848	0.9	0.81	0.853	0.979
RANDOM (FNN_WV)	0.848	0.906	0.805	0.853	0.98
SGNS (FNN_WV)	0.847	0.893	0.812	0.851	0.979
OH (LSTM_OH)	0.848	0.906	0.804	0.852	0.979
CBOW (LSTM_WV)	0.848	0.907	0.805	0.853	0.98
RANDOM (LSTM_WV)	0.848	0.908	0.804	0.853	0.98
SGNS (LSTM_WV)	0.849	0.907	0.805	0.853	0.98

Table 4.7: Comparison of average performance on the 5×2-fold cross-validation for the **BPI Challenge 2012** dataset. Embedding techniques used a vector dimension of 256.

Technique	Accuracy	Precision	Recall	F-Score	Top-3 Accuracy
OH (FNN_OH)	0.874	0.878	0.869	0.873	0.991
CBOW (FNN_WV)	0.874	0.878	0.868	0.873	0.991
RANDOM (FNN_WV)	0.874	0.877	0.871	0.874	0.991
SGNS (FNN_WV)	0.874	0.877	0.871	0.874	0.991
OH (LSTM_OH)	0.875	0.878	0.872	0.875	0.992
CBOW (LSTM_WV)	0.875	0.878	0.872	0.875	0.992
RANDOM (LSTM_WV)	0.875	0.877	0.873	0.875	0.992
SGNS (LSTM_WV)	0.875	0.877	0.872	0.874	0.992

Table 4.8: Comparison of average performance on the 5×2-fold cross-validation for the **BPI Challenge 2017** dataset. Embedding techniques used a vector dimension of 256.

Technique (Dim=8)	Training Time	Training Epochs
RANDOM (FNN_WV)	4.176	17.9
CBOW (FNN_WV)	4.615	21.3
SGNS (FNN_WV)	**4.055**	18.0
RANDOM (LSTM_WV)	11.607	22.3
CBOW (LSTM_WV)	11.552	23.0
SGNS (LSTM_WV)	**10.704**	20.2
Technique (Dim=32)		
RANDOM (FNN_WV)	**3.825**	16.8
CBOW (FNN_WV)	3.991	17.8
SGNS (FNN_WV)	4.457	19.7
RANDOM (LSTM_WV)	9.224	16.3
CBOW (LSTM_WV)	**7.196**	11.7
SGNS (LSTM_WV)	9.385	16.7
Technique (Dim=128)		
RANDOM (FNN_WV)	**3.039**	12.1
CBOW (FNN_WV)	3.767	16.1
SGNS (FNN_WV)	3.197	13.4
RANDOM (LSTM_WV)	8.568	14.8
CBOW (LSTM_WV)	**6.865**	10.6
SGNS (LSTM_WV)	8.775	15.2
Technique (Dim=256)		
RANDOM (FNN_WV)	**2.817**	11.0
CBOW (FNN_WV)	3.379	14.7
SGNS (FNN_WV)	3.098	12.8
RANDOM (LSTM_WV)	10.393	19.3
CBOW (LSTM_WV)	8.243	13.9
SGNS (LSTM_WV)	**7.754**	12.8

Table 4.9: Comparison of the average training time in seconds and the number of epochs on the 5×2-fold cross-validation of different embedding techniques for embedding dimensions 8, 32, 128 and 256 on the **Helpdesk** dataset. The lowest training times are bold and underlined for each model type (FNN, LSTM) and each embedding dimension.

Technique (Dim=512)	Training Time	Training Epochs
RANDOM (FNN_WV)	**2.824**	11.8
CBOW (FNN_WV)	3.075	13.1
SGNS (FNN_WV)	3.218	14.1
RANDOM (LSTM_WV)	**8.751**	16.1
CBOW (LSTM_WV)	9.217	17.5
SGNS (LSTM_WV)	9.744	18.9
Technique (Dim=1024)		
RANDOM (FNN_WV)	2.762	12.0
CBOW (FNN_WV)	**2.437**	10.1
SGNS (FNN_WV)	2.92	12.5
RANDOM (LSTM_WV)	9.023	17.2
CBOW (LSTM_WV)	8.306	15.3
SGNS (LSTM_WV)	**8.281**	15.1

Table 4.10: Comparison of the average training time in seconds and the number of epochs on the 5×2-fold cross-validation of different embedding techniques for embedding dimensions 512 and 1024 on the **Helpdesk** dataset. The lowest training times are bold and underlined for each model type (FNN, LSTM) and each embedding dimension.

Technique (Dim=8)	Training Time	Training Epochs
RANDOM (FNN_WV)	63.62	36.2
CBOW (FNN_WV)	**51.45**	**28.9**
SGNS (FNN_WV)	59.89	33.5
RANDOM (LSTM_WV)	146.01	40.8
CBOW (LSTM_WV)	**135.70**	**39.1**
SGNS (LSTM_WV)	167.76	47.9
Technique (Dim=32)		
RANDOM (FNN_WV)	42.71	23.9
CBOW (FNN_WV)	**40.18**	**22.6**
SGNS (FNN_WV)	44.17	24.8
RANDOM (LSTM_WV)	**124.90**	**34.7**
CBOW (LSTM_WV)	132.11	37.3
SGNS (LSTM_WV)	133.64	37.7
Technique (Dim=128)		
RANDOM (FNN_WV)	**28.26**	**15.3**
CBOW (FNN_WV)	42.30	23.2
SGNS (FNN_WV)	32.52	17.8
RANDOM (LSTM_WV)	**106.16**	**29.7**
CBOW (LSTM_WV)	118.15	33.1
SGNS (LSTM_WV)	116.00	32.6
Technique (Dim=256)		
RANDOM (FNN_WV)	**24.70**	**13.2**
CBOW (FNN_WV)	36.02	19.8
SGNS (FNN_WV)	29.58	16.5
RANDOM (LSTM_WV)	**92.11**	**25.6**
CBOW (LSTM_WV)	111.26	31.4
SGNS (LSTM_WV)	108.95	30.3

Table 4.11: Comparison of the average training time in seconds and the number of epochs on the 5×2-fold cross validation of different embedding techniques for embedding dimensions 8, 32, 128 and 256 on the **Hospital** dataset. The lowest training times are bold and underlined for each model type (FNN, LSTM) and each embedding dimension.

Technique (Dim=512)	Training Time	Training Epochs
RANDOM (FNN_WV)	**<u>20.502</u>**	11.1
CBOW (FNN_WV)	31.709	17.6
SGNS (FNN_WV)	27.099	14.6
RANDOM (LSTM_WV)	**<u>94.408</u>**	26.3
CBOW (LSTM_WV)	125.897	35.1
SGNS (LSTM_WV)	107.858	30.0
Technique (Dim=1024)		
RANDOM (FNN_WV)	**<u>21.269</u>**	11.5
CBOW (FNN_WV)	29.57	16.2
SGNS (FNN_WV)	25.477	14.0
RANDOM (LSTM_WV)	**<u>99.509</u>**	27.6
CBOW (LSTM_WV)	129.507	36.6
SGNS (LSTM_WV)	126.563	35.4

Table 4.12: Comparison of the average training time in seconds and the number of epochs on the 5×2-fold cross-validation of different embedding techniques for embedding dimensions 512 and 1024 on the **Hospital** dataset. The lowest training times are bold and underlined for each model type (FNN, LSTM) and each embedding dimension.

4.4 Discussion

In this section, we will discuss the results that we obtained for the final experiment on the performance evaluation of input encoding techniques on the next event prediction task. We compared the activity embedding learned by CBOW and Skip-Gram embedding methods with one-hot encoded activities as neural network model input. After we performed a set of modified paired Student's t-tests on the results, as recommend in [102], we came to the conclusion, that there is no significant difference in F-score performance, when using either embeddings or one-hot encoding.

In the following, we will outline three potential reasons, that might explain these findings. Additionally, we will also show the limitations of our experiments in the latter parts of this section. Beginning with the underlying factors, that might contribute to our results, we looked at the performance gains achieved by embedding methods in the NLP domain and tried to identify the differences to the process mining domain and the next-activity-prediction task.

One obvious factor, which we think might also be the most severe, is that we cannot leverage additional information from large external data sources like in the NLP domain. When pre-training word embeddings, there are typically external data sources of enormous size (e.g., entire Wikipedia dumps with billions of words [3, 42]), before these embeddings are then reused on a downstream task. Consequently, these embeddings can be refined on huge training data sets with new information at very low computational cost. This is demonstrated by the shallow neural network models of [1, 2] that learn high quality embeddings from a 1.6 billion dataset in less than a day.

Since words are generally used similarly in language[2] and the vocabulary needed for a task is often times a subset of the learned vocabulary (one vocabulary can include the most common words of an entire language), their embeddings can be shared among tasks. This is not possible for activities in process mining, since they lack the universality, in contrast to words in language. Generally, different event logs do not share the same activities. Thus, we simply cannot leverage information from external data sources like in the NLP domain to pretrain high-quality embeddings from large vocabularies, that can be later reused for downstream tasks.

But if we want to increase the model performance on test data, we can only upscale model capacity in terms of the number of trainable parameters by introducing more hidden layers or hidden units (which might work to some point, until the model starts to overfit) or we need to increase the size of the training data [85, pp. 112-117, 426-427], [110]. For our case, this would mean to indirectly increase the amount of

[2] except, e.g., for word sense ambiguities, but as we have seen before, we can also use contextual embeddings to capture different senses

available information on the downstream tasks, by pre-training the embeddings on large external training datasets.

However, we trained our activity embeddings on the same dataset as was used on the downstream task (next activity prediction). The information that is used during embedding training is exactly the same as during training for the task, without any additional new information. We speculate that the convergence of the neural network models to near identical performances for all encoding methods used in our evaluation with the early stopping method (One-Hot, Random, CBOW and Skip-Gram embeddings), could suggest that the neural networks learn the underlying patterns of the data anyway by updating the input parameters in the optimization process. Whether the starting point are activities represented by equidistant points in space (one-hot encoding), or if they are enriched with semantics (embeddings), does not seem to make a difference.

The second reason, we believe, that might contribute to our finding, that embeddings do not lead to a significant increase in performance, when compared to one-hot encoding, is the potential for dimensionality reduction. For one-hot encoding, the vector size is equal to the number of unique activities. When learning embeddings, the dimensionality is a parameter of choice, but typically it is chosen significantly lower than vocabulary size. In fact, we did not decrease the dimensionality of the activity vectors, but rather increase it, since 256 showed similar results while having lower training times ([14] also used embedding dimension 500 for next-activity-prediction on datasets, where the number of activities is smaller).

For comparison, the vocabulary size for word embeddings from [2] was 692k, while the embedding dimension was only 300, leading to a drastic decrease in dimensionality. The highest number of unique activities we had in an event logs, was 624 in the Hospital dataset in our first experiments and only 26 for the BPI Challenge 2017 for our final evaluation. Thus, the positive effects of dimensionality reduction may not be applicable in the same way in process mining, as they are in NLP, since generally event logs do not contain a number of unique activities nearly in the order of magnitude of the vocabulary sizes in NLP.

Our third possible explanation, which might inhibit the performance of activity embeddings, is the structural differences between language and event logs and the lack of adaptation of embedding techniques to the process mining domain. Word embedding techniques exploit the distributional hypothesis (word context in form of word co-occurrence statistics) to focus on capturing word semantics in the learned vectors. The two techniques we employed, CBOW and Skip-Gram from [1], both neglect syntax (word order) in their neural network setup (input position for CBOW and output position for Skip-Gram, see section 3.2.1). This may work well for word meaning, since the concept, that similar words occur in similar context (windows), ignoring the order or word proximity, might not suffer to the same degree as syntactic (positional) understanding ([1, 2, 111].

However, syntax and the relative activity position may be crucial when trying to embed process control flow information in activity vectors. Additional activity tagging for exclusive or parallel branches etc. could help here. Another option could be to employ some type of RNN architecture instead of FNNs, which can memorize positional dependencies (see, e.g., [112, 113]). This could help to better embed the underlying control flow information in the activity vectors [5].

Finally, we want to discuss some limitations of our experiments. It is important to highlight, that these results are obtained only for the task of next activity prediction, and thus we should not draw any premature conclusions to other process mining tasks. Following on from the last point of our discussion, we only tried two embedding techniques, CBOW and Skip-Gram, and did not adapt them specifically to the process mining domain. As discussed earlier, focusing on syntax instead of semantics, might be crucial to learn control flow related information.

We did not include additional attributes from the event log, such as resource, timestamp, or other data elements when learning our embeddings, but only activity sequence information. Neural network architectures could be extended to leverage this information, as [5] also suggested. We conclude, that the small selection of techniques and the lack of adaption to the process mining domain are two major limitations of our experiments and might be of interest for future research.

5 Conclusion

We had three research objectives for this master thesis: providing a systematic literature on word and sentence embedding techniques, the exploration of interpretability of activity embeddings and the evaluation of activity embeddings as input for next activity prediction. In the following, we want to review the achievement of these objectives in the three main chapters of this work, and we also want to point to some potential areas of future research.

In Chapter 1 we gave an introduction to business process management, process mining, in particular next activity prediction. We illustrated how pretrained embeddings are employed in the NLP domain to capture word meaning and enhance the performance on downstream tasks. We also explored the potential of these techniques, when applied to process mining. Additionally, we stated our three research objectives and presented related work at the end of the chapter.

Subsequently, we provided a comprehensive overview on word and sentence embedding techniques in chapter 2. Our research involved a systematic exploration of literature across three scientific databases, supplemented by a search on Google Scholar, resulting in approximately 3900 studies, that we found. After including studies from backward search and applying a three-phase selection protocol incorporating inclusion and exclusion criteria, 37 studies remained. The included studies either proposed new techniques for learning word or sentence embeddings or improved existing ones. We further categorized the word embedding methods as static, contextual, predictive or count-based techniques and organized the studies chronologically in their publication date, before we started to explain each technique in more detail. After gaining a broad overview and understanding of existing embedding methods from NLP, we opted for the Skip-Gram and CBOW model from [1, 2] to learn activity embeddings for our later experiments on interpretability and next activity prediction, due to their simplicity, ease of use and proven effectiveness.

Following that, we explored the interpretability of activity embeddings learned by the Skip-Gram and CBOW models on synthetic event logs and one real world event log from the BPI Challenge 2017 in chapter 3. While results seemed hard to interpret for the synthetic event logs at first, we observed, that the activity embeddings of the real word event log encoded two interesting properties of activities: i) the distances between two activities within traces, indicating their possible relative positions in the underlying process model and ii) the relative frequency of their co-occurrences in shared paths as their strength of association. We came to finding i) by taking a closer look at the Skip-Gram objective and how two activities share contexts, to map activities to vectors that follow the distributional hypothesis. For finding ii) we applied the PMI, which was shown to be a measure of association between target and context in Skip-Gram [43], to explain how relative path frequencies of two activities influence the similarity for the learned vectors.

In chapter 4 we pretrained activity embeddings on two event logs in a first experiment to determine a good hyperparameter choice for the embedding dimension by measuring neural network prediction performance and training time in next activity prediction. To our surprise, we got consistently lower training time by increasing embedding dimension until 256. In the subsequent experiment, we compared the next activity prediction performance of activity embeddings of dimension 256 (Random, CBOW, Skip-Gram) and one-hot encoded activities as input for two neural network architectures (FNN and LSTM) on three datasets with 5×2-fold cross validation.

We found no significant performance difference in F1-Score between models using embeddings and one-hot encoded activities as input, as we fail to reject the null hypothesis, when applying a paired Student's t-test modified for machine learning model comparison on all three datasets. Following that, we identified three potential reasons for our findings: the inability to learn from external datasets to gain additional information, inherent to the process mining domain, the insufficient potential for dimensionality reduction and the lack of adaption to the task and process mining in general. Again, the source code for our experiments is available at GitHub[1] for reproducibility or for own use.

While addressing the limitations of our work, we also highlight some potential areas for future work. We think, the adaption of NLP embedding techniques to the process mining domain to improve the quality of the learned embeddings should be further investigated. While equally weighting words in context windows might work to learn semantically-rich representations in NLP, this might not be the case for process mining, when trying to capture control information. Also, we did not learn embeddings considering other event information than the activity. However, extending the embedding techniques to include resource, timestamp, or other data elements could be very valuable information to improve on down stream tasks. Lastly, our evaluation focused exclusively on the task of next activity prediction. One could further explore the impact of embeddings on various other process mining tasks using not only activity embeddings, but trace, log, and model embeddings, as well.

[1] https://github.com/Christian-Gebhardt/act2vec_evaluation

Bibliography

[1] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient estimation of word representations in vector space,” *arXiv preprint arXiv:1301.3781*, 2013.

[2] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their compositionality,” *Advances in neural information processing systems*, vol. 26, 2013.

[3] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, “Natural language processing (almost) from scratch,” *Journal of machine learning research*, vol. 12, no. ARTICLE, pp. 2493–2537, 2011.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of deep bidirectional transformers for language understanding,” in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, (Minneapolis, Minnesota), pp. 4171–4186, Association for Computational Linguistics, June 2019.

[5] P. De Koninck, S. vanden Broucke, and J. De Weerdt, “act2vec, trace2vec, log2vec, and model2vec: Representation learning for business processes,” in *Business Process Management: 16th International Conference, BPM 2018, Sydney, NSW, Australia, September 9–14, 2018, Proceedings 16*, pp. 305–321, Springer, 2018.

[6] M. Weske, *Business Process Management Architectures*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2019.

[7] M. Dumas, M. L. Rosa, J. Mendling, and H. A. Reijers, *Fundamentals of Business Process Management, Second Edition*. Springer, 2018.

[8] W. M. P. van der Aalst, *Process Mining: Data Science in Action*. Heidelberg: Springer, 2 ed., 2016.

[9] W. M. van der Aalst and J. Carmona, *Process mining handbook*. Springer Nature, 2022.

[10] W. M. van der Aalst, “Foundations of process discovery,” in *Process Mining Handbook*, pp. 37–75, Springer, 2022.

[11] G. T. Lakshmanan, D. Shamsi, Y. N. Doganata, M. Unuvar, and R. Khalaf, "A markov prediction model for data-driven semi-structured business processes," *Knowledge and Information Systems*, vol. 42, pp. 97–126, 2015.

[12] M. Unuvar, G. T. Lakshmanan, and Y. N. Doganata, "Leveraging path information to generate predictions for parallel business processes," *Knowledge and Information Systems*, vol. 47, pp. 433–461, 2016.

[13] D. Breuker, M. Matzner, P. Delfmann, and J. Becker, "Comprehensible predictive models for business processes," *Mis Quarterly*, vol. 40, no. 4, pp. 1009–1034, 2016.

[14] J. Evermann, J.-R. Rehse, and P. Fettke, "Predicting process behaviour using deep learning," *Decision Support Systems*, vol. 100, pp. 129–140, 2017.

[15] S. Schönig, R. Jasinski, L. Ackermann, and S. Jablonski, "Deep learning process prediction with discrete and continuous data features," in *Proceedings of the 13th international conference on evaluation of novel approaches to software engineering*, pp. 314–319, 2018.

[16] M. Camargo, M. Dumas, and O. González-Rojas, "Learning accurate lstm models of business processes," in *Business Process Management: 17th International Conference, BPM 2019, Vienna, Austria, September 1–6, 2019, Proceedings 17*, pp. 286–302, Springer, 2019.

[17] C. Guo and F. Berkhahn, "Entity embeddings of categorical variables," *arXiv preprint arXiv:1604.06737*, 2016.

[18] Z. S. Harris, "Distributional structure," *Word*, vol. 10, no. 2-3, pp. 146–162, 1954.

[19] J. Firth, "A synopsis of linguistic theory, 1930-1955," *Studies in linguistic analysis*, 1957.

[20] F. Almeida and G. Xexéo, "Word embeddings: A survey," *arXiv preprint arXiv:1901.09069*, 2019.

[21] L. Gutiérrez and B. Keith, "A systematic literature review on word embeddings," in *Trends and Applications in Software Engineering: Proceedings of the 7th International Conference on Software Process Improvement (CIMPS 2018) 7*, pp. 132–141, Springer, 2019.

[22] S. Hong, M. Wu, H. Li, and Z. Wu, "Event2vec: Learning representations of events on temporal sequences," in *Web and Big Data: First International Joint*

Conference, APWeb-WAIM 2017, Beijing, China, July 7–9, 2017, Proceedings, Part II 1, pp. 33–47, Springer, 2017.

[23] N. A. Wahid, T. N. Adi, H. Bae, and Y. Choi, "Predictive business process monitoring–remaining time prediction using deep neural network with entity embedding," *Procedia Computer Science*, vol. 161, pp. 1080–1088, 2019.

[24] D. A. Neu, J. Lahann, and P. Fettke, "A systematic literature review on state-of-the-art deep learning methods for process prediction," *Artificial Intelligence Review*, pp. 1–27, 2022.

[25] A. Al-Jebrni, H. Cai, and L. Jiang, "Predicting the next process event using convolutional neural networks," in *2018 IEEE International Conference on Progress in Informatics and Computing (PIC)*, pp. 332–338, IEEE, 2018.

[26] N. Di Mauro, A. Appice, and T. M. Basile, "Activity prediction of business process instances with inception cnn models," in *AI* IA 2019–Advances in Artificial Intelligence: XVIIIth International Conference of the Italian Association for Artificial Intelligence, Rende, Italy, November 19–22, 2019, Proceedings 18*, pp. 348–361, Springer, 2019.

[27] A. Khan, H. Le, K. Do, T. Tran, A. Ghose, H. Dam, and R. Sindhgatta, "Memory-augmented neural networks for predictive process analytics," *arXiv preprint arXiv:1802.00938*, 2018.

[28] L. Lin, L. Wen, and J. Wang, "Mm-pred: A deep predictive model for multi-attribute event sequence," in *Proceedings of the 2019 SIAM international conference on data mining*, pp. 118–126, SIAM, 2019.

[29] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, and S. Linkman, "Systematic literature reviews in software engineering–a systematic literature review," *Information and software technology*, vol. 51, no. 1, 2009.

[30] Y. Bengio, R. Ducharme, P. Vincent, and C. Janvin, "A neural probabilistic language model," *J. Mach. Learn. Res.*, vol. 3, p. 1137–1155, mar 2003.

[31] R. Collobert and J. Weston, "A unified architecture for natural language processing: Deep neural networks with multitask learning," in *Proceedings of the 25th international conference on Machine learning*, pp. 160–167, 2008.

[32] A. Mnih and K. Kavukcuoglu, "Learning word embeddings efficiently with noise-contrastive estimation," *Advances in neural information processing systems*, vol. 26, 2013.

[33] W. Ling, C. Dyer, A. W. Black, and I. Trancoso, “Two/too simple adaptations of word2vec for syntax problems,” in *Proceedings of the 2015 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 1299–1304, 2015.

[34] T. Cohen and D. Widdows, “Bringing order to neural word embeddings with embeddings augmented by random permutations (earp),” in *Proceedings of the 22nd Conference on Computational Natural Language Learning*, pp. 465–475, 2018.

[35] B. Wang, D. Zhao, C. Lioma, Q. Li, P. Zhang, and J. G. Simonsen, “Encoding word order in complex embeddings,” *arXiv preprint arXiv:1912.12333*, 2019.

[36] Y. Song, S. Shi, J. Li, and H. Zhang, “Directional skip-gram: Explicitly distinguishing left and right context for word embeddings,” in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pp. 175–180, 2018.

[37] W. Ling, Y. Tsvetkov, S. Amir, R. Fermandez, C. Dyer, A. W. Black, I. Trancoso, and C.-C. Lin, “Not all contexts are created equal: Better word representations with variable attention,” in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pp. 1367–1372, 2015.

[38] S. Sonkar, A. E. Waters, and R. G. Baraniuk, “Attention word embedding,” *arXiv preprint arXiv:2006.00988*, 2020.

[39] D. L. Rohde, L. M. Gonnerman, and D. C. Plaut, “An improved model of semantic similarity based on lexical co-occurrence,” *Communications of the ACM*, vol. 8, no. 627-633, p. 116, 2006.

[40] R. Lebret and R. Collobert, “Word emdeddings through hellinger pca,” *arXiv preprint arXiv:1312.5542*, 2013.

[41] R. Lebret and R. Collobert, “Rehabilitation of count-based models for word vector representations,” in *Computational Linguistics and Intelligent Text Processing: 16th International Conference, CICLing 2015, Cairo, Egypt, April 14-20, 2015, Proceedings, Part I 16*, pp. 417–429, Springer, 2015.

[42] J. Pennington, R. Socher, and C. D. Manning, “Glove: Global vectors for word representation,” in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[43] O. Levy and Y. Goldberg, "Neural word embedding as implicit matrix factorization," *Advances in neural information processing systems*, vol. 27, 2014.

[44] A. Salle, M. Idiart, and A. Villavicencio, "Matrix factorization using window sampling and negative sampling for improved word representations," *arXiv preprint arXiv:1606.00819*, 2016.

[45] K. Stratos, M. Collins, and D. Hsu, "Model-based word embeddings from decompositions of count matrices," in *Proceedings of the 53rd annual meeting of the association for computational linguistics and the 7th international joint conference on natural language processing (Volume 1: Long papers)*, pp. 1282–1291, 2015.

[46] P. S. Dhillon, D. P. Foster, and L. H. Ungar, "Eigenwords: spectral word embeddings.," *J. Mach. Learn. Res.*, vol. 16, pp. 3035–3078, 2015.

[47] J. Reisinger and R. Mooney, "Multi-prototype vector-space models of word meaning," in *Human Language Technologies: the 2010 annual conference of the north american chapter of the association for computational linguistics*, pp. 109–117, 2010.

[48] J. Reisinger and R. Mooney, "A mixture model with sharing for lexical semantics," in *Proceedings of the 2010 Conference on Empirical Methods in Natural Language Processing*, pp. 1173–1182, 2010.

[49] P. Dhillon, D. P. Foster, and L. Ungar, "Multi-view learning of word embeddings via cca," *Advances in neural information processing systems*, vol. 24, 2011.

[50] E. H. Huang, R. Socher, C. D. Manning, and A. Y. Ng, "Improving word representations via global context and multiple word prototypes," in *Proceedings of the 50th annual meeting of the association for computational linguistics (Volume 1: Long papers)*, pp. 873–882, 2012.

[51] F. Tian, H. Dai, J. Bian, B. Gao, R. Zhang, E. Chen, and T.-Y. Liu, "A probabilistic model for learning multi-prototype word embeddings," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, pp. 151–160, 2014.

[52] A. Neelakantan, J. Shankar, A. Passos, and A. McCallum, "Efficient non-parametric estimation of multiple embeddings per word in vector space," *arXiv preprint arXiv:1504.06654*, 2015.

[53] Y. Liu, Z. Liu, T.-S. Chua, and M. Sun, "Topical word embeddings," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, 2015.

[54] P. Liu, X. Qiu, and X. Huang, "Learning context-sensitive word embeddings with neural tensor skip-gram model.," in *IJCAI*, pp. 1284–1290, 2015.

[55] S. Bartunov, D. Kondrashkin, A. Osokin, and D. Vetrov, "Breaking sticks and ambiguities with adaptive skip-gram," in *artificial intelligence and statistics*, pp. 130–138, PMLR, 2016.

[56] Y. Sun, N. Rao, and W. Ding, "A simple approach to learn polysemous word embeddings," *arXiv preprint arXiv:1707.01793*, 2017.

[57] S. Li, Y. Zhang, R. Pan, and K. Mo, "Adaptive probabilistic word embedding," in *Proceedings of The Web Conference 2020*, pp. 651–661, 2020.

[58] Y. Meng, J. Huang, G. Wang, Z. Wang, C. Zhang, and J. Han, "Unsupervised word embedding learning by incorporating local and global contexts," *Frontiers in big Data*, vol. 3, p. 9, 2020.

[59] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *International conference on machine learning*, pp. 1188–1196, PMLR, 2014.

[60] J. Li and E. Hovy, "A model of coherence based on distributed sentence representation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 2039–2048, 2014.

[61] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," *Advances in neural information processing systems*, vol. 28, 2015.

[62] F. Hill, K. Cho, and A. Korhonen, "Learning distributed representations of sentences from unlabelled data," *arXiv preprint arXiv:1602.03483*, 2016.

[63] M. Pagliardini, P. Gupta, and M. Jaggi, "Unsupervised learning of sentence embeddings using compositional n-gram features," *arXiv preprint arXiv:1703.02507*, 2017.

[64] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[65] P. Gupta and M. Jaggi, "Obtaining better static word embeddings using contextual embedding models," *arXiv preprint arXiv:2106.04302*, 2021.

[66] K. Lund and C. Burgess, "Producing high-dimensional semantic spaces from lexical co-occurrence," *Behavior research methods, instruments, & computers*, vol. 28, no. 2, pp. 203–208, 1996.

[67] S. Deerwester, S. T. Dumais, G. W. Furnas, T. K. Landauer, and R. Harshman, "Indexing by latent semantic analysis," *Journal of the American society for information science*, vol. 41, no. 6, pp. 391–407, 1990.

[68] T. K. Landauer and S. T. Dumais, "A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge.," *Psychological review*, vol. 104, no. 2, p. 211, 1997.

[69] H. Schütze, "Automatic word sense discrimination," *Computational linguistics*, vol. 24, no. 1, pp. 97–123, 1998.

[70] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[71] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, *et al.*, "Improving language understanding by generative pre-training," 2018.

[72] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017.

[73] P. Vincent, H. Larochelle, I. Lajoie, Y. Bengio, P.-A. Manzagol, and L. Bottou, "Stacked denoising autoencoders: Learning useful representations in a deep network with a local denoising criterion.," *Journal of machine learning research*, vol. 11, no. 12, 2010.

[74] R. Řehůřek, "Word2vec." `https://radimrehurek.com/gensim/models/word2vec.html`, 2023. Date last accessed 04-October-2023.

[75] L. Finkelstein, E. Gabrilovich, Y. Matias, E. Rivlin, Z. Solan, G. Wolfman, and E. Ruppin, "Placing search in context: The concept revisited," in *Proceedings of the 10th international conference on World Wide Web*, pp. 406–414, 2001.

[76] F. Hill, R. Reichart, and A. Korhonen, "Simlex-999: Evaluating semantic models with (genuine) similarity estimation," *Computational Linguistics*, vol. 41, no. 4, pp. 665–695, 2015.

[77] A. Burattin, B. Re, L. Rossi, and F. Tiezzi, "A purpose-guided log generation framework," in *International Conference on Business Process Management*, pp. 181–198, Springer, 2022.

[78] "Purple." `https://pros.unicam.it/purple`, 2023. Date last accessed 04-October-2023.

[79] O. Levy and Y. Goldberg, “Linguistic regularities in sparse and explicit word representations,” in *Proceedings of the eighteenth conference on computational natural language learning*, pp. 171–180, 2014.

[80] B. F. van Dongen, “Bpi challenge 2017.” `https://data.4tu.nl/articles/dataset/BPI_Challenge_2017/12696884`, 2017. Date last accessed 10-October-2023.

[81] L. Blevi, L. Delporte, and J. Robbrecht, “Process mining on the loan application process of a dutch financial institute,” *BPI Challenge*, pp. 328–343, 2017.

[82] G. Scheithauer, R. Henne, A. Kerciku, R. Waldenmaier, and U. Riedel, “Suggestions for improving a bank’s loan application process based on a process mining analysis,” in *13th International BPM Workshop on Business Process Intelligence. 7th International Business Process Intelligence Challenge (BPIC), Barcelona*, pp. 1–30, 2017.

[83] C. W. Günther and A. Rozinat, “Disco: Discover your processes.,” *BPM (Demos)*, vol. 940, no. 1, pp. 40–44, 2012.

[84] “Disco.” `https://fluxicon.com/disco`, 2023. Date last accessed 04-October-2023.

[85] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.

[86] Z. C. Lipton, J. Berkowitz, and C. Elkan, “A critical review of recurrent neural networks for sequence learning,” *arXiv preprint arXiv:1506.00019*, 2015.

[87] Y. Bengio, P. Simard, and P. Frasconi, “Learning long-term dependencies with gradient descent is difficult,” *IEEE transactions on neural networks*, vol. 5, no. 2, pp. 157–166, 1994.

[88] S. Hochreiter and J. Schmidhuber, “Long short-term memory,” *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[89] G. AI, “Tensorflow.” `https://www.tensorflow.org`, 2023. Date last accessed 04-October-2023.

[90] A. R. Barron, “Universal approximation bounds for superpositions of a sigmoidal function,” *IEEE Transactions on Information theory*, vol. 39, no. 3, pp. 930–945, 1993.

[91] N. Tax, I. Verenich, M. La Rosa, and M. Dumas, “Predictive business process monitoring with lstm neural networks,” in *Advanced Information Systems*

Engineering: 29th International Conference, CAiSE 2017, Essen, Germany, June 12-16, 2017, Proceedings 29, pp. 477–492, Springer, 2017.

[92] Y. Goldberg, "A primer on neural network models for natural language processing," *Journal of Artificial Intelligence Research*, vol. 57, pp. 345–420, 2016.

[93] M. Polato, "Dataset belonging to the help desk log of an italian company." `https://data.4tu.nl/articles/dataset/Dataset_belonging_to_the_help_desk_log_of_an_Italian_Company/12675977`, 2017. Date last accessed 10-October-2023.

[94] B. F. van Dongen, "Real-life event logs - hospital log." `https://data.4tu.nl/articles/dataset/Real-life_event_logs_-_Hospital_log/12716513`, 2011. Date last accessed 10-October-2023.

[95] B. F. van Dongen, "Bpi challenge 2012." `https://data.4tu.nl/articles/dataset/BPI_Challenge_2012/12689204`, 2012. Date last accessed 10-October-2023.

[96] F. Mannhardt, "Sepsis cases - event log." `https://data.4tu.nl/articles/dataset/Sepsis_Cases_-_Event_Log/12707639`, 2016. Date last accessed 10-October-2023.

[97] "Xes standard." `https://xes-standard.org`, 2023. Date last accessed 04-October-2023.

[98] F. I. for Industrial Engineering (IAO), "pm4py." `https://pm4py.fit.fraunhofer.de`, 2023. Date last accessed 04-October-2023.

[99] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[100] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural Networks: Tricks of the Trade: Second Edition*, pp. 437–478, Springer, 2012.

[101] N. Corporation, "Nvidia cuda toolkit." `https://developer.nvidia.com/cuda-toolkit`, 2023. Date last accessed 11-October-2023.

[102] T. G. Dietterich, "Approximate statistical tests for comparing supervised classification learning algorithms," *Neural computation*, vol. 10, no. 7, pp. 1895–1923, 1998.

[103] M. Grandini, E. Bagli, and G. Visani, "Metrics for multi-class classification: an overview," *arXiv preprint arXiv:2008.05756*, 2020.

[104] M. Sokolova and G. Lapalme, “A systematic analysis of performance measures for classification tasks,” *Information processing & management*, vol. 45, no. 4, pp. 427–437, 2009.

[105] L. O. Jimenez and D. A. Landgrebe, “Supervised classification in high-dimensional space: geometrical, statistical, and asymptotical properties of multivariate data,” *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, vol. 28, no. 1, pp. 39–54, 1998.

[106] J.-N. Hwang, S.-R. Lay, and A. Lippman, “Nonparametric multivariate density estimation: a comparative study,” *IEEE Transactions on Signal Processing*, vol. 42, no. 10, pp. 2795–2810, 1994.

[107] Z. Yin and Y. Shen, “On the dimensionality of word embedding,” *Advances in neural information processing systems*, vol. 31, 2018.

[108] Z. Huang, G. Zweig, M. Levit, B. Dumoulin, B. Oguz, and S. Chang, “Accelerating recurrent neural network training via two stage classes and parallelization,” in *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, pp. 326–331, IEEE, 2013.

[109] D. E. Rumelhart, G. E. Hinton, and R. J. Williams, “Learning representations by back-propagating errors,” *nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[110] V. N. Vapnik and A. Y. Chervonenkis, “On the uniform convergence of relative frequencies of events to their probabilities,” *Theory of Probability and its Applications*, vol. 16, no. 2, pp. 264–280, 1971.

[111] T. Mikolov, W.-t. Yih, and G. Zweig, “Linguistic regularities in continuous space word representations,” in *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pp. 746–751, 2013.

[112] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur, “Recurrent neural network based language model.,” in *Interspeech*, vol. 2, pp. 1045–1048, Makuhari, 2010.

[113] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, “Learning phrase representations using rnn encoder-decoder for statistical machine translation,” *arXiv preprint arXiv:1406.1078*, 2014.

List of Figures

1.1 Business Process Model Example 2
1.2 BPM Lifecycle 4
1.3 Event Log Example 6
1.4 Process Mining Techniques in Business Process Management 8
1.5 Example BPMN Model Next Event Prediction 9
1.6 Word Embeddings in Vector Space 13

2.1 Categorization of Identified Studies 24
2.2 CBOW and Skip-Gram Architecture 29
2.3 COALS Word-Word Correlation Matrix 31
2.4 Contextual Multi-Prototype Approach 32
2.5 PV-DM-Model 35

3.1 Sequential Example Process Model 41
3.2 Exclusive Example Process Model 42
3.3 Parallel Example Process Model 42
3.4 Activity Embeddings Sequential 42
3.5 Activity Embeddings Exclusive 43
3.6 Activity Embeddings Parallel 43
3.7 Activity Embeddings BPI 2017 44
3.8 Embeddings Heatmap BPI 2017 46
3.9 BPI Challenge 2017 Process Map (Part 1) 49
3.10 BPI Challenge 2017 Process Map (Part 2) 50
3.11 BPI Challenge 2017 Process Map (Part 3 50
3.12 BPI Challenge 2017 Process Map (Part 4) 51

4.1 Feed Forward Neural Net 55
4.2 LSTM Cell Structure 56

List of Tables

2.1 Summary of Search Results 21
2.2 Comparison of Word Embedding Techniques (1) 25
2.3 Comparison of Word Embedding Techniques (2) 26

4.1 Dataset Summary (1) 59
4.2 Dataset Summary (2) 59
4.3 Embedding Performance Comparison Helpdesk 66
4.4 Embedding Performance Comparison Hospital 67
4.5 Statistical Test Results 68
4.6 Next Activity Prediction Performance Comparison Sepsis 69
4.7 Next Activity Prediction Performance Comparison BPI 2012 69
4.8 Next Activity Prediction Performance Comparison BPI 2017 70
4.9 Training Time Comparison Embeddings Helpdesk 71
4.10 Extended Training Time Comparison Embeddings Helpdesk 72
4.11 Training Time Comparison Embeddings Hospital 73
4.12 Extended Training Time Comparison Embeddings Hospital 74

Name: Christian Gebhardt Matrikelnr.: 1579200

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit eigenständig und ohne fremde Hilfe angefertigt habe. Textpassagen, die wörtlich oder dem Sinn nach auf Publikationen oder Vorträgen anderer Autoren beruhen, sind als solche kenntlich gemacht. Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und auch noch nicht veröffentlicht.

Bayreuth, 20. November. 2023 ....Christian Gebhardt..................

Christian Gebhardt