UNIVERSITÄT KOBLENZ · LANDAU

IWVI

Konzeption, Implementierung und Test eines Business Process Modeling Recommender Systems auf Basis von Long Short-Term Memory Neural Networks

Bachelorarbeit

Zur Erlangung des Grades eines Bachelor of Science im Studiengang Informatik

vorgelegt von

Alexander Ritschl

[215 200 172]

Koblenz, im Juli 2019

Erstgutachter:	Prof. Dr. Patrick Delfmann
	(Institut für Wirtschafts- und Verwaltungsinformatik, AG Delfmann)
Zweitgutachter:	M.Sc. Christoph Drodt
	(Institut für Wirtschafts- und Verwaltungsinformatik, AG Delfmann)
Betreuer:	Prof. Dr. Patrick Delfmann

Zusammenfassung

Das Ziel dieser Arbeit ist es, zu bestimmen, ob neuronale Netze (insbesondere LSTM) zur Prozessvorhersage eingesetzt werden können. Dabei soll eine möglichst genaue Vorhersage zu dem Nachfolger eines Events getroffen werden.

Dazu wurde Python mit dem Framework TensorFlow genutzt, um ein rekurrentes neuronales Netz zu erstellen. Dabei werden zwei Netze erstellt, wobei das eine für das Training und das andere für die Vorhersage genutzt wird.

Die verwendeten Datensätze bestehen aus mehreren Prozessen mit jeweils mehreren Events. Mit diesen Prozessen wird das Netz trainiert und die Parameter nach dem Training gespeichert. Das Netz zur Vorhersage nutzt dann dieselben Parameter, um Vorhersagen zu Events zu treffen.

Das neuronale Netz ist in der Lage, nachfolgende Events eindeutig vorherzusagen. Auch Verzweigungen können vorhergesagt werden.

In der weiteren Entwicklung ist eine Einbindung in andere Programme möglich. Dabei ist es empfehlenswert, auf eine eindeutige Benennung der Events zu achten oder eine geeignete Umbenennung durchzuführen.

Abstract

The main goal of this paper is to ascertain, if neural networks (especially LSTM) are helpful in predicting processes by making predictions as accurately as possible.

TensorFlow is the used framework in Python to build recurrent neural networks. Two networks are built, whereby one is used for training and the other one for prediction.
Used datasets contain several processes with several events each. With those processes, the network ist trained and afterwards, the parameters are saved. The network for prediction uses these parameters to make predictions.

The neural network is able to make clear predictions about subsequent events. Even branches can be predicted.

When developed further, integration in other programs is possible. It is recommended to use unique names for the events or to rename them.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Abkürzungsverzeichnis

CPU ..... Central Processing Unit
GPU ..... Graphics Processing Unit
RNN ..... Rekurrentes Neuronales Netz
LSTM ..... Long Short-Term Memory
BPMN ..... Business Process Model and Notation

1 Einleitung

Mit Hilfe von neuronalen Netze ist es mittlerweile möglich, realistische Gesichter von Menschen zu erstellen, die kaum von echten zu unterscheiden sind (z. B. Karras et al. 2019). Dabei dienen reale Gesichter als Datensatz. Diese werden überlagert, wobei das neuronale Netz die jeweiligen "high-level attributes" (vgl. Karras et al. 2019, S. 1), also Merkmale wie Identität oder Pose, von den "stochastic variaton[s]" (vgl. Karras et al. 2019, S. 1), zum Beispiel Sommersprossen oder Haare) unterscheiden und diese so beliebiger miteinander kombinieren kann.
Zusätzlich ist das Netz nicht auf Gesichter beschränkt. Es kann die Bilder jedes zugrundeliegenden Datensatzes kombinieren und neue Kombinationen erzeugen.

Mit Long Short-Term Memory (LSTM) Netzen können auch Eingabesequenzen erkannt und verarbeitet werden. Damit ist es zum Beispiel möglich, dass ein neuronales Netz eine Sprache lernt (z. B. Gers & Jürgen 2001). Hierbei kann das Netz sowohl kontextfreie als auch kontextsensitive Sprachen lernen, was mit rekurrenten neuronalen Netzen ohne LSTM nicht möglich sei (vgl. Gers & Jürgen 2001, S. 2).

In der Prozessmodellierung ist es hilfreich, Vorhersagen über Abläufe treffen zu können, um die Planung oder auch den Prozess selbst effizienter zu gestalten.
Ziel dieser Arbeit ist es, diese Vorhersagen innerhalb von Prozessen mittels eines rekurrenten neuronalen Netzes (RNN) zu treffen.
Dazu werden zunächst eigene Datensätze zum Trainieren und Testen sowie ein RNN mit TensorFlow erstellt. Im Anschluss wird das Netz auf diese Datensätze trainiert und trifft danach Vorhersagen mithilfe der trainierten Parameter.
Im Folgenden wird das Training sowie die Vorhersage dargestellt. Dazu wird zuerst aufgezeigt, wie das Netz aufgebaut ist. Danach ist das Training dargestellt und im Anschluss wird die Vorhersage erläutert, wobei hierbei auch auf die Genauigkeit der Vorhersage eingegangen wird. Abschließend werden die Ergebnisse dargestellt, um festzustellen, ob rekurrente neuronale Netze zur Prozessvorhersage eingesetzt werden können.
Neuronale Netze sind meist nur auf ein Problem oder eine Aufgabe beschränkt (etwa Erzeugen von Gesichtern). Daher ist es notwendig für das vorliegende Problem ein auf dieses ausgerichtete neuronale Netz zu erstellen. Dabei liegt der Fokus darauf, dass das Netz eine bestimmte Struktur der genutzten Datensätze verarbeiten kann (siehe Abschnitt Datensätze).

2 Grundlagen

Im folgenden Kapitel werden Grundlagen zu Recommender Systemen und neuronalen Netzen dargestellt. Dabei wird insbesondere auf die in dieser Arbeit verwendeten Formen von neuronalen Netzen eingegangen (LSTM) und die Funktionsweise beschrieben. Außerdem werden die verwendeten Datensätze sowie das genutzte Framework TensorFlow zusammengefasst.

2.1 Recommender Systems

Ein *Recommender System* oder Empfehlungsdienst ist ein System, das dem Nutzer aus einer großen Menge an Daten je nach Absicht des Nutzers eine passende kleinere Menge empfiehlt, um so eine Entscheidungshilfe zu liefern und eine bessere Übersicht zu schaffen (vgl. Adomavicius & Tuzhilin 2005, S. 734).

In den Neunzigerjahren entwickelten sich Recommender Systems zu einem eigenen Forschungsgebiet, das sich vor allem auf bewertungsorientierte Empfehlungsdienste konzentrierte (vgl. Adomavicius & Tuzhilin 2005, S. 734).
Es gibt verschiedene Kategorien von Recommender Systems. *Inhaltliche* Empfehlungsdienste liefern Vorschläge basierend auf den Präferenzen des Nutzers in der Vergangenheit. *Kollaborative* Recommender Systems orientieren sich an den Vorlieben anderer Nutzer mit gleichen Interessen. Außerdem gibt es Mischformen aus diesen zwei Kategorien, die beide Ansätze kombinieren.
Zusätzlich unterscheidet man zwischen *absoluten* und *relativen* Empfehlungen. Bei der absoluten Empfehlung wird für jedes Objekt eine individuelle Bewertung erstellt. Im Gegensatz dazu werden bei relativen Empfehlungen die Objekte untereinander verglichen und so eine Einstufung erstellt (vgl. Adomavicius & Tuzhilin 2005, S. 735).
Nach (vgl. Adomavicius & Tuzhilin 2005, S. 734, Introductions) können heutige Recommender Systems zum Beispiel im Bezug auf die Darstellung der Information, die Einbeziehung kontextbezogener Informationen und das Bewertungssystem noch weiter verbessert werden.
Bekannte Einsatzgebiete von Empfehlungsdiensten sind etwa Filmvorschläge basierend auf den bisher angesehenen Filmen oder Kaufvorschläge bei Onlineplattformen, die sich auf bereits gekaufte Artikel oder Artikel, die andere Nutzer gekauft haben, beziehen.

Das in dieser Arbeit vorgestellte und entwickelte Recommender System ist ein inhaltlicher Empfehlungsdienst. Das System bezieht sich auf vorhandene Prozesse und deren konkreten Ablauf und liefert anhand dessen Empfehlungen. Es hat sowohl absolute als auch relative

Merkmale, da es zu jeder Empfehlung eine Wahrscheinlichkeit berechnet, aber die einzelnen Objekte auch in Relation zueinander stehen.
Es werden zum Beispiel drei Empfehlungen (A, B, C) gemacht. Zu jeder Empfehlung wird eine Wahrscheinlichkeit angegeben, zum Beispiel hat A eine Wahrscheinlichkeit von 50 %, B 30 % und C 20 %. Jeder Vorschlag hat also einen konkreten Wert, steht aber auch in Relation zu den anderen.

2.2 Neuronale Netze

Im Folgenden werden neuronale Netze im Allgemeinen und die in dieser Arbeit verwendeten weiteren Formen dargestellt.

Ein neuronales Netz ist eine gewichtete Verkettung mehrerer Neuronen. Dabei sind diese in Schichten angeordnet und, im Falle von feedforward Netzen, von einer Schicht zur nächsten miteinander verbunden (feedforward Netz, siehe Abbildung 2.1).
Jedes Netz besteht aus einer Eingabeschicht (2.1, grün), die die Eingaben verarbeitet, und einer Ausgabeschicht (2.1, rot), welche die Ausgaben zurückgibt. Dazwischen liegt die Hidden Layer (versteckte Schicht, 2.1, grau), die aus mehreren Schichten bestehen kann und den eigentlichen Rechenanteil des Netzes übernimmt. Dabei ist die Anzahl der Neuronen in einer Schicht von Netz zu Netz verschieden.

Modelle, die den neuronalen Netzen ähneln, werden schon seit den Sechziger- und Siebzigerjahren genutzt. Seitdem wurde auch backpropagation entwickelt (vgl. Schmidhuber 2015, S. 86). Dabei handelt es sich um eine Methode zum überwachten Lernen (supervised learning). Hierbei muss den Überwacher zu jedem Input der gewünschte Output bekannt sein. Durch backpropagation werden dann die Parameter des Netzes angepasst. So nähert sich das Netz immer weiter den gewollten Ergebnissen an.
Dieses Verfahren wurde 1981 auf neuronale Netze angewandt. In den Neunzigerjahren wurden neuronale Netze mit vielen Schichten ein eigenes Forschungsgebiet. Seit 2000 wird neuronalen Netzen immer mehr Aufmerksamkeit zuteil, da sie andere Methoden des Machine Learnings übertreffen (vgl. Schmidhuber 2015, S. 86).
Die am häufigsten genutzte Methode im Machine Learning ist das oben bereits erwähnte supervised learning (überwachtes Lernen). Dabei entsteht die Vorhersage des neuronalen Netzes durch eine Zuordnung $f(x)$, mit der eine Ausgabe y für jedes x erstellt wird. Diese Zuordnung kann in verschiedenen Formen stattfinden, z. B. durch Entscheidungsbäume, logistische Regression oder die hier behandelten neuronalen Netze (vgl. Jordan & Mitchell 2015, S. 257).
Einen großen Einfluss hatten in den letzten Jahren die Deep Networks. Diese bestehen aus mehreren Schichten mit mehreren Neuronen, von denen jedes einige einfache Parameter

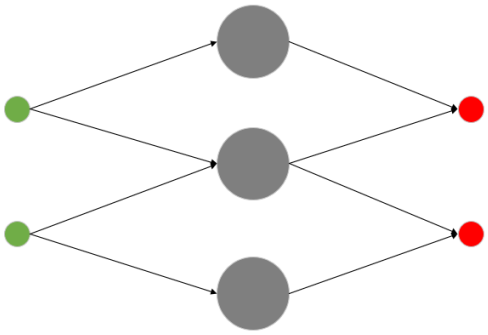

Abbildung 2.1: Beispielhafte Darstellung eines neuronalen Netzes (eigene Darstellung in Anlehnung an Zhang et al. (1998, S. 38))

berechnet. Mittels Gradientverfahren (z. B. backpropagation) können diese Netze trainiert werden (vgl. Jordan & Mitchell 2015, S. 257). Dabei berechnet das Netz beim Training für die Eingabe eine dazugehörige Ausgabe. Danach wird die erwartete Ausgabe mit der errechneten verglichen. Diese Differenz ist der Fehler des Netzes. Mit diesem Fehler wird nun rückwärts über die Schichten zurückgerechnet. Dabei werden die Gewichtungen der einzelnen Verbindungen zwischen den Neuronen angepasst. Durch diese Anpassung nähert man sich bei jedem weiteren Durchlauf der gewünschten Ausgabe an. Hierbei handelt es somit auch um supervised learning, da die erwartet Ausgabe dem Netz bekannt gemacht werden muss.
Hilfreich für die Berechnung dieser Vielzahl von Parametern sind die modernen GPUs. Mit diesen ist es mittlerweile möglich, das Training von neuronalen Netzen um mindestens den Faktor 50 zu beschleunigen (vgl. Schmidhuber 2015, S. 90).
Im Gegensatz zum supervised learning steht das unsupervised learning (unbeaufsichtigtes Lernen). Dabei werden unstrukturierte Eingabedaten im Hinblick auf strukturelle Eigesnchaften analysiert. Unsupervised learning wird häufig zum Clustering eingesetzt (vgl. Jordan & Mitchell 2015, S. 258). Hierbei handelt es um die Gruppierung von Objekten aufgrund ihrer Eigenschaften.
Eine dritte Methode ist das reinforcement learning (bestärkendes Lernen). Anstatt Eingabedaten mit erwarteten Ausgaben gibt es hier Trainingsdaten, die nur Anzeigen enthalten, ob eine ausgeführte Aktion korrekt oder nicht korrekt ist. Bei einer falschen Aktion muss weiter nach der richtigen gesucht werden. So entwickelt das Netz eine eigene Strategie der Problemlösung (vgl. Jordan & Mitchell 2015, S. 258).

Bei dem in dieser Arbeit vorgelegten und beschriebenen Netz handelt es sich um supervised learning, da zu einer Eingabe immer eine erwartete Ausgabe existiert (siehe Abschnitt Training).

2.2.1 Rekurrente Neuronale Netze

In einem rekurrenten neuronalen Netz sind auch Verknüpfungen zwischen den Neuronen in derselben Schicht oder zu vorherigen Schichten möglich (siehe Abbildung 2.2).

Nach Schmidhuber seien RNN die "deepest of all NNs [neural networks]"(vgl. Schmidhuber 2015, S. 4). Der große Vorteil von rekurrenten neuronalen Netzen liege darin, dass sie "sequential and parallel information processing" (vgl. Schmidhuber 2015, S. 4) miteindander vermischen können und in der Lage sind, "memories of arbitrary sequences of input patterns" (vgl. Schmidhuber 2015, S. 49) zu erstellen und zu verarbeiten.
Bei sequenziellen Eingaben, z. B. bei der Sprachverarbeitung, liegt der Vorteil von RNNs darin, dass sie zwar zum gegebene Zeitpunkt ein Element einer Eingabesequenz betrachten, aber im Hintergrund ein "state vector" (vgl. LeCun et al. 2015, S. 441) existiert, der alle Informationen der bereits bearbeiteten Elemente der Sequenz beinhaltet. Mit diesem Status eignen sich RNNs zum Beispiel gut zu Wort- oder Buchstabenvorhersage in einem Text oder Wort (vgl. LeCun et al. 2015, S. 441).
RNNs werden vor allem für langfristige Zusammenhänge genutzt. Dennoch ist es ihnen kaum möglich, Informationen lange zu behalten (vgl. LeCun et al. 2015, S. 442).

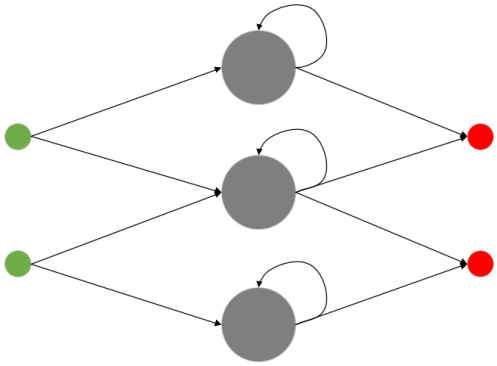

Abbildung 2.2: Beispielhafte Darstellung eines rekurrenten neuronalen Netzes (eigene Darstellung in Anlehnung an Zhang et al. (1998, S. 38))

2.2.2 Long Short-Term Memory

Long Short-Term Memory Netze (Hochreiter & Schmidhuber 1997) gehören zu den rekurrenten neuronalen Netzen (siehe Abbildung 2.3).
Durch spezielle "memory cell[s]" (vgl. LeCun et al. 2015, S. 442) ist es LSTM Netzen möglich, das Problem der RNN zu vermeiden und Informationen länger zu behalten. Dabei hat die Speicherzelle eine Verbindung zu sich selbst zum nächsten Zeitpunkt. So wird der Status der Zelle weitergegeben (vgl. LeCun et al. 2015, S. 442).

Im Folgenden werden *weights* (liegen als Vektoren vor) mit W, *biases* mit b, der Eingabevektor mit x und der Zustandsvektor mit h bezeichnet. σ ist die Sigmoidfunktion und $tanh$ der Tangens hyperbolicus. Alle Formeln sind in Anlehnung an (Hochreiter & Schmidhuber 1997) und (Evermann et al. 2017).

Zuerst wird im *forget gate* (siehe Formel 2.1, orange in Abbildung 2.3) berechnet, wie viel des neuen Inputs t_t behalten wird. Dazu wird aus W_{xf} und x_t sowie aus W_{hf} und h_{t-1}

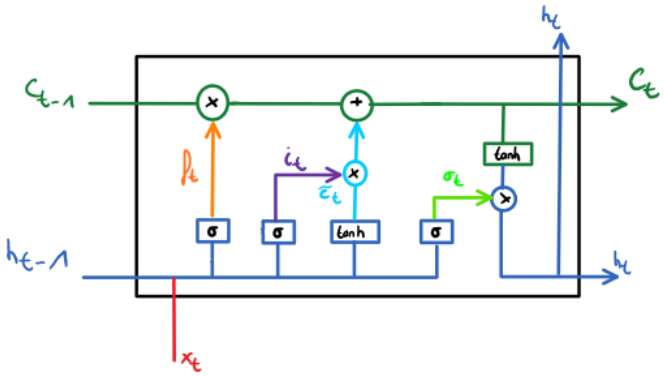

Abbildung 2.3: LSTM Zelle (eigene Darstellung, in Anlehnung an Hochreiter & Schmidhuber (1997, S. 7))

jeweils die Skalarprodukte gebildet. Anschließend werden diese beiden sowie b_f addiert und auf die Summe die Sigmoid-Funktion angewendet.
Die Sigmoidfunktion σ ergibt einen Wert zwischen Null und Eins. Bei einer Null werden alle Information vergessen, bei einer Eins keine.

$$f_t = \sigma(W_{xf}x_t + W_{hf}h_{t-1} + b_f) \tag{2.1}$$

Im Folgenden läuft die Berechnung sehr ähnlich zu f_t. Die Skalarprodukte werden aus der Eingabe x_t mit der jeweiligen Gewichtung W_x sowie aus dem Status der vorherigen Zelle h_{t-1} und der zugehörigen Gewichtung W_h gebildet und zum bias addiert. Danach wird eine Aktivierungsfunktion (Sigmoid, Tangens hyperbolicus) auf die Summe angewendet.

Mit $\tilde{c}_t$ (Formel 2.3 hellblau in Abbildung 2.3) wird der neue Zellzustand mithilfe des Tangens hyperbolicus berechnet und das *input gate* (Formel 2.2, lila in Abbildung 2.3) beeinflusst, wie viel der Informationen in die neue Zelle übergeht. Auch hier liefert die Sigmoidfunktion Werte zwischen Null und Eins (bei Null wenig Informationen, bei Eins viel Informationen).

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_{t-1} + b_i) \tag{2.2}$$

$$\tilde{c}_t = tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \tag{2.3}$$

Anschließend wird in Formel 2.4 das *output gate* berechnet.

$$o_t = \sigma(W_{ox}x_t + W_{oh}h_{t-1} + b_o) \tag{2.4}$$

Die Ergebnisse des *forget gates* f, des Zellzustands der vorherigen Zelle c_{t-1}, des *input gates* i und des neuen Zellzustandes $\tilde{c}$ ergeben dann den aktuellen Zustand c_t (Formel 2.5, grün in Abbildung 2.3) der Zelle. Dazu wird mittels Skalarprodukt $f_t c_{t-1}$ berechnet, wie viele Informationen aus dem vorherigen Zustand behalten werden, und mit dem Skalarprodukt $i_t\tilde{c}_t$, was der neue Input der Zelle ist. Aus der Summe diesen alten und neuen Informationen

werden dann die aktuellen Informationen der Zelle berrechnet.

$$c_t = f_t c_{t-1} + i_t \tilde{c}_t \tag{2.5}$$

Abschließend wird aus dem aktuellen Zustand die Aktivierung der Zelle berechnet und mit dem *output gate* verrechnet. Somit hängen die aktuellen Informationen der Zelle h_t (Formel 2.6, dunkelblau in Abbildung 2.3) vom Faktor o_t ab. Dieser bestimmt, wie viele Informationen weitergegeben werden.

$$h_t = o_t * tanh(c_t) \tag{2.6}$$

2.3 TensorFlow

TensorFlow (seit 2015) ist ein open-source Framework für C++ und Python von Google, das es ermöglicht, Machine Learning Algorithmen darzustellen und diese auszuführen. Es basiert auf dem Projekt DistBelief (2011), dem Vorgänger von TensorFlow. Dabei ist es möglich, die Ergebnisse von TensorFlow auch auf mobilen Geräten (iOS und Android) zu nutzen. Im Vergleich zum Vorgänger DistBelief sei die Programmierung in TensorFlow flexibler, die Performance besser und das Training für eine Vielzahl an Plattformen möglich (vgl. Abadi et al. 2016, S. 1).
Das Modell zur Berechnung in TensorFlow wird durch einen gerichteten Graphen dargestellt, der aus einer Menge an Knoten besteht. Jeder Knoten erhält dabei keine oder mehrere Eingaben und übergibt keine oder mehrere Ausgaben. Die Werte, die dabei an den Kanten übergeben werden, werden *tensors* genannt. Tensoren sind Arrays von beliebiger Dimension, deren Typ festgelegt wird, wenn sie erstellt werden (vgl. Abadi et al. 2016, S. 2).
Im Gegensatz zu den während der Programmausführung immer wieder geänderten oder verworfenen *tensors* existieren in TensorFlow Variablen. Diese enthalten Werte in Form von Vektoren, welche aber während der Ausführung des Programms nachhaltig verwaltet werden und immer wieder benutzt werden können (vgl. Abadi et al. 2016, S. 3).
Google nutzt TensorFlow aktuell für die online Handschrifterkennung. Dazu verwendet man "user input in the form of an ink" (vgl. Carbune et al. 2019, S. 1) und gibt die "textual interpretation of this input" (vgl. Carbune et al. 2019, S. 1) zurück.

Alternativen zu TensorFlow sind Microsoft Cognitive Toolkit und Theano von der University of Montreal. Aufgrund der hohen Popularität und der guten Dokumentation wurde in dieser Arbeit trotz dieser Alternativen TensorFlow genutzt.

2.4 Datensätze

Die Datensätze sind erstellte BPMN (Business Process Model and Notation) Modelle (siehe Abbildungen 2.4 und 2.5) und die dazugehörigen möglichen Prozesse, die als Daten in Form von Arrays (*tensors*) an das Netz weitergegeben werden (siehe Abbildung 2.6).

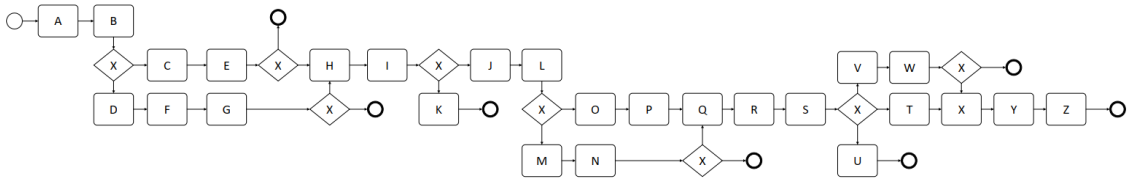

Abbildung 2.4: BPMN zu Datensatz 3 (eigene Darstellung)

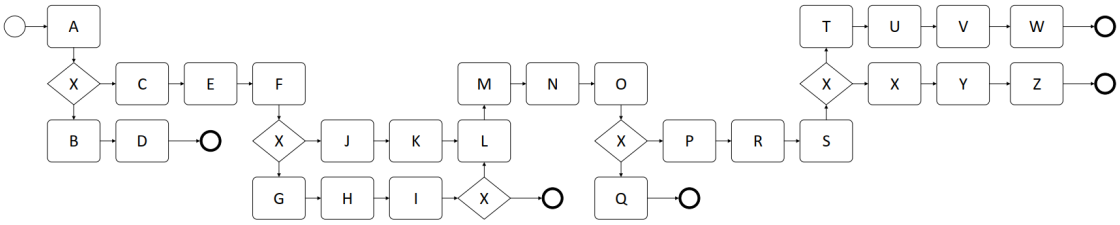

Abbildung 2.5: BPMN zu Datensatz 4 (eigene Darstellung)

Die Datensätze sind eigens erstellt. Sie entsprechen keinem tatsächlichen Prozess, sondern dienen nur als Beispiele. Zur Vereinfachung wurden für die Benennung der einzelnen Events die Buchstaben des Alphabets genutzt.
In den verwendeten Datensätzen haben alle Pfade die gleiche Wahrscheinlichkeit. Das heißt, an einer Verzweigung sind alle Events gleich wahrscheinlich. In einer Verzweigung mit zwei Events besteht für jedes die Wahrscheinlichkeit von 50 Prozent.
Jeder Datensatz besteht aus mehreren Zeilen. Dabei entspricht jede Zeile einer Prozessinstanz. Die einzelnen Events sind durch ein Zeichen getrennt (hier "-", siehe Abbildung 2.6 zur Verdeutlichung der Nutzereingabe). Dies ist notwendig, da es zwar auch möglich wäre, jeden Buchstaben einzeln einzulesen. Das geht aber nur in dem einfachen Fall, dass einzelne Buchstaben für eine Eventbezeichnung genutzt werden. In diesem fiktiven, selbst erstellten Beispiel werden die Buchstaben "A" bis "Z" verwendet und jeder steht für ein Event. Falls Events mit einem oder mehreren Wörtern benannt werden, funktioniert diese Methode des Einlesens nicht mehr. Somit wird also jedes einzelne Event durch ein Zeichen vom nächsten abgetrennt.
Am Ende jeder Zeile steht ein Zeilenumbruch "\n", der das Ende eines Prozesses markiert. Wegen dieses Zeilenumbruchs sind auch Trennzeichen am Anfang und am Ende jeder Zeile notwendig, da sonst der Zeilenumbruch mit zum nächsten Event der nächsten Zeile zählen würde (z. B. "\nA" statt "A").

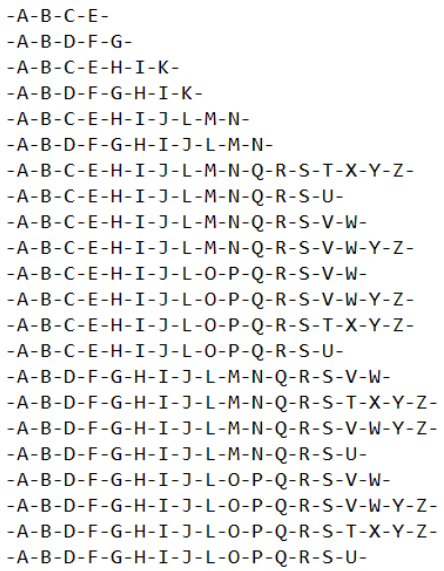

Abbildung 2.6: Prozesse zu Datensatz 3 (eigene Darstellung)

3 Implementierung eines neuronalen Netzes auf Basis von LSTM

Das folgende Kapitel behandelt die Umsetzungen des neuronalen Netzes. Dabei wird auf den Aufbau des Netzes in verschiedenen Schichten sowie deren Funktion im Netz eingegangen.

3.1 Aufbau des Netzes

Das erstellte Netz (siehe Abbildung 3.1) besteht aus einer Eingabeschicht, der *hidden layer* und einer Ausgabeschicht.
3.1 zeigt dabei nicht nur den Aufbau des Netzes, sondern auch eine Übersicht zum Ablauf des implementierten Programmes. Die Datensätze werden zuerst als Zeichenkette eingelesen (hier als String). Danach wird jedem Event ein Zahlenwert zugeordnet.

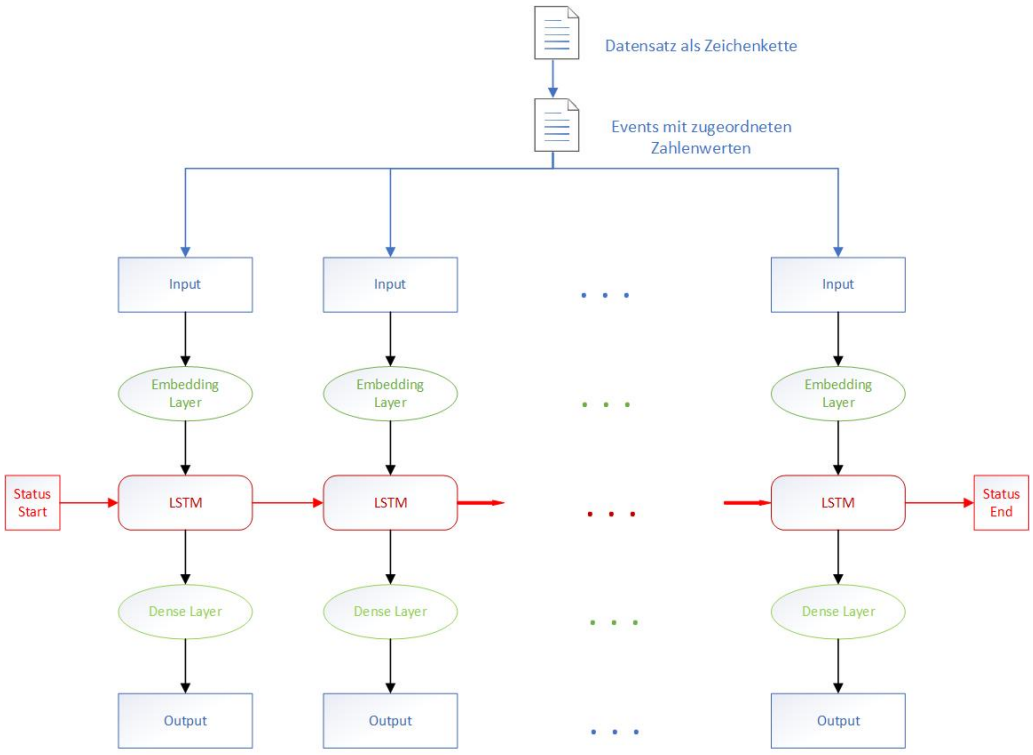

Abbildung 3.1: Aufbau des Modells in TensorFlow (eigene Darstellung)

Die Eingabeschicht (embedding layer in TensorFlow) transformiert Zahlenwerte in Vektoren (Tensoren), mit denen TensorFlow arbeiten kann.

Dafür werden die Zeichenketten, also die Eventbezeichnungen, in den Datensätzen in Zahlen umgewandelt. Jedem Event wird genau ein Zahlenwert zugeordnet. Da die Events hier als die 26 Buchstaben des Alphabets dargestellt werden, bekommen diese die Zahlen 1 bis 26 und der Zeilenumbruch den Wert 0 zugeordnet.
Die so generierten Zahlenwerte werden von der Eingabeschicht nur eingelesen und in passender Dimension (siehe Abschnitt Dimension der Schichten) in Form von Vektoren an die nächste Schicht weitergegeben.
Der *hidden layer* besteht aus einer Schicht von LSTM Zellen, deren Anzahl vom Nutzer festgelegt werden kann (siehe Abschnitt Parameter). Diese können trainiert und so an die vorhandenen Daten angepasst werden (siehe Abschnitt Training). Dabei heißen die LSTM Zellen hier "CuDNNLSTM" (siehe Abbildung 3.2), da diese über die Grafikkarte trainiert werden können (siehe Abschnitt Nutzung der Graphics Processing Unit (GPU)).
In der letzten Schicht (dense layer in TensorFlow und Output) werden die Ergebnisse des *hidden layer* ausgegeben. Hier kann auch eine Aktivierungsfunktion genutzt werden. In diesem Fall ist diese die lineare Funktion $f(x) = x$, da die Ergebnisse nur unverändert ausgegeben werden sollen.

3.1.1 Nutzung der Graphics Processing Unit (GPU)

NVIDIA CUDA ("Compute Unified Device Architecture") ist eine Softwarearchitektur, bei der Hardware und Software zusammenarbeiten. Sie ermöglicht es GPUs von NVIDIA, Programme auszuführen. Dabei werden mehrere Tausende Threads und Hunderte von Prozessorkernen genutzt. Dies ermöglicht die parallele Ausführung von Programmteilen (vgl. Nickolls & Dally 2010, S. 59). NVIDIA CUDA ermöglicht es, CPU (Central Processing Unit) und GPU zusammenarbeiten zu lassen. Serielle Programmteile werden dabei durch die CPU und parallele Teile durch die GPU ausgeführt. Diese Zusammenarbeit optimiert die Performance des Programms (vgl. Nickolls & Dally 2010, S. 64).

Die in dieser Implementation verwendeten LSTM Zellen heißen "CuDNNLSTM". "CuDNN" steht für das *Deep Neural Network* von *NVIDIA CUDA* (*CuDNN* (2019)). Diese werden verwendet, weil sie mit der GPU schneller trainiert werden können als die normalen LSTM Zellen, welche über die CPU trainiert werden.

3.1.2 Dimension der Schichten

Jede Schicht übergibt ihre Ausgabe in bestimmten Dimensionen (siehe Abbildung 3.2), die von den Parametern (siehe Abschnitt Parameter) des Netzes abhängig sind.
Dabei entspricht der erste Wert der Dimension jeder Schicht der Größe der *Batches*. Dabei wird der Datensatz in mehrere Batches, also kleine Teilstücke zerlegt. Diese werden nacheinander vom Netz zum Training genutzt. Nach jedem Batch passt das Netz seine internen Parameter an. Dieser Wert ist die 64 in der Abbildung 3.2). Dabei ist der Wert grundsätzlich frei wählbar.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (64, None, 512)           13824
_________________________________________________________________
cu_dnnlstm (CuDNNLSTM)       (64, None, 512)           2101248
_________________________________________________________________
dense (Dense)                (64, None, 27)            13851
=================================================================
Total params: 2,128,923
Trainable params: 2,128,923
Non-trainable params: 0
_________________________________________________________________
```

Abbildung 3.2: Dimensionen des Netzes zum Training (Darstellung mit TensorFlow)

Der zweite Wert ist bei allen Schichten *None*. Dieser Wert None steht aber nicht für eine nicht angegebenen Parameter, sondern dafür, dass der Wert variabel ist. Dieser entspricht hier der Sequenzlänge, die abgearbeitet wurde. Diese ist aber nicht in jedem Batch gleich und somit variabel.
In der Embedding Schicht entspricht der dritte Wert des festgelegten Parameters für diese Schicht. Für die zweite Schicht gibt der Wert die Anzahl der LSTM Zellen an, welcher auch in den Parametern festgelegt wird. In der Ausgabeschicht (*Dense*) wird der Wert durch die Länge des Vokabulars, also der Anzahl der eindeutigen Events, festgelegt. Dabei besteht das Vokabular aus den 26 Buchstaben des Alphabets und dem Zeilenumbruch. Somit hat es die Länge 27.

3.2 Training

Der Datensatz wird zum Training verändert, um eine Trainingsgrundlage zu schaffen. Das letzte Event jedes Prozesses wird als erwartete Ausgabe verwendet. Die vorigen Events bilden die Eingabe. Somit kann das Netz mit der geschaffenen Eingabe trainiert und das Ergebnis mit dem letzten erwarteten Event verglichen werden.

Das Netz wird über mehrere Epochen trainiert (siehe Abbildung 3.3, Zeile 3). Eine Epoche ist ein Durchgang, in dem alle Batches vom Netz abgearbeitet werden und somit der ganze Datensatz das Netz ein Mal durchlaufen hat.
Dabei wird aus jeder Zeile des Datensatzes der Input sowie der zu vorhersagende Output eingelesen. Der Input wird in das Netz gegeben, damit dieses dazu eine Vorhersage berechnet. Danach wird der vorhergesagte Wert des Netzes mit dem tatsächlichen Wert aus dem Datensatz verglichen (Zeile 6) und so der *loss*, also die Abweichung vom erwarteten Wert, berechnet.
TensorFlow kann automatisch alle trainierbaren Parameter abrufen. Diese werden dann mit dem *loss* und einem Optimierer neu berechnet (Zeile 7). Dieser Vorgang wird für jede Zeile im Datensatz wiederholt.

```
datensatz = leseDatensatz

für jede Epoche:
    für jede Zeile(input, ziel) in datensatz:
        erstelleVorhersage
        vergleicheVorhersageMitZiel
        berechneNeueParameter

    speichereParameter

    wenn lossDifferenzKlein:
        brichTrainingAb
```

Abbildung 3.3: Pseudocode für das Training (eigene Darstellung)

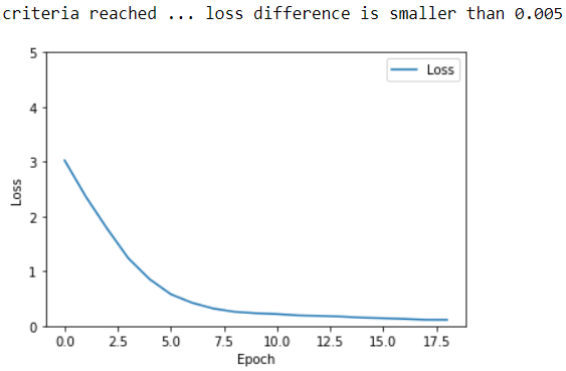

Abbildung 3.4: Grafische Darstellung des *loss* des Trainings zu Datensatz 3 (Darstellung mit matplotlib in Python)

Anschließend werden die berechneten Parameter gespeichert, damit man sie nach dem Training wieder abrufen kann. Das Training des Netzes wird abgebrochen, wenn der Unterschied des *loss* der aktuellen Epoche zur vorherigen kleiner als ein vorher festgelegter Wert ist. Danach wird der *loss* aus jeder Epoche in einem Koordinatensystem grafisch dargestellt (siehe Abbildung 3.4).
Dadurch, dass ein bereits trainiertes Netz wieder abgerufen werden kann, ist es nicht notwendig, jedes Mal ein neues Netz zu trainieren. Übergibt man die gleichen Parameter, zu denen bereits ein Training gespeichert wurde, kann dieser Trainingsstand verwendet werden.

Das Netz wird in jeder Epoche über mehrere Batches trainiert und der *loss* berechnet. Am Ende jeder Epoche wird der letzte *loss*, die Genauigkeit (*accuracy*) sowie die Zeit, die für das Training gebraucht wurde, ausgegeben (siehe Abbildung 3.5).

3.2.1 Parameter

Der Nutzer kann mehrere Parameter für das Training und die Ausgabe selbst einstellen (Optimierer, Größe der Batches, Dimension der Embedding Schicht, Anzahl der LSTM Zellen, Trainingsepochen, Schwellenwert (siehe Abschnitt Vergleich zweier Datensätze) und Epochen, in denen der Trainingsstand gespeichert wird).
Außerdem ist es möglich, die Anzahl der getroffenen Vorhersagen sowie die Anzahl der

```
Epoch 9 Batch 0 Loss 0.1169
Epoch 9 Batch 1 Loss 0.1225
Epoch 9 Batch 2 Loss 0.1320
Epoch 9 Batch 3 Loss 0.1238
Epoch 9 Batch 4 Loss 0.1212
Epoch 9 Batch 5 Loss 0.1131
Epoch 9 Batch 6 Loss 0.1168
Epoch 9 Batch 7 Loss 0.1127
Epoch 9 Batch 8 Loss 0.1154
Epoch 9 Batch 9 Loss 0.1067
Epoch 9 Batch 10 Loss 0.1184
Epoch 9 Batch 11 Loss 0.1060
Epoch 9 Batch 12 Loss 0.1152
Epoch 9 Loss 0.1152 Accuracy 0.9685
Time taken for epoch 9: 1.20 sec
```

Abbildung 3.5: Beispiel des Trainings zu Datensatz 3 (eigene Darstellung)

dann als Vorhersagen angezeigten Events festzulegen. Eine höhere Anzahl an Vorhersagen erhöht die Genauigkeit.

Für den Aufbau des Netzes relevante Werte sind Embeddingdimension, Anzahl der Zellen im *hidden layer* und Typ der Zellen (LSTM oder GRU), da sie den Aufbau der einzelnen Schichten festlegen.
Das Training selbst kann eingestellt werden, indem man die Anzahl der Epochen, den Schwellenwert des loss und den Optimierer festlegt. Außerdem kann der Nutzer festlegen, ob und wie oft ein Trainingszustand gespeichert wird. Es kann sowohl gar nicht, als auch in bestimmten Abständen gespeichert werden, etwa in jeder oder in jeder zehnten Epoche.
Für die Vorhersage können die Anzahl der getroffenen sowie die Anzahl der ausgegebenen Vorhersagen festgesetzt werden. Eine höhere Zahl an getroffenen Vorhersagen erhöht die Genauigkeit. Bei der Ausgabe werden immer die Events mit der höchsten Wahrscheinlichkeit angezeigt. Diese Liste lässt sich in einer beliebigen Länge anzeigen. Sollte die Angabe des Nutzers zur Länge der Liste größer sein als die Anzahl an Vorhersagen, die vom Netz überhaupt gemacht werden, werden nur die tatsächlich getroffenen Vorhersagen angezeigt. Sollte der Nutzer zum Beispiel eine Liste der Länge fünf fordern, das Netz liefert hingegen nur drei Events als Vorhersage, werden auch nur diese drei Events angezeigt.
Einmalig festzusetzende Werte sind die Pfade zu den Datensätzen und den gespeicherten Trainingszuständen.

Alle Parameter können in einer JSON Datei festgelegt werden. Die Pfade zu den jeweiligen Datensätzen können entweder im Quellcode selbst angepasst werden oder es wird der angegeben Pfad erstellt und genutzt.

3.2.2 Vergleich zweier Datensätze

Vergleicht man die Datensätze 3 (siehe Abbildung 2.4) und 4 (siehe Abbildung 2.5) für jeweils 512 und 1024 LSTM Zellen bei gleicher Dimension der Embeddingschicht, ist zu sehen, dass eine Verdopplung der LSTM Zellen in mehr als einer Verdopplung der Trainingszeit resultiert (vergleiche Tabelle 3.1). Für Datensatz 4 ergibt sich sogar fast eine Verdreifachung der Trainingszeit. Dabei bleiben die in dieser Zeit trainierten Epochen in etwa gleich. Für beide Datensätze werden für 1024 Zellen nur jeweils zwei Epochen mehr trainiert.
Der *loss* ist bei beiden Datensätzen mit 1024 Zellen nicht viel niedriger als bei 512 Zellen (Differenz etwa 0.7 für Datensatz 3 und 0.2 für Datensatz 4).
Dabei ist zu beachten, dass das Netz für jede Konfiguration nur einmal trainiert wurde. Es sind also die Messwerte eines einzigen Trainings und keine Mittelwerte. Dennoch lässt sich sagen, dass eine Verdopplung der Zellen nur in einer Vervielfachung der Trainingszeit resultiert, aber keinen großen Gewinn bei dem Traningsresultat liefert.
Das Training wurde jeweils abgebrochen, nachdem die Differenz des *loss* der aktuellen Epoche weniger als 0.005 kleiner war als der *loss* der vorherigen Epoche.
Dieser Schwellenwert (*treshold*) ist vom Nutzer frei einstellbar oder auch deaktivierbar. Wie die grafischen Darstellungen zeigen (siehe Abbildung 3.4), gibt es beim Training einen Wert beim *loss*, dem sich das Netz immer weiter annäherd. Da die Differenzen des *loss* von Epoche zu Epoche weniger werden, wird Zeit beim Training eingespart, indem man das Training abbricht, wenn der Schwellenwert in der Differenz unterschritten ist. Falls der Schwellenwert nicht unterschritten wird oder deaktiviert ist, wird das Netz über die angegebenen Epochen trainiert.

Tabelle 3.1: Vergleich des Trainings zweier Datensätze, eigene Darstellung

Datensatz	Units	Embedding-Dim.	Epochen	*loss*	Zeit
Datensatz 3	512	512	16	0.1512	13.01s
Datensatz 3	1024	512	18	0.888	30.57s
Datensatz 4	512	512	12	0.0933	11.27s
Datensatz 4	1024	512	14	0.0791	30.57s

3.3 Vorhersage

Für die Vorhersage wird zuerst ein neues Netz erstellt. Das ist notwendig, da nun nicht mehr mehrere Batches, sondern nur noch eine Eingabe abgerufen werden (siehe Abbildung 3.6 und 3.7). Beim Netz, welches für das Training genutzt wurde, entsprach der erste Wert jeder Schicht der Anzahl der Batches, mit denen traininert wurde. In diesem Netz, das zur Vorhersage genutzt wird, wird nicht mehr trainiert. Es bekommt daher nur eine Eingabe (siehe Abbildung 3.6, vergleiche Abbildung 3.2). Diese Eingabe entspricht dem Event oder dem Pfad (siehe Abschnitt Vorhersage nach einzelnen/mehreren Events), dessen Nachfolger vorhergesagt werden soll.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (1, None, 512)            13824
_________________________________________________________________
cu_dnnlstm_1 (CuDNNLSTM)     (1, None, 512)            2101248
_________________________________________________________________
dense_1 (Dense)              (1, None, 27)             13851
=================================================================
Total params: 2,128,923
Trainable params: 2,128,923
Non-trainable params: 0
_________________________________________________________________
```

Abbildung 3.6: Aufbau des Models zur Vorhersage (mit TensorFlow)

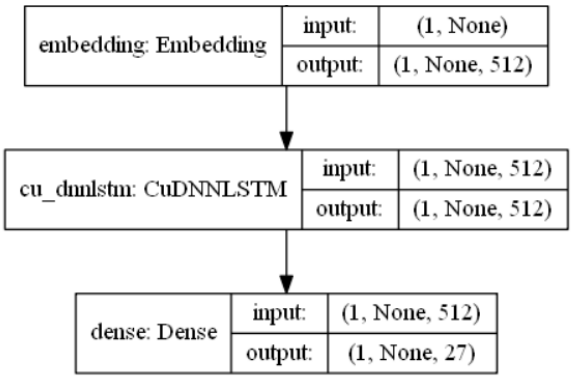

Abbildung 3.7: Aufbau des Models zur Vorhersage (Darstellung mit plot_model von Keras)

Deswegen sind die Dimensionen im Vergleich mit dem Netz, das für das Training genutzt wurde (siehe Abbildung 3.2), etwas verändert. Die Batch-Größe entspricht nicht mehr dem vorher festgelegten Parameter, sondern dem Wert Eins.
In dieses neue Netz werden dann die vorher trainierten Parameter geladen, damit man mit einem bereits trainierten Netz Vorhersagen treffen kann.

Es ist erforderlich, dass die Parameter für das Netz zur Vorhersage, die gleichen sind wie bei dem zum Training genutzten Netz. Andernfalls ist ein Abrufen der gespeicherten Werte nicht möglich, was zu einer Fehlermeldung führt.
Dieses Netz liefert Vorhersagen zu einem gegebenen Event oder einer Reihe von Events. Vorhersage und Training laufen vergleichbar ab. Das Event, dessen Nachfolger das Netz vorhergesagt, liegt als Zeichenkette (String) vor. Diese wird erst in einen Zahlenwert übersetzt (vergleiche Abschnitt Aufbau des Netzes und Abbildung 3.8, Zeilen 1 und 2) und dann in das Netz gegeben. Das Netz erstellt mit den vorher trainierten Parametern eine

```
1  input_string = "INPUT"
2  input_int = transformiereZuZahlenwert(input_string)
3
4  für AnzahlVorhersagen:
5      erstelleVorhersage
6      fügeVorhersageZuInputHinzu
7
8  gibVorhersagenAus
```

Abbildung 3.8: Pseudocode für die Vorhersage (eigene Darstellung)

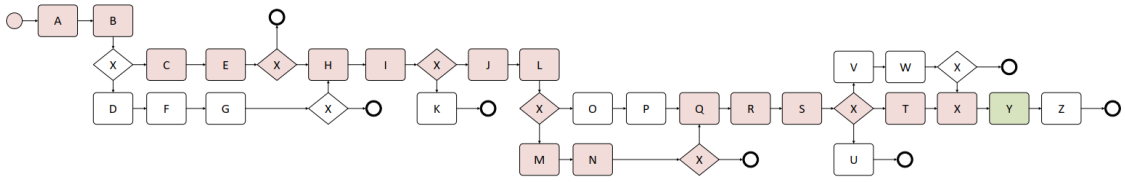

Abbildung 3.9: Pfad "ABCEHIJLMNQRSTX" in Datensatz 3 (eigene Darstellung)

Vorhersage (Zeile 5). Daraus wird mittels Multinomialverteilung ein Wert berechnet (siehe Abschnitt Unterschiede bei der Vorhersage). Danach wird die Vorhersage zum Input hinzugefügt und es könnte eine weitere Vorhersage mit der so entstandenen neuen Eingabe getroffen werden. Somit ist es möglich, auch einen Pfad, der auf das aktuelle Event folgt, vorherzusagen.
Dieses Erstellen einer Vorhersage wird mehrmals durchgeführt und dann eine Liste mit Wahrscheinlichkeiten für das nächste Event ausgegeben. Dabei wird nur eine gewisse Anzahl an Events ausgegeben, die der Nutzer vorher einstellen kann, wobei die Events der Wahrscheinlichkeit nach absteigend angezeigt werden.
In Zeile 1 von Tabelle 3.2 werden mögliche Nachfolger für das Event "X" vorhergesagt. Das wahrscheinlichste Nachfolgerevent ist "Y", was dem tatsächlichen Nachfolgerevent entspricht (siehe BPMN zu Datensatz 3, Abbildung 2.4).

Tabelle 3.2: verschiedene Vorhersagen für Datensatz 3 (eigene Darstellung)

Eingabe	Vorhersagen		
"X"	"Y", 16.0%	"T", 10.0%	"U", 8.0%
"Q-R-S-T-X"	"Y", 81.0%	"X", 9.0%	"W", 2.0%
"A-B-C-E-H-I-J-L-M-N-Q-R-S-T-X"	"Y", 100.0%		
"Q-R-S"	"V", 45.0%	"U", 28.0%	"T", 25.0%

Es besteht aber auch die Möglichkeit, dem Netz mehr als ein Event als Eingabe zu geben. Je länger die Eingabe, desto genauer ist die Vorhersage.
Übergibt man dem Netz statt "X" als Eingabe "QRSTX", also noch die vier direkten Vorgängerevents von "X", erhält man eine Wahrscheinlichkeit für Event "Y" von 81 Prozent (siehe Zeile 2 von Tabelle 3.2). Bei der Eingabe des kompletten bisher abgelaufenen Prozesses (siehe Abbildung 3.9, roter Pfad) ist die Vorhersage für "Y" (Abbildung 3.9, grünes Rechteck) sogar eindeutig (siehe Zeile 3 von Tabelle 3.2).

Betrachtet man den Pfad "ABCEHIJLMNQRS" und dessen Teilpfad "QRS", der in dieser Reihenfolge eindeutig ist und in dem vor Event "S" keine Verzweigungen auftreten, kann das Netz auch die möglichen Nachfolgerevents nach "S" vorhersagen (siehe Zeile 4 von Tabelle 3.2). Dass die Wahrscheinlichkeit für Event "V" höher ist, liegt daran, dass im nachfolgenden Pfad eine Verzweigung auftritt. Der Pfad, der auf "V" folgt, kommt doppelt so oft im Datensatz vor (achtmal) wie die beiden anderen Pfade nach "T" beziehungsweise "U" (jeweils viermal, siehe Abbildung 2.6). Mathematisch betrachtet sollte die Wahrscheinlichkeit für "V" also bei 50 Prozent und die von "T" und "U" bei jeweils 25 Prozent liegen. Da das Netz bei dieser Eingabe von nur drei Events mehr Vorhersagen als

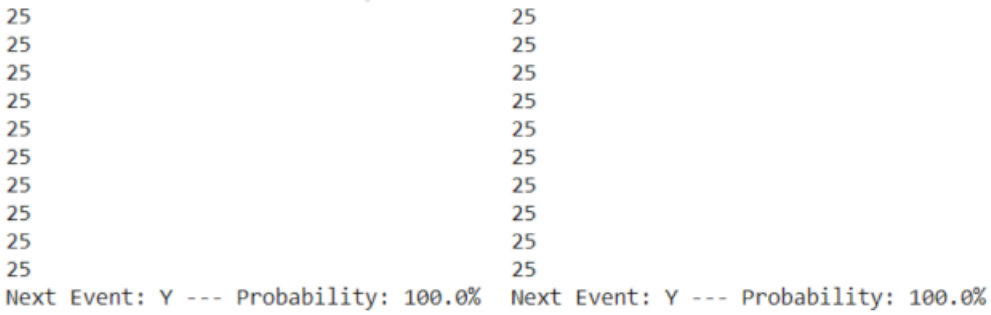

Abbildung 3.10: vorhergesagte Events zu "ABCEHIJLMNQRSTX" in Datensatz 3 (eigene Darstellung)

die drei angezeigten trifft, ergibt die Summe der Wahrscheinlichkeiten nicht 100 Prozent.

3.3.1 Unterschiede bei der Vorhersage

Da die Vorhersage nach der Ausgabe noch mittels einer Multinomialverteilung berechnet wird, können Unterschiede in der Vorhersage zu demselben Event in demselben Datensatz und demselben Netz auftreten.
Im Folgenden werden der Übersichtlichkeit halber nur zehn Vorhersagen getroffen.
Bei eindeutigen Vorhersagen wie der von Pfad "ABCEHIJLMNQRSTX", auf den das Event "Y" folgen muss (siehe Abschnitt 3.3), sind auch die Vorhersagen des Events immer eindeutig. So ist für diesen Pfad die Vorhersage immer "Y" (siehe Abbildung 3.10). Die vorhergesagte 25 entspricht dem Event "Y" (siehe Abbildung 3.12).

Bei Pfaden, in denen zum Beispiel Events nach einer Verzweigung auftreten oder wenn die Länge der Eingabe kürzer ist, kann die Vorhersage durch die Multinomialverteilung etwas abweichen. Für den Pfad "ABCEHIJLMNQRS", auf den die drei Events "V", "T" und "U" folgen können, werden bei mehreren Durchläufen verschiedene Wahrscheinlichkeiten für das nächste Event vorhergesagt. Dabei können keine willkürlichen Events auftreten. Die Vorhersage für diesen Pfad liegt immer bei Werten um 20 bis 22 (siehe Abbildung 3.11), also den Events "T", "U" und "V" (siehe Abbildung 3.12), mit teilweise vorkommenden Ausreißern nach oben oder unten. Es kann dabei zum Beispiel vorkommen, dass als nächstes Event "S" in der Vorhersage steht, da "S" im Vokabular der Wert 19 zugeordnet ist (siehe Abbildungen 3.12 und Zeile 4 von Tabelle 3.2).

3.3.2 Vorhersage nach einzelnen/mehreren Events

Da das Netz so trainiert wird, dass es einen gesamtem Pfad als Eingabe erhält und nur das letzte Event die erwartete Ausgabe ist (vergleiche Abschnitt Training), ist die Genauigkeit der Vorhersage höher, je länger die Eingabe des Nutzers ist. Am genauesten ist das Netz, wenn es die Eingaben erhält, auf die es auch trainiert wurde. Das entspricht einem

```
22                                  20
21                                  22
20                                  22
20                                  22
21                                  21
22                                  22
20                                  20
22                                  20
21                                  20
21                                  21
Next Event: U --- Probability: 40.0%    Next Event: T --- Probability: 40.0%
Next Event: V --- Probability: 30.0%    Next Event: V --- Probability: 40.0%
Next Event: T --- Probability: 30.0%    Next Event: U --- Probability: 20.0%
```

Abbildung 3.11: vorhergesagte Events zu "ABCEHIJLMNQRS" in Datensatz 3 (eigene Darstellung)

```
'\n'    --->    0
'A'     --->    1
'B'     --->    2
'C'     --->    3
'D'     --->    4
'E'     --->    5
'F'     --->    6
'G'     --->    7
'H'     --->    8
'I'     --->    9
'J'     --->   10
'K'     --->   11
'L'     --->   12
'M'     --->   13
'N'     --->   14
'O'     --->   15
'P'     --->   16
'Q'     --->   17
'R'     --->   18
'S'     --->   19
'T'     --->   20
'U'     --->   21
'V'     --->   22
'W'     --->   23
'X'     --->   24
'Y'     --->   25
'Z'     --->   26
```

Abbildung 3.12: Zuordnung der Events zu Zahlenwerten (eigene Darstellung)

kompletten Pfad, bei dem das Netz nur noch das letzte Event vorhersagen muss.
Betrachtet man den Pfad "ABCEHIJLOP" in Datensatz 3 (siehe Abbildung 2.4), der eindeutig als nächstes Event "Q" enthält, sollte die Vorhersage des Netzes 100 Prozent betragen.
In den Zeilen 1 und 2 der Tabelle 3.3 betragen die Wahrscheinlichkeiten für "Q" 99.7 Prozent für "ABCEHIJLOP" und 89.7 Prozent für "JLOP". Für einen Pfad der Länge 10 erreicht das Netz also annähernd eine Genauigkeit von 100 Prozent und auch für die Länge 4 werden noch fast 90 Prozent erreicht.

Tabelle 3.3: unterschiedliche Vorhersagen für Datensatz 3 nach verschiedenen Eingaben (eigene Darstellung)

Eingabe	Vorhersagen		
"A-B-C-E-H-I-J-L-O-P"	"Q", 99.7%	"P", 0.3%	
"J-L-O-P"	"Q", 89.7%	"P", 6.8%	"R", 1.8%
"O-P"	"Q", 59.6%	"P", 13.2%	"R", 7.0%
"P"	"Q", 39.8%	"R", 7.8%	"P", 5.4%
"A-C-E-F-G-H-I-L-M-N-O-P-R-S"	"X", 49.5%	"T", 48.7%	"Y", 0.6%
"O-P-R-S"	"X", 42.3%	"T", 37.2%	"S", 8.3%
"R-S"	"X", 29.4%	"T", 16.4%	"Y", 6.6%
"S"	"X", 17.0%	"T", 16.5%	"Y", 6.6%

Betrachtet man nun kürzere Pfade oder nur ein Event (siehe Tabelle 3.3, Zeile 3 und 4), stellt man eine Abnahme der Vorhersagegenauigkeit fest, je kürzer der Pfad ist. Dennoch liefert ein Pfad, der nur zwei Events enthält, immer noch eine Genauigkeit von 60 Prozent und auch nach einem einzelnen Event kann das nächste noch relativ genau vorhergesagt werden.

In diesem betrachteten Pfad in Datensatz 3 ist das nächste Event eindeutig und es kann somit nur eine richtige Vorhersage geben. Da in Pfaden aber auch Verzweigungen vorhergesagt werden sollen, muss diese Vorhersage gesondert betrachtet werden.

Auf den Pfad "ACEFGHILMNOPRS" in Datensatz 4 (siehe Abbildung 2.5) kann sowohl "X" als auch "T" als nächstes Event folgen. Es wird dieser Pfad als Beispiel betrachtet, da in den Pfaden nach "X" und "T" keine weiteren Verzweigungen auftreten und das Netz somit die gleiche Wahrscheinlichkeit für beide vorhersagen sollte. Im besten Fall beträgt die Wahrscheinlichkeit für beide Events 50 Prozent.
In Zeile 5 der Tabelle 3.3 werden die Wahrscheinlichkeiten für den kompletten Pfad "ACEFGHILMNOPRS" mit der Länge 14 sowie die letzten vier Events des Pfades dargestellt. Bei dem kompletten Pfad wird eine Genauigkeit von 49.5 Prozent für "X" und 48.7 Prozent für "T" erreicht. Das entspricht in etwa den erwarteten 50 Prozent für beide Events.
Mit abnehmender Länge der Pfade nimmt auch in diesem Beispiel die Genauigkeit ab. Bei dem Pfad "OPRS" werden nur noch 42.3 Prozent für "X" und 37.2 Prozent für "T" erzielt.

```
-A-B-xor-C-E-xor-
-A-B-xor-D-F-G-xor-
-A-B-xor-C-E-xor-H-I-xor-K-
-A-B-xor-D-F-G-xor-H-I-xor-K-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-M-N-xor-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-M-N-xor-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-M-N-xor-Q-R-S-xor-T-X-Y-Z-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-M-N-xor-Q-R-S-xor-U-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-M-N-xor-Q-R-S-xor-V-W-xor-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-M-N-xor-Q-R-S-xor-V-W-xor-Y-Z-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-O-P-Q-R-S-xor-V-W-xor-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-O-P-Q-R-S-xor-V-W-xor-Y-Z-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-O-P-Q-R-S-xor-T-X-Y-Z-
-A-B-xor-C-E-xor-H-I-xor-J-L-xor-O-P-Q-R-S-xor-U-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-M-N-xor-Q-R-S-xor-V-W-xor-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-M-N-xor-Q-R-S-xor-T-X-Y-Z-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-M-N-xor-Q-R-S-xor-V-W-xor-Y-Z-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-M-N-xor-Q-R-S-xor-U-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-O-P-Q-R-S-xor-V-W-xor-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-O-P-Q-R-S-xor-V-W-xor-Y-Z-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-O-P-Q-R-S-xor-T-X-Y-Z-
-A-B-xor-D-F-G-xor-H-I-xor-J-L-xor-O-P-Q-R-S-xor-U-
```

Abbildung 3.13: Datensatz 3 mit xor (eigene Darstellung)

Die Genauigkeit für "RS" und "R" nimmt weiter ab (siehe Zeilen 7 und 8 der Tabelle 3.3). Für ein Event "S" liegt sie sogar nur noch bei 17 beziehungsweise 16.4 Prozent für "X" und "T". Wichtig und auffällig ist aber, dass die Wahrscheinlichkeit für beide möglichen Events "X" und "T" in allen Fällen jeweils ungefähr gleich ist. Da beide Pfade nach "S" keine Verzweigungen mehr enthalten, kommen sie im Datensatz gleich oft vor, weil an allen Verzweigungen die gleiche Wahrscheinlichkeit für alle Events gilt. Daraus muss resultieren, dass die Wahrscheinlichkeit für beide Events nach "S" gleich hoch ist, was das Netz auch unabhängig von der Länge der Eingabe so voraussagt.

3.3.3 Möglichkeiten bei der Vorhersage

Mit dem erstellten Netz ist es möglich, jede Form von Flussdiagrammen vorherzusagen, da es jede Abfolge von Events akzeptiert.

Im Folgenden wird dies anhand von Verzweigungen verdeutlicht. Diese können vorhergesagt werden, indem man sie im Datensatz einfügt (siehe Abbildung 3.13). Dabei ist für das Netz zu beachten, dass jede Verzweigung hier gleich benannt und somit als ein Event betrachtet wird. Das erschwert die richtige Vorhersage, weil die Verzweigungen nicht eindeutig sind. Um dies zu umgehen, könnte jede Verzweigung separat benannt werden (z. B. "xor1", "xor2").

Auch ohne eindeutige Benennung liefert das Netz Vorhersagen. In Zeile 2 von Tabelle 3.4 sagt das Netz nur Nachfolger von "xor" vorher, ohne zu wissen, auf welches "xor" man sich bezieht. Dabei können alle getroffenen Vorhersagen ("J", "H", "O") nach Verzweigungen vorkommen (siehe Abbildung 2.4).

Tabelle 3.4: Vorhersagen mit und ohne xor für Datensatz 3 (eigene Darstellung)

Eingabe	Vorhersagen		
"I"	"J", 49.2%	"L", 8.0%	"K", 8.0%
"xor"	"J", 14.4%	"H", 12.6%	"O", 6.9%
"I-xor"	"J", 53.3%	"H", 9.8%	"O", 6.0%
"A-B-xor"	"C", 41.4%	"D", 39.6%	"B", 5.3%
"A-B"	"C", 47.9%	"D", 39.6%	"B", 5.3%

Eindeutiger werden die Vorhersagen, wenn man dem Netz auch ein vorhergehendes Event oder einen Teilpfad übergibt. Bei einen vorhergehenden Event (siehe Zeile 3 von Tabelle 3.4) wird das wahrscheinlichste nachfolgende Event "J" eindeutig vorhergesagt und es besteht kaum ein Unterschied zur Vorhersage ohne "xor" (vergleiche dazu Zeile 1 von Tabelle 3.4). In beiden Fällen wird "J" als wahrscheinlichster Nachfolger vorhergesagt.
Auch die Angabe eines Teilpfades steigert die Genauigkeit (siehe Zeile 4 von Tabelle 3.4). Die Events "C" und "D" sind die möglichen Nachfolger nach "B" (siehe Abbildung 2.4). Auch hier sind kaum Unterschiede zur Vorhersage ohne "xor" zu sehen (vergleiche Zeile 5 von Tabelle 3.4) und in beiden Fällen werden "C" und "D" als in etwa gleich wahrscheinlich vorhergesagt.

Es wird jedoch deutlich, dass einheitlich benannte Verzweigungen nicht vorhergesagt werden. Das Netz sagt nur das nachfolgende Event zur Eingabe voraus, nicht aber "xor".
Bei einer eindeutigen Benennung der Verzweigungen (in diesem Fall insbesondere "xor", siehe Abbildung 3.14) kann das Netz diese auch vorhersagen (siehe Tabelle 3.5).

Tabelle 3.5: Vorhersagen mit und ohne eindeutigem xor für Datensatz 3 (eigene Darstellung)

Eingabe	Vorhersagen		
"I"	"xor4", 70.3%	"J", 5.2%	"K", 4.1%
"xor4"	"J", 52.6%	"K", 10.3%	"L", 4.0%
"I-xor4"	"J", 85.9%	"K", 6.0%	"xor4", 3.8%
"A-B-xor1"	"C", 52.6%	"D", 42.8%	"xor1", 2.1%
"A-B"	"xor1", 92.8%	"B", 2.9%	"C", 1.9%

Dabei werden Verzweigungen wie ein Event behandelt. Durch die eindeutige Benennung ist eine Vorhersage der Verzweigung und auch dessen Nachfolger möglich.

Zusätzlich ist es möglich, auch aus Zeichenketten bestehende Prozesse (siehe Abbildungen 3.15 und 3.16 in dem Netz zu verwenden. Dabei wird jedem Event (getrennt durch "-") ein fester Wert zugeordnet und so das Vokabular erstellt.

Das Netz ist in der Lage, auch dafür passende Vorhersagen zu treffen (siehe Abbildung 3.17).

```
-A-B-xor1-C-E-xor2-
-A-B-xor1-D-F-G-xor3-
-A-B-xor1-C-E-xor2-H-I-xor4-K-
-A-B-xor1-D-F-G-xor3-H-I-xor4-K-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-M-N-xor6-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-M-N-xor6-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-M-N-xor6-Q-R-S-xor7-T-X-Y-Z-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-M-N-xor6-Q-R-S-xor7-U-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-M-N-xor6-Q-R-S-xor7-V-W-xor8-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-M-N-xor6-Q-R-S-xor7-V-W-xor8-Y-Z-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-O-P-Q-R-S-xor7-V-W-xor8-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-O-P-Q-R-S-xor7-V-W-xor8-Y-Z-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-O-P-Q-R-S-xor7-T-X-Y-Z-
-A-B-xor1-C-E-xor2-H-I-xor4-J-L-xor5-O-P-Q-R-S-xor7-U-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-M-N-xor6-Q-R-S-xor7-V-W-xor8-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-M-N-xor6-Q-R-S-xor7-T-X-Y-Z-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-M-N-xor6-Q-R-S-xor7-V-W-xor8-Y-Z-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-M-N-xor6-Q-R-S-xor7-U-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-O-P-Q-R-S-xor7-V-W-xor8-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-O-P-Q-R-S-xor7-V-W-xor8-Y-Z-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-O-P-Q-R-S-xor7-T-X-Y-Z-
-A-B-xor1-D-F-G-xor3-H-I-xor4-J-L-xor5-O-P-Q-R-S-xor7-U-
```

Abbildung 3.14: Datensatz 3 mit eindeutigen "xor" (eigene Darstellung)

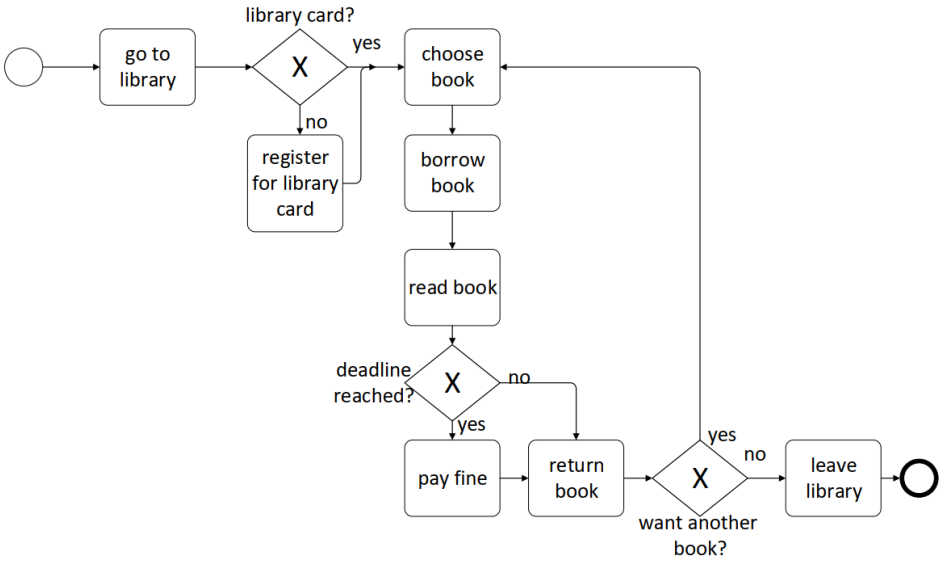

Abbildung 3.15: BPMN zu Datensatz 6 (eigene Darstellung)

```
-go to library-choose book-borrow book-read book-pay fine-return book-choose book-
-go to library-register for library card-choose book-borrow book-read book-pay fine-return book-choose book-
-go to library-choose book-borrow book-read book-return book-choose book-
-go to library-register for library card-choose book-borrow book-read book-return book-choose book-
-go to library-choose book-borrow book-read book-pay fine-return book-leave library-
-go to library-register for library card-choose book-borrow book-read book-pay fine-return book-leave library-
-go to library-choose book-borrow book-read book-return book-leave library-
-go to library-register for library card-choose book-borrow book-read book-return book-leave library-
```

Abbildung 3.16: Prozesse in Datensatz 6 (eigene Darstellung)

```
Prediction for: go to library

List of predictions
Next Event: choose book --- Probability: 39.00%
Next Event: register for library card --- Probability: 27.10%
Next Event: borrow book --- Probability: 10.60%
```

Abbildung 3.17: Vorhersage für "go to library" in Datensatz 6 (eigene Darstellung)

```
import Prediction as p

p.makePredictions('Dataset3', 'default_variables.json')
```

Abbildung 3.18: Import in ein anderes Pythonprogramm (eigene Darstellung)

3.4 Verwendung des Netzes in anderen Programmen

Es ist möglich, das vorliegende Programm in jedem anderen Pythonprogramm zu verwenden. Dazu muss dieses nur in das jeweilige Programm importiert werden (siehe Abbildung 3.18) und kann dann zur Vorhersage verwendet werden.
Dafür ist eine Integration von TensorFlow nötig. Um das Training zu beschleunigen, können NVIDIA Grafikarten verwendet werden (siehe Abschnitt Nutzung der Graphics Processing Unit (GPU)).

Dabei liegen sowohl die Datensätze, die Datei für die Variablen, die gespeicherten Checkpoints, sowie das vorliegende Programm für das Netz und das ausgeführte Programm selbst im selben Verzeichnis.

Der Ordner "checkpoints" muss nicht verändert werden, da hier die Parameter des Netzes beim Training gespeichert werden. Im Ordner Datasets liegen die jeweiligen Trainingsdatensätze mit ".txt"-Endung. Dem Programm wird der Name des Datensatzes (hier "Dataset3", siehe Abbildung 3.18), der auch dem Dateinamen entspricht (hier "Dataset3.txt"), sowie die zu verwendende Datei für die Variablen übergeben.
In variables liegen zwei Dateien, "custom_variables" und "default_variables". Die Datei "default_variables" enthält die für diese Arbeit verwendeten Parameter.
Hier können alle für das Netz relevanten Parameter sowie solche für das Training eingestellt werden (siehe Tabelle 3.6).

Tabelle 3.6: Parameter, eigene Darstellung

Parameter	Funktion
splitChar	Zeichen, das Events im Datensatz trennt
showTrainingDetails	Angabe, ob Details während des Trainings gezeigt werden
saveCheckpoint	Angabe, ob die Parameter im Training gespeichert werden
train	Angabe, ob trainiert oder nur vorhergesagt wird
use_treshold	Angabe, ob ein Abbruchkriterium verwendet wird
showPlotting	Angabe, ob eine grafische Darstellung nach dem Training gezeigt wird
learning_rate	
optimizer_string	
seq_length	
batch_size	
buffer_size	Parameter für das Netz
embedding_dim	
units	
epochs	
loss_treshold	
saveEveryEpoch	Epochen, in denen gespeichert wird (Standard: jede Epoche)
num_predictions	Anzahl der getroffenen Vorhersagen
event_list_len	Anzahl der ausgegebenen Vorhersagen

Die Ausführung des Programms wird über die Eingabe "exit" oder die Tastenkombination "Strg" + "C" beendet. Dies ist nötig, da nicht zwei Instanzen auf der gleichen Grafikkarte laufen können. Falls dies versucht wird, bekommt man eine entsprechende Fehlermeldung.

4 Auswertung und Fazit

Ziel der vorgelegten Arbeit war es, ein Netz zu erstellen, das Events in Prozessen anhand von vorhandenen Datensätzen vorhersagen kann. Dazu wird es mit einem Datensatz bestehend aus beliebig vielen Prozessen trainiert und soll dann anhand dieses Trainings Vorhersagen treffen.

Das Netz liefert in allen getesteten Szenarien eine eindeutige und richtige Vorhersage und ist somit sehr gut als Empfehlungsdienst geeignet.
Dabei fällt auf, dass aufgrund des Trainings längere Pfade zur Vorhersage eine höhere Genauigkeit liefern, aber auch für einzelne Events das nächste Event sicher vorhergesagt werden kann. Außerdem ist das Netz für jede mögliche Abfolge von Events (z. B. Flussdiagramme, BPMN) nutzbar, unabhängig davon, ob der Prozess Verzweigungen enthält oder nicht. Das macht es in vielen Bereichen variabel einsetzbar.

In der weiteren Entwicklung wäre eine direkte Einbindung von Modellen (z. B. BPMN) und deren Simulation sinnvoll, falls keine konkreten Datensätze vorliegen. Damit könnten Datensätze direkt aus einem Modell extrahiert und in das Netz übertragen werden.
Das Netz ist in allen anderen Pythonprogrammen nutzbar (siehe Abschnitt Verwendung des Netzes in anderen Programmen). Somit ist keine Schnittstelle nötig, falls Python genutzt wird. Zudem wäre auch die Erstellung einer ausführbaren Datei (.exe) aus dem Pythonprogramm (.py) möglich (siehe hierzu *pyinstaller*, womit ausführbare Dateien erstellt werden können).

Abschließend kann festgehalten werden, dass rekurrente neuronale Netze (hier LSTM) auch für die Prozessvorhersage genutzt werden können.

Literaturverzeichnis

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I. J., Harp, A., Irving, G., Isard, M., Jia, Y., Józefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D. G., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P. A., Vanhoucke, V., Vasudevan, V., Viégas, F. B., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y. & Zheng, X. (2016), 'Tensorflow: Large-scale machine learning on heterogeneous distributed systems', *CoRR* .

Adomavicius, G. & Tuzhilin, A. (2005), 'Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions', *IEEE Transactions on Knowledge and Data Engineering* **17**(6), 734–749.

Carbune, V., Gonnet, P., Deselaers, T., Rowley, H., Daryin, A., Calvo, M., Wang, L.-L., Keysers, D., Feuz, S. & Gervais, P. (2019), 'Fast multi-language lstm-based online handwriting recognition'.

CuDNN (2019). aberufen am 26.03.2019.
URL: *https://developer.nvidia.com/cudnn*

Evermann, J., Rehse, J.-R. & Fettke, P. (2017), 'Predicting process behaviour using deep learning'.

Gers, F. & Jürgen, S. (2001), 'Lstm recurrent networks learn simple context-free and context-sensitive languages'.

Hochreiter, S. & Schmidhuber, J. (1997), 'Long short-term memory', *Neural computation* **9**, 1735–80.

Jordan, M. I. & Mitchell, T. M. (2015), 'Machine learning: Trends, perspectives, and prospects', *Science* **349**(6245), 255–260.

Karras, T., Laine, S. & Aila, T. (2019), 'A style-based generator architecture for generative adversarial networks'.

LeCun, Y., Bengio, Y. & Hinton, G. (2015), 'Deep learning', *Nature* **521**, 436–44.

Nickolls, J. & Dally, W. J. (2010), 'The gpu computing era', *IEEE Micro* **30**(2), 56–69.

Schmidhuber, J. (2015), 'Deep learning in neural networks: An overview', *Neural Networks* **61**, 85 – 117.

Zhang, G., Patuwo, B. E. & Hu, M. Y. (1998), 'Forecasting with artificial neural networks:: The state of the art', *International Journal of Forecasting* **14**(1), 35 – 62.