Applications in Data Science

Hausarbeit

Name: Lukas Dech

Zenturie: W18b Matr.-Nr.: 9376

Datum: 18.11.2021

Gliederung

Abbildungsverzeichnis

1. Abstract

Im vorliegenden Notebook wird ein Model erstellt, welches das Ziel hat, Wörter und Sätze anhand von im Bundestag gehaltenen Reden, zu generieren.
Das Problem, welches behandelt wird, stellt ein Klassifizierungsproblem aus dem Bereich des Natural Language Processing dar.

Durch das Herunterbrechen der Daten auf spezifische Politiker oder einzelnen Parteien können bei erfolgreicher Generierung,
Textstrukturierung sowie Meinung zu einem spezifischen Thema herausgefiltert generiert werden.
Weiterhin können die erfolgreich generierten Satz Strukturen genutzt werden, um als Vorlage für die Struktur zukünftig gehaltener reden zudienen.

2. Einführung

Mit der steigernden Relevanz von Daten und deren Verarbeitung im wirtschaftlichen Kontext, wird deren Analyse einen immer größer werdenden Stellenwert zugeschrieben. Der Bereich Data-Science erfreut sich damit in den letzten Jahren einer immer größeren werdenden Popularität. Vermehrt werden Prozesse angepasst, deren Gestaltung bisher auf Erfahrungswerten und Beobachtungen beruhten.

Mit der Möglichkeit immer größer werdende Datensätze zu speichern und auch Quantitative Datenmengen wie Bild, Ton und Schrift auszuwerten, werden immer mehr Tools welche auf dem Einsatz von Künstlicher Intelligenz beruhen, im Alltag eingesetzt (Ipsos, 2020). Besonders ist dieses Phänomen im Bereich des Natural Language Processing zu beobachten, da Chatbots und Sprachassistenten schon langen keine Seltenheit mehr darstellen.

Die voranschreitende Entwicklung in diesen Bereich diente auch als Anstoß für die vorliegende Arbeit.
Mit dem Hintergrund das, dass Erstellen einer qualitativen Rede unter den Gesichtspunkten der Strukturierung und den inhaltlichen Aspekten einen großen Aufwand mit sich ziehen kann, ist die Idee, dass die Strukturierung von Reden mithilfe einer Vorlage der automatisch generierter Reden vereinfacht werden kann.
die Methodik zum Erstellen des Notebooks orientiert sich an einen bereits existierende Vorgehensmodelle aus dem Bereich Data-Science, welches im Kapitel 4 vorgestellt wird.

3. Aufgabenstellung

Das Ziel, welches in der Auftragserklärung definiert worden ist, ist es ein Model mithilfe von im Bundestag gehaltenen Reden zu trainieren, sodass es schlussendlich in der Lage ist, eine eigene Rede mit einer grammatikalisch korrekten Satzstruktur und verständlichen Inhalt generieren zu können.

Das Problem stellt ein Klassifikation Problem dar, da anhand der bereits verfassten Reden das nächste Wort vorhergesagt wird. Jedes Wort wird als eigene Klasse dargestellt. Durch das mehrmalige Wiederholen dieses Prozesses kann ein Zusammenhängender Text erstellt werden. Nach der erfolgreichen Generierung der Texte kann die Genauigkeit des Models mithilfe verschiedener Metriken validiert werden. Außerdem lässt sich mithilfe der generieren Texte eine Aussage darüber treffen ob das Model imstande ist, ein Grammatikalisch Korrekte sowie inhaltlich Sinnvolle Rede zu generieren.

4. Methodik

Das Vorgehen zur Erstellung des Notebooks und Bearbeitung der Fragestellung orientiert sich an dem DASC-PM (Data Science Process Modell). Der DASC-PM wurde von einer Arbeitsgruppe der Nordakademie entworfen welches als Ziel hat Klarheit darüber zu schaffen, inwiefern Data-Science als Disziplin von anderen abzugrenzen ist. Darauf aufbauend welche Besonderheiten in Data-Science Projekten auftauchen und welche Kompetenzen im Team vorhanden sein müssen, damit die saubere Durchführung des Projekts gewährleistet ist.(Neuhaus, et al. 2020)

Der DASC PM ist in folgende vier Arbeitsphasen unterteilt:

- Daten
- Analyse
- Nutzbarmachung
- Nutzung

Im Vorgehensmodell wird weiterhin beschrieben, dass die Arbeitsdomäne, in der das Projekt durchgeführt wird, übergreifend in jeder Phase betrachtet werden muss, da in den Arbeitsphasen Domänenspezifische Grenzen gesetzt sein können.
Außerdem sollte das Projekt unter Berücksichtigung der vorliegenden IT-Infrastruktur durchgeführt werden. Die IT-Infrastruktur sollte vor Projektbeginn neu bewertet werden, um auszuschließen, dass das Projekt in einer Arbeitsphase, durch die nicht gegebene Infrastruktur abgebrochen werden muss.(Neuhaus, et al. 2020)

Eine Umfangreiche Beschreibung des Vorgehensmodells ist hier finden :
https://www.nordakademie.de/sites/default/files/2020-02/20200220_DASC-PM%20%28002%29.pdf

Zur Erstellung des Notebooks werden die einzelnen Phasen jeweils beschrieben und anschließend abgearbeitet.

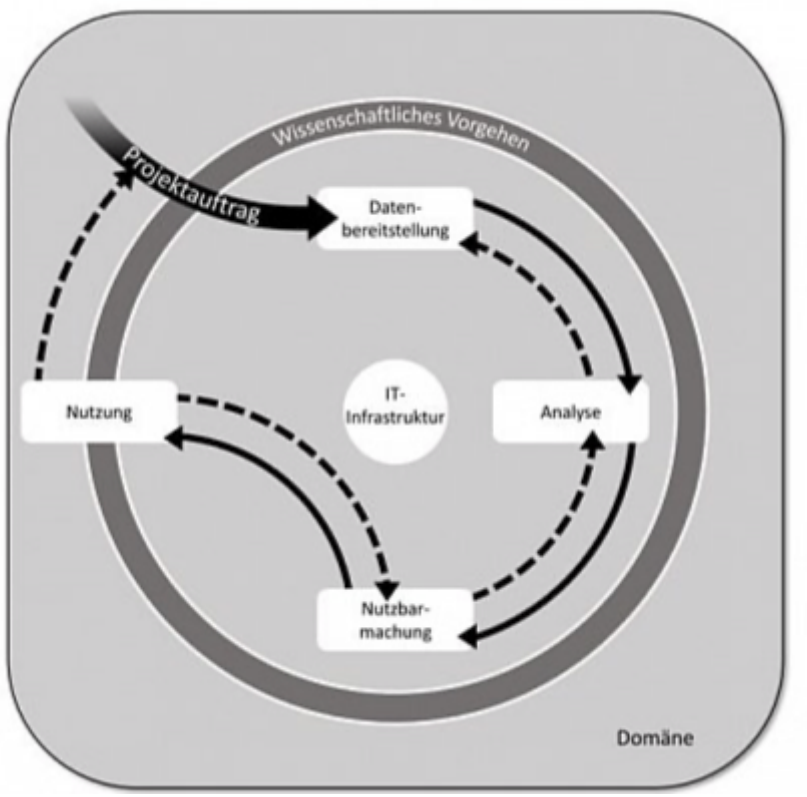


Abbildung 1 DASC_PM (Neuhaus, et al. 2020, S.23)

5. Artefakt

In [2]:

```
import nltk
from nltk import word_tokenize
nltk.download("punkt")
from sklearn.feature_extraction.text import CountVectorizer
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras import layers
from tensorflow.keras import models
import numpy as np
import pandas as pd
import keras.utils
from tensorflow.keras.utils import to_categorical
from keras.models import Sequential
from keras.layers import Dense, LSTM, Dropout, Embedding
import tqdm
import pickle
import matplotlib.pyplot as plt
```

...

In [3]:

```
SEQUENCE_LENGTH = 100 # Länge der trainierten Sequence (Anzahl Wörter Pro Beispiel)
EMBEDDING_SIZE = 100  # Dimension des verwendeten Embeddings
BATCH_SIZE = 128 # Größe des trainierten Batches
EPOCHS = 10 # Anzahl der trianierten Epochen
CV_MAX_FEATURES = 5000 # Größe des Vokabluar
```

5.1 Daten

In der ersten Arbeitsphase werden alles spezifische Aufgaben, die zur Vorbereitung für die eigentliche Analyse dienen zusammengefasst.
Damit ist sowohl die Daten Akquisition verbunden also auch die Aufbereitung des Datensatzes und das Datenmanagement. Außerdem wird schon im Schritt der Daten Akquisition eine erste explorative Datenanalyse durchgeführt, anhand der Ergebnisse lassen sich dann nötige Schritte für die Aufbereitung des Datensatzes definieren.

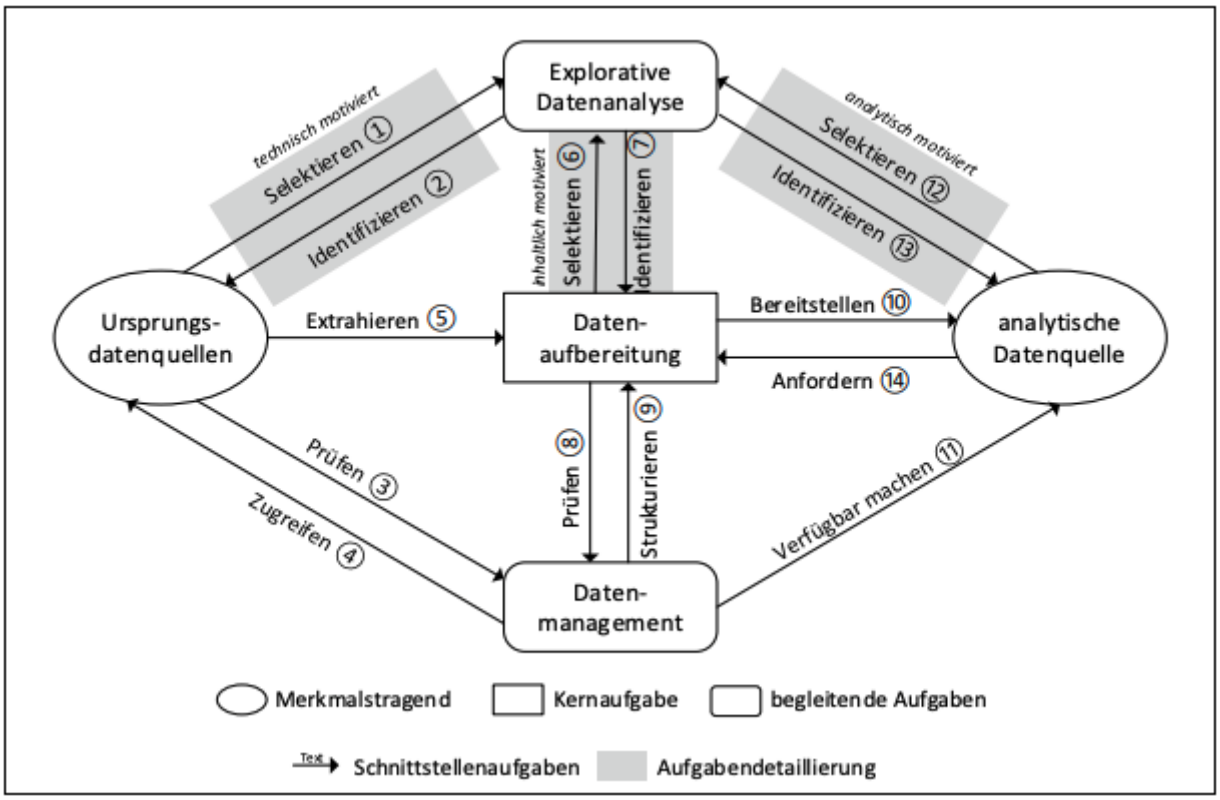


Abbildung 2 Schlüsselbereich Daten (Neuhaus, et al. 2020, S.26)

Urpsrungsdatenquelle

Die Analyse der Ursprungsdatenquelle umfasst die Betrachtung von Faktoren, welche das Reibungslose Arbeiten im Projekt ermöglichen.
Vor allem der Beschaffungs- und Verwaltungsaufwand spielen dabei eine übergeordnete Rolle. Allerdings wird auch die Qualität des vorliegenden Datensatzes bewertet. Da die Erstellung des Notebooks in einzeln Arbeit erfolgt und damit der Zugriff auf die Daten nicht eingeschränkt ist und der Beschaffungs- und Verwaltungsaufwand gering ist, wird im folgenden auf diese Punkte nicht weiter eingegangen.

Wie bereits in der Fragestellung dargestellt soll das Ziel des Notebooks sein, ein Model zu erarbeiten welches es ermöglicht auf Grundlage von Politischen Reden, Texte zu generieren. Als Datengrundlage wurden Plenarprotokolle gewählt.
Plenarprotokolle werden in jeder Sitzung des Bundestages von 3 Stenografen angefertigt, die Protokolle enthalten dabei neben den Gehaltenen Reden auch Zurufe, Fragen sowie die Tagesordnung.
Die Protokolle sind als TXT Dateien frei unter folgender URL als Download verfügbar:
(https://www.bundestag.de/dokumente/protokolle/plenarprotokolle

Da die Daten unstrukturiert im TXT vorliegen, besteht ein Teil des Preprocessing darin die Daten in eine Form zu bringen, die es ermöglicht nur die gehaltenen Reden mit den vorhandenen Metadaten wie Name des Politikers oder Partei des Politikers zu verarbeiten.
Dieser Schritte wurden bereits im folgenden Repository bearbeitet:
https://github.com/Datenschule/offenesparlament-data

Hierbei wurden die Plenarprotokolle, die zu einzelnen Sitzungen angefertigt worden sind, bereits verarbeitet und in ein CSV Format überführt. Dies ermöglicht es nun die Daten in den folgenden Schritten für ein geeignetes Analyseverfahren vorzubereiten.

In [4]:

```
dfs = []
#Einlesen der Plenarprotokolle
for i in range(165,175):
    protokoll_path = f"../Artefakt_LukasDech_9376/06_TXT/{i}.csv"
    dfs.append(pd.read_csv(protokoll_path, encoding = 'unicode_escape',
                           sep = ",", error_bad_lines = False))

#Zusammenfügen der Einzelnen Plenarprotokolle als ein Pandas Dataframe
speeches_concat = pd.concat(dfs, ignore_index = True)
```

Exploartive Datenanalyse

Das Ziel der Explorativen Datenanalyse ist es ein besseres inhaltliches Verständnis über den vorliegenden Datensatz aufzubauen.
Mithilfe dessen kann geklärt werden, ob die Daten in ausreichender Menge zur Verfügung stehen und inwiefern die Qualität und Aussagekraft der Daten zur Beantwortung der Fragestellung genügt. Die Explorative Datenanalyse besteht dabei aus der Datenvisualisierung und der Anwendungen von statistischen Methoden.

In [5]:

```
#Allgemeine Information über die Rohdaten
speeches_concat.info()
```

```
Data columns (total 15 columns):
 #   Column           Non-Null Count  Dtype
---  ------           --------------  -----
 0   Unnamed: 0       13313 non-null  int64
 1   id               13313 non-null  int64
 2   profile_url      8659 non-null   object
 3   sequence         13313 non-null  int64
 4   sitzung          13313 non-null  int64
 5   speaker          9346 non-null   object
 6   speaker_cleaned  9345 non-null   object
 7   speaker_fp       9345 non-null   object
 8   speaker_key      8659 non-null   float64
 9   speaker_party    7588 non-null   object
 10  text             13313 non-null  object
 11  top              13313 non-null  object
 12  top_id           13313 non-null  int64
 13  type             13313 non-null  object
 14  wahlperiode      13313 non-null  int64
dtypes: float64(1), int64(6), object(8)
memory usage: 1.5+ MB
```

In [16]:

```
#Ausgabe der ersten 10 Zeilen der Rohdaten
speeches_concat.head(10)
```

Out[16]:

	Unnamed: 0	id	profile_url	sequence	sitzung	spe
0	0	542008	https://www.abgeordnetenwatch.de/profile/norbe...	0	165	PrÃ¤si Dr. No Lam
1	1	542009	NaN	1	165	
2	2	542010	https://www.abgeordnetenwatch.de/profile/norbe...	2	165	PrÃ¤si Dr. No Lam
3	3	542011	NaN	3	165	
4	4	542012	NaN	4	165	SteinbrÃ (S
5	5	542013	NaN	5	165	
6	6	542014	NaN	6	165	SteinbrÃ (S
7	7	542015	NaN	7	165	
8	8	542016	NaN	8	165	SteinbrÃ (S
9	9	542017	NaN	9	165	

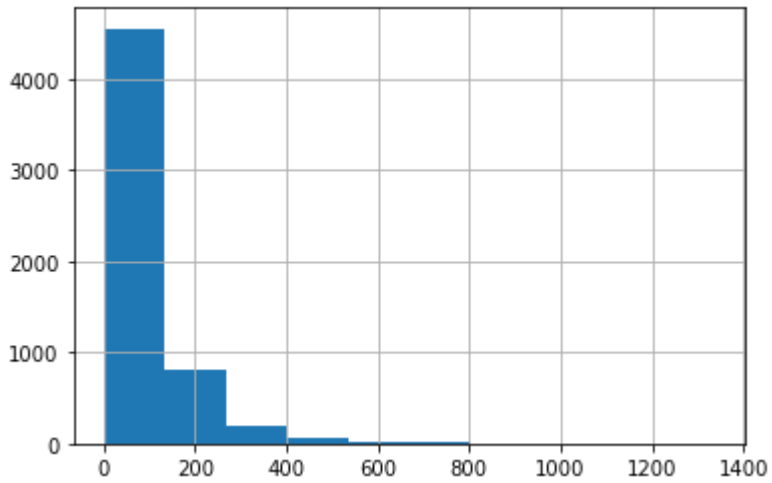
In [7]:

```
#Visualiserung der Reden
lens = speeches_concat[speeches_concat["type"] == "speech"]["text"]
                      .str.split().apply(lambda x: len(x))
print(lens.describe())
lens.hist()
```

```
count    5641.000000
mean       88.341783
std        95.532146
min         1.000000
25%        28.000000
50%        59.000000
75%       113.000000
max      1334.000000
Name: text, dtype: float64
```

Out[7]:

```
<AxesSubplot:>
```



Datenaufbereitung

Das Grundlegende Ziel der Datenaufbereitung ist es, die Daten in eine Form zubringen das sich diese für ein ausgewähltes Analyseverfahren eignen.
Weiterhin werden mithilfe der Ergebnisse aus der Explorativen Datenanalyse werden weitere Änderung am Datensatz vorgenommen, um die Qualität der Daten zu steigern und um zu verhindern das Unstimmigkeiten im Datensatz für eine Verzerrung des Analyseergebnisses sorgen.

Daten Filtern

Wie in der Explorativen Datenanalyse dargestellt, ist zuerkennen das sich im Datensatz auch Beiträge befinden, die keine Politische Rede sind.
Schlussfolgernd werden alle Zeilen, welche nicht dem Typ Speech sind oder keiner Partei zugeordnet werden können, rausgefiltert.

Außerdem werden nicht relevante Spalten entfernt. Für die weitere Verarbeitung werden nur die Texte benötigt, anhand der vorhandenen Metadaten wird der Datensatz außerdem auf nur eine Partei beschränkt, in diesem Fall die CDU / CSU, da dies die Partei mit den mitgehaltenen Reden im Datensatz ist.

In [9]:

```
#Filtern der Daten, sodass nur notwendige Daten erhalten bleiben
filtered_column_speeches = speeches_concat[["speaker_party","text","top","type"]]
filtered_row_speeches = filtered_column_speeches[
                        (filtered_column_speeches["speaker_party"].notnull()) &
                        (filtered_column_speeches["type"] == "speech") &
                        (filtered_column_speeches["speaker_party"] == "cducsu")]
```

In [7]:

```
#Anzahl an verschiedenen Reden im gesäuberten Datensatz
filtered_row_speeches.speaker_party.value_counts()
```

Out[7]:

```
cducsu    1772
Name: speaker_party, dtype: int64
```

In [17]:

```
#Ersten 3 Zeilen des gesäuberten Datensatzes
filtered_row_speeches.head(3)
```

Out[17]:

	speaker_party	text	top	type
61	cducsu	Herr PrÃ¤sident! Meine sehr geehrten Damen und...	TOP 19 Weiterentwicklung der transatlantischen...	speech
63	cducsu	Dann wÃ¤re auch das in die Debatte eingebracht...	TOP 19 Weiterentwicklung der transatlantischen...	speech
65	cducsu	Das stimmt.	TOP 19 Weiterentwicklung der transatlantischen...	speech

Vorbereitung Daten
Nachdem alle Reden in einen zusammenhängenden Text transformiert worden sind, kann mithilfe eines Sets ausgewertet werden wie viele verschiedene Wörter sich im Datensatz befinden.

Da Es sich bei einen NLP Problem um ein Klassifizierungsproblem handelt und Wörter untereinander Nominalskaliert werden besteht die Notwendigkeit die Daten zu One-Hot-Encoden, da das Modell ansonsten nicht vorhandene Zusammenhänge lernt.
Beim One-Hot-Encoding wird jedes Wort als Vektor dargestellt, die Länge des Vektors entspricht denn insgesamt zu lernenden Wörtern. Daraus resultiert das die Verarbeitungszeit, die das Modell zum Trainieren benötigt stark von der Anzahl an zu trainierende Wörter abhängt.

Der Pool an verschiedenen Wörtern, welche im Datensatz vorhanden sind, wird auf die meist genutzten eingeschränkt, um die Trainingsdauer zu verringern und möglichst wenig Qualitätsverluste zuhaben.
Mithilfe eine word_tokenizers welcher den Text in einzelne Wörter zerlegt und einen Countvectorizer welcher es ermöglich den Datensatz auf die an den häufigsten genutzten Worten zu reduzieren wird der Datensatz eingeschränkt.

Beding durch die Komplexität und die damit verbundene Dauer, die für das Training des Models benötigt wird. Ist die Maximal Anzahl an zu trainierenden Wörtern auf 5000 beschränkt, da dies dem aktiv genutzten Vokabular eines Muttersprachlers entspricht (Römer 2005).

In [10]:
```
#Alle Reden als zusammenhängenden Text darstellen
all_speeches_series = pd.Series(list(filtered_row_speeches["text"]))
all_speeches_text = all_speeches_series.str.cat(sep=' ')

#Text in einezelne Wörter zerlegen
tokens_speeches = word_tokenize(all_speeches_text)

#Wie viele verschiedene Wörter sind vorhanden
print(len(set(tokens_speeches)))

#max_features = es werden sich nur die X Häufigsten Wörter angeschaut
cv = CountVectorizer(max_features= CV_MAX_FEATURES,
                     lowercase= False, token_pattern ="(.*)")
cv.fit(tokens_speeches)

#features = X Häufigsten Wörter
features = cv.get_feature_names()
```

```
19184
```

Input vorbereiten
Neuronale Netze können nur Zahlen als Input und Output verarbeiten.
Schlussfolgernd ist es notwendig jedem Wort eine Zahl zuzuordnen, um die Verarbeitung im Model zu ermöglichen und die umgekehrte Codierung von Zahl zu Wort ebenfalls abzuspeichern, damit es möglich ist die Vorhersage des Modells wieder in Schrift umzuwandeln.

In [10]:

```
word_to_int = {}
int_to_word = {}

#jedes wort wird einer zahl zugeordnet
for i in range(0, len(features)):
    word = features[i]
    word_to_int[word] = i
    int_to_word[i] = word

#Alle tokens wird in eine zahl umgewandelt allerdings nur die die auch zu denn 5000
tokens_transformed = [word_to_int[word] for word
                      in tokens_speeches if word in word_to_int]
```

In [11]:

```
with open("word_to_int.speechGeneration.pickle", "wb") as file:
    pickle.dump(word_to_int, file)

with open("int_to_word.speechGeneration.pickle", "wb") as file:
    pickle.dump(int_to_word, file)
```

Erstellung Trainingsdatensatz
Anschließend werden die Daten in eine Form gebracht mit welchen das Training möglich ist. Da die Vorhersage des nächsten Wortes auf Grundlage des Kontextes erfolgen soll, sind die Inputvariablen eine Sequenz von Wörtern welche aufeinanderfolgend im Text vorkommen. Die Output Variable entspricht dem Wort, welches nach der Sequenz folgt.

Das Ziel ist es damit dem Modell die Möglichkeit zugeben die Vorhersage anhand eines oder mehrere Sätze treffen zukönnen.

Die Output Variablen werden One-Hot-encoded wodurch die Vorhersage von unabhängigen Klassen ermöglicht wird, wobei jedes Wort eine Klasse darstellt. Das One-Hot-Encoding bei den Input Variablen ist nicht notwendig, da die erste Schicht des Netzes ein Embedding Layer ist welches das One-Hot-Encoding übernimmt.

Testdatensatz
Eine weiter Unterteilung zwischen Trainings- und Testdaten wird an dieser Stelle noch nicht vorgenommen, da nicht Alle Plenarprotokolle eingelesen worden sind und die nicht eingelesen Protokolle bei der Evaluierung der Ergebnisse als Testdatensatz dienen werden.

In [12]:

```
X = []
y = []
# x = vektor an zahlen (satz)
# y = nächstes wort (One-Hot-Encoding)
for i in range(0, len(tokens_transformed) - SEQUENCE_LENGTH):
    X.append(tokens_transformed[i:i + SEQUENCE_LENGTH])
    y.append(tokens_transformed[i+SEQUENCE_LENGTH])
X = np.array(X)
y = np.array(to_categorical(y, num_classes = cv.max_features))
```

5.2 Analyse

Alle Vorgestellten Architekturen wurden bereits trainiert und mit als Prüfungsleistung abgeben. Die trainierten Modele werden im Kapitel der Ergebnis Evaluation vorgestellt.

Nachdem der Datensatz aufbereitet ist, kann angepasst an den Anforderungen, ein geeignetes Analyseverfahren ausgewählt werden. Sobald ein geeignetes Verfahren gefunden worden ist, kann der Datensatz entsprechend verarbeitet werden und die Ergebnisse evaluiert werden. Im vorliegenden Notebook wurde sich aufgrund der Komplexen Aufgabenstellung für Deep-Learning als Analysemethode entscheiden.

Im Folgenden wird zum einen die Grundlegende Theorie zu den genutzten Architekturen und Modellen vorgestellt, sowie der Code zum Trainieren der Modele erstellt. Anschließend werden die Ergebnisse anhand verschiedenere Metriken bewertet.

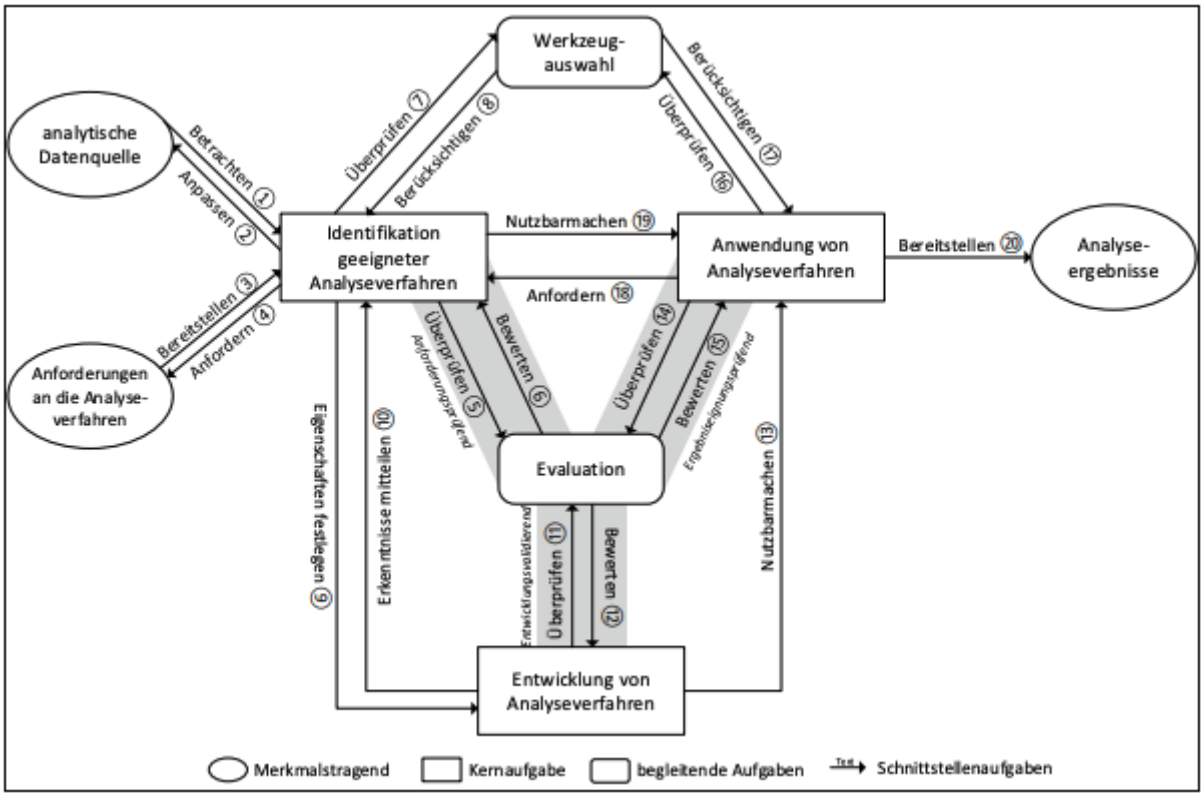


Abbildung 3 Schlüsselbereich Analyse (Neuhaus, et al. 2020, S.37)

Aktivierungsfunktion

Sigmoid
die Sigmoid-Funktion besitzt eine große Signalverstärkung im zentralen Bereich und nur eine kleine Signalverstärkung an den äußeren Bereichen, dies führt dazu das die Werte vermehrt gegen 1 oder 0 tangieren wodurch erkannte Eigenschaften verstärkt werden (Chigozie, Enyinna, Nwankpa 2020).

Softmax
Die Softmax definiert anhand der Inputvariablen für jeden Output eine Wahrscheinlichkeit. die Summe der definierten Wahrscheinlichkeiten ist 1, da die Softmax Aktivierungsfunktion in den Beispielen, als Aktivierungsfunktion für den letzten Dense_Layer gewählt worden ist wird das Wort welches den höchsten Wert zugewiesen bekommt als vorraussage definiert (Chigozie, Enyinna, Nwankpa 2020).

Layer

Dense Layer
Der Dense Layer wird auch als Fully Connected Layer bezeichnet. Es handelt sich hier um eine Standartschicht, bei der alle Neuronen mit sämtlichen Inputs und Outputs verbunden sind. In dem letzten Dense Layer findet die endgültige Klassifizierung statt (Emmert-Streib F 2020).

Dropout Layer
Der Dropout Layer ist eine Prävention für mögliches Overffitting. Dabei werden beim Training des Netzes eine definierte Anzahl von Neuronen beim nächsten Berechnungsschritt nicht beachtet. Das hat zufolge, dass, das Training des Netzes meist langsamer ist und sich weniger an die Trainingsdaten anpasst. Damit kann ein zu großer Dropout auch negative Folgen haben, allerdings ist es eine Möglichkeit, mögliches auswendig lernen des Netzes zu verhindern (Emmert-Streib F 2020).

LSTM_Layer(Long-Short-Term-Memory)
LSTM_Layer sind die Weiterentwicklung von RNN Layer, LSTM Laer verfügen über die Eigenschaft, vergangen Informationen in Sequenzen zu selektieren. Damit besitzen diese die Eigenschaft aus Vergangenen Werten die Informationen Herauszufiltern, die für den aktuellen Output relevant sind und welche nicht notwendig sind und somit vergessen werden können. Damit verwenden LSTM_Layer vermehrt in der Zeitreihenanalyse oder im NLP Bereich eingesetzt, da es sich dabei immer um Daten handelt die als Sequenz vorliegen (Emmert-Streib F 2020).

Word Embeddings
Alle im folgenden vorgestellten Architekturen beinhalten ein Embedding Layer bei welchen die Matrix auf ein Word Embedding basiert.

Beim Word Embedding wird ein Wort nicht als Ziffer dargestellt, sondern als Vektor im vektorraum dargestellt wodurch es möglich ist verschiedene Relationen zwischen den einzelnen Wörtern erkennbar zu machen. Je näher die Vektoren beieinander liegen desto eher stimmt die Bedeutung der Wörter überein oder die Wörter werden im gleichen Kontext verwendet.
Die Darstellung im Vektorraum ermöglicht es den Neuronalen Netz die Vorhersage des nächsten Wortes anhand des vorhandenen Kontextes vorherzusagen (Bengio 2014).
Für die vorgestellten Modelle wurde vermehrt ein Pretrained Word Embedding genutzt. Ausgewähltes Embedding: German CoNLL17 corpus vektor size 100
http://vectors.nlpl.eu/repository/

Embedding einbinden
Das Word Embedding wurde aufgrund der Dateiengröße nicht als Prüfungsleistung mit abgegeben. Zum einbinden des Embeddings muss diese unter den angegeben Link gedownloadet werden und die dazugehörig Text Datei muss unter folgenden Pfad mit dem Namen "model.txt“ abgelegt werden:
../Artefakt_LukasDech_9376/07_Word_Embedding_V100/

In [13]:

```
def get_embedding_vectors(dim=100):
    embedding_index = {}
    #Einlesen Pretrained Embedding
    with open(f"../Artefakt_LukasDech_9376/07_Word_Embedding_V100/model.txt",
              encoding='unicode_escape') as f:
        for line in tqdm.tqdm(f, "Reading Embedding"):

            values = line.split()
            if not values:
              break
            try:
                #Erster Wert im Array ist immer das Wort
                word = values[0]
                #Restlichen Werte repräsentieren den Vektor
                vectors = np.asarray(values[1:], dtype='float64')
                embedding_index[word] = vectors
            except ValueError:
                pass
    word_index = word_to_int
    embedding_matrix = np.zeros((len(word_index), dim))
    for word, i in word_index.items():
        embedding_vector = embedding_index.get(word)
        if embedding_vector is not None:
            embedding_matrix[i] = embedding_vector

    return embedding_matrix
```

Auswahl Analyseverfahren

Bedingt dadurch das die Auswahl einer geeignten Architektur sowie die Einstellung diverser Hyperparameter ein Iteratives Verfahren ist, werden im folgende verschiedene Archtiekturen trainiert und die Ergebnisse bewertet.

Erste Architektur

Für das Model werden sowohl LSTM Layer als auch Embedding Layer verwendet als Output Layer sowie als letzter Hiddenlayer wird ein Dense Layer verwendet.

Die Idee hinter der Architektur war es, dass mithilfe Die Input Sequenz von zwei Embedding Layer verarbeitet wird, damit zum einen das Model in der Lage ist durch das Pretrained Embedding das nächste Wort anhand des Kontexts vorherzusagen und das durch den zweiten Embedding Layer die Gewichte vermehrt, auf die im Text häufig vorkommenden Wörter gelegt werden. Weiterhin wurden als Hidden layer LSTM Layer verwendet damit die vom Embedding Layer übergebene Sequenz erhalten bleibt. Um ein mögliches gegen ein mögliches Overfitting vorzugehen, wurde auch ein Dropout von 15% der Verbindung zwischen Layern geschaltet.

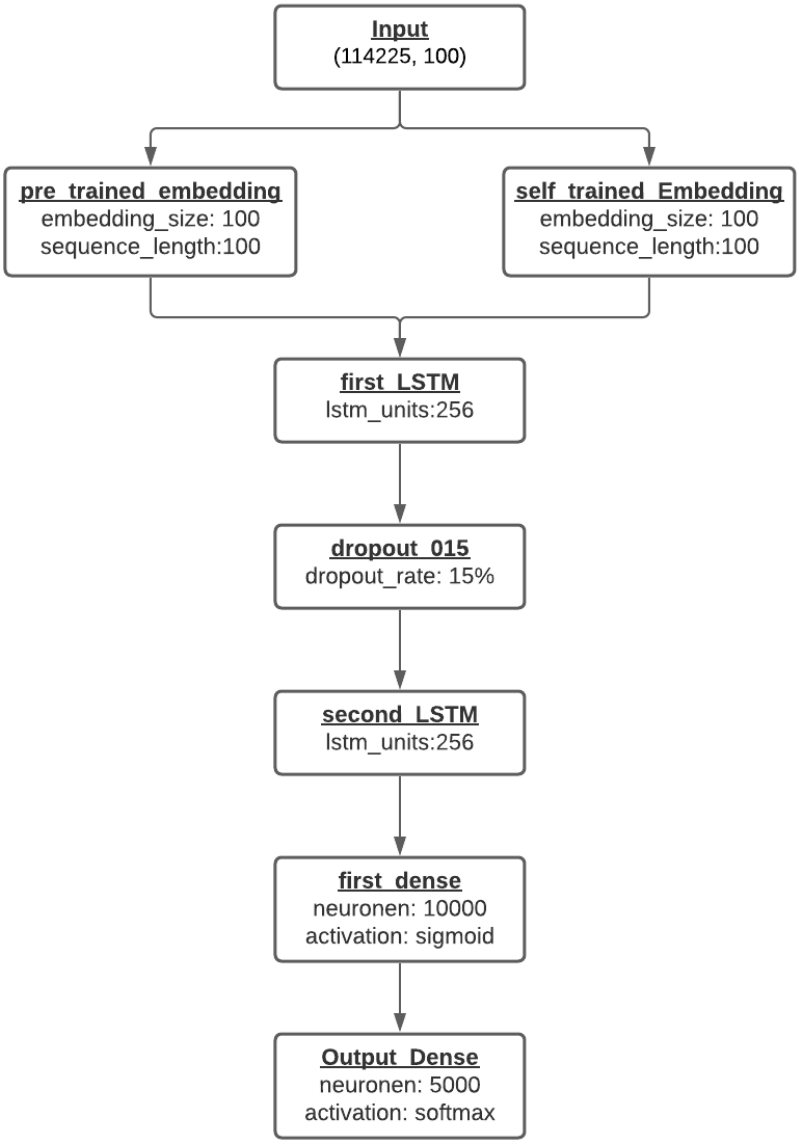


Abbildung 4 Erste Architektur (Eigene Darstellung)

In [14]:

```
embedding_matrix = get_embedding_vectors()
#Input_Layer
input1 = layers.Input((SEQUENCE_LENGTH))
input2 = layers.Input((SEQUENCE_LENGTH))

#Embedding_Layer
pretrained_Embedding = layers.Embedding(cv.max_features,
                        EMBEDDING_SIZE,
                        weights=[embedding_matrix],
                        trainable=False,
                        input_length=SEQUENCE_LENGTH)(input1)

selftrained_Embedding = layers.Embedding(cv.max_features,
                         EMBEDDING_SIZE,
                         trainable=True,
                         input_length=SEQUENCE_LENGTH)(input2)

#Embedding_Layer zusammenfügen
layerlist = [pretrained_Embedding, selftrained_Embedding]
concat = layers.Concatenate(axis = -1)(layerlist)
#LSTM first Hiddenlayer
first_LSTM = layers.LSTM(256, return_sequences = True)(concat)
#Dropout (Präventativ gegen mögliches overfitting)
dropout_015 = layers.Dropout(0.15)(first_LSTM)
#LSTM second Hiddenlayer
second_LSTM = layers.LSTM(256, return_sequences = False)(first_LSTM)
#Dense third Hiddenlayer
first_dense = layers.Dense(10000,activation= "sigmoid")(second_LSTM)
#Dense Output_Layer
output = layers.Dense(cv.max_features, activation="softmax")(second_LSTM)

#Model Trainieren
model = models.Model([input1,input2],output)
model.compile(optimizer="rmsprop", loss="categorical_crossentropy",
              metrics=["accuracy",tf.keras.metrics.Precision(),
                       tf.keras.metrics.Recall()])
```

...

In [15]:

```
#Model trainieren und abspeichern
model.fit([X,X],y, batch_size=BATCH_SIZE, epochs=EPOCHS,verbose=1)
model.save("speechGeneration1.pickle")
```

```
Epoch 1/10
1359/1359 [==============================] - 4333s 3s/step - loss: 5.7999 - accuracy: 0.1070 - precision: 0.6410 - recall: 0.0065
Epoch 2/10
1359/1359 [==============================] - 4252s 3s/step - loss: 5.2961 - accuracy: 0.1467 - precision: 0.6719 - recall: 0.0262
Epoch 3/10
1359/1359 [==============================] - 4231s 3s/step - loss: 5.1169 - accuracy: 0.1652 - precision: 0.6889 - recall: 0.0408
Epoch 4/10
1359/1359 [==============================] - 4222s 3s/step - loss: 5.0132 - accuracy: 0.1793 - precision: 0.6984 - recall: 0.0512
Epoch 5/10
1359/1359 [==============================] - 4132s 3s/step - loss: 4.9304 - accuracy: 0.1890 - precision: 0.7060 - recall: 0.0605
Epoch 6/10
1359/1359 [==============================] - 4098s 3s/step - loss: 4.8731 - accuracy: 0.1970 - precision: 0.7128 - recall: 0.0686
Epoch 7/10
1359/1359 [==============================] - 4049s 3s/step - loss: 4.8005 - accuracy: 0.2052 - precision: 0.7188 - recall: 0.0756
Epoch 8/10
1359/1359 [==============================] - 4135s 3s/step - loss: 4.7704 - accuracy: 0.2117 - precision: 0.7194 - recall: 0.0831
Epoch 9/10
1359/1359 [==============================] - 4101s 3s/step - loss: 4.7402 - accuracy: 0.2190 - precision: 0.7272 - recall: 0.0891
Epoch 10/10
1359/1359 [==============================] - 4107s 3s/step - loss: 4.7416 - accuracy: 0.2252 - precision: 0.7300 - recall: 0.0949
```

Zweite Architektur

Die zweite Architektur ist ähnlich aufgebaut wie die erst Architektur, jedoch verlaufen alle Layer sequentiell und es wird nur mit einen Embedding Layer und den Pretrained Word Embedding gearbeitet. Da die beiden Architekturen ähnlich sind können die Ergebnisse Aufschluss darüber geben, ob sich das Parallel schalten der Embeddinglayer negativ oder positiv auf das Ergebnis ausgewirkt hat.

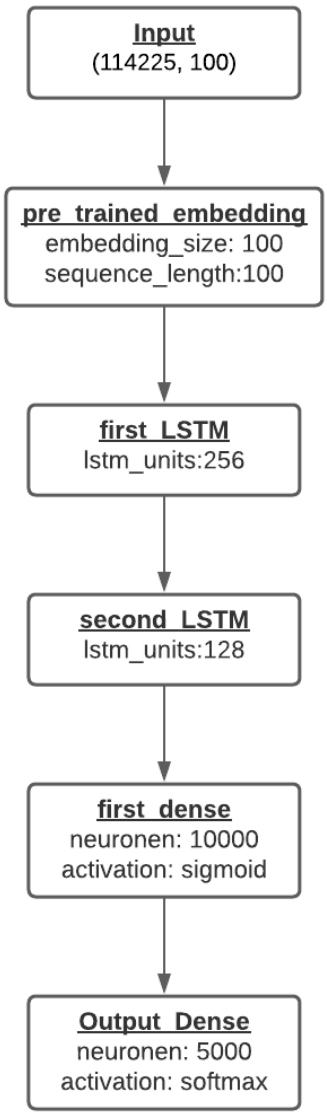


Abbildung 5 Zweite Architektur (Eigene Darstellung)

In [16]:

```
#Model Initalisieren und Hyperparamter einstellen
model = Sequential()
model.add(Embedding(cv.max_features,
                    EMBEDDING_SIZE,weights=[get_embedding_vectors()],
                    trainable=False,
                    input_length=SEQUENCE_LENGTH))
model.add(LSTM(256, return_sequences=True))
model.add(LSTM(128))
model.add(Dense(10000,activation= "sigmoid"))
model.add(Dense(cv.max_features, activation="softmax"))

model.compile(optimizer="rmsprop", loss="categorical_crossentropy",
              metrics=["accuracy",tf.keras.metrics.Precision(),
                       tf.keras.metrics.Recall()])
```

```
Reading GloVe: 142115it [00:04, 34530.03it/s]
```

In [17]:

```
#Model trainieren und abspeichern
model.fit(X,y,batch_size=BATCH_SIZE, epochs=EPOCHS,verbose=1)
model.save("speechGeneration2.pickle")
```

```
Epoch 1/10
1359/1359 [==============================] - 5426s 4s/step - loss: 6.4799 - accuracy: 0.0662 - precision: 0.0312 - recall: 1.1505e-04
Epoch 2/10
1359/1359 [==============================] - 5320s 4s/step - loss: 6.2647 - accuracy: 0.0674 - precision: 0.0000e+00 - recall: 0.0000e+00
Epoch 3/10
1359/1359 [==============================] - 5311s 4s/step - loss: 6.2535 - accuracy: 0.0674 - precision: 0.0000e+00 - recall: 0.0000e+00
Epoch 4/10
1359/1359 [==============================] - 5183s 4s/step - loss: 6.1752 - accuracy: 0.0680 - precision: 0.4091 - recall: 1.5532e-04
Epoch 5/10
1359/1359 [==============================] - 5113s 4s/step - loss: 6.0841 - accuracy: 0.0698 - precision: 0.4737 - recall: 0.0013
Epoch 6/10
1359/1359 [==============================] - 5061s 4s/step - loss: 5.9521 - accuracy: 0.0861 - precision: 0.5231 - recall: 0.0027
Epoch 7/10
1359/1359 [==============================] - 5130s 4s/step - loss: 5.8659 - accuracy: 0.0986 - precision: 0.5803 - recall: 0.0065
Epoch 8/10
1359/1359 [==============================] - 5115s 4s/step - loss: 5.7979 - accuracy: 0.1135 - precision: 0.6086 - recall: 0.0146
Epoch 9/10
1359/1359 [==============================] - 3384s 2s/step - loss: 5.7354 - accuracy: 0.1218 - precision: 0.6350 - recall: 0.0237
Epoch 10/10
1359/1359 [==============================] - 2419s 2s/step - loss: 5.6777 - accuracy: 0.1292 - precision: 0.6511 - recall: 0.0287
```

Dritte Architektur

Die Dritte Architektur basiert vermehrt auf der Nutzung von LSTM Layern. Dabei ist der erste Input Layer wie in den vorangegangenen Modellen ein Embedding Layer, danach werden abwechselnd LSTM Layer und Dropout Layer hintereinander geschaltet.

Wie bei den anderen Architekturen ist auch bei dieser der Output Layer ein Dense Layer, bei welchen die Anzahl an Neuronen den trainierten Vokabular entspricht.

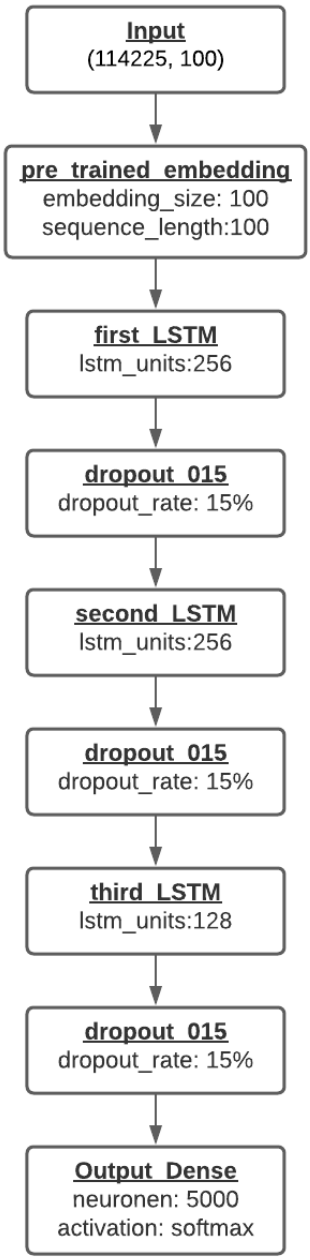


Abbildung 6 Dritte Architektur (Eigene Darstellung)

In [18]:

```
#Model Initalisieren und Hyperparamter einstellen
model = Sequential()
model.add(Embedding(cv.max_features,
                    EMBEDDING_SIZE,weights=[get_embedding_vectors()],
                    trainable=False,
                    input_length=SEQUENCE_LENGTH))
model.add(LSTM(256, return_sequences=True))
model.add(Dropout(0.15))
model.add(LSTM(256, return_sequences=True))
model.add(Dropout(0.15))
model.add(LSTM(128))
model.add(Dropout(0.15))
model.add(Dense(cv.max_features, activation="softmax"))

model.compile(optimizer="rmsprop", loss="categorical_crossentropy",
              metrics=["accuracy",tf.keras.metrics.Precision(),
                       tf.keras.metrics.Recall()])
```

```
Reading GloVe: 142115it [00:04, 33695.01it/s]
```

In [19]:

```
#Model trainieren und abspeichern
model.fit(X,y,batch_size=BATCH_SIZE, epochs=EPOCHS,verbose=1)
model.save("speechGeneration3.pickle")
```

```
Epoch 1/10
1359/1359 [==============================] - 4765s 4s/step - loss: 6.0788 - accuracy: 0.0820 - precision: 0.4729 - recall: 7.0184e-04
Epoch 2/10
1359/1359 [==============================] - 4936s 4s/step - loss: 5.6417 - accuracy: 0.1158 - precision: 0.4966 - recall: 0.0055
Epoch 3/10
1359/1359 [==============================] - 4909s 4s/step - loss: 5.4931 - accuracy: 0.1254 - precision: 0.5331 - recall: 0.0119
Epoch 4/10
1359/1359 [==============================] - 4858s 4s/step - loss: 5.4724 - accuracy: 0.1300 - precision: 0.5597 - recall: 0.0174
Epoch 5/10
1359/1359 [==============================] - 4762s 4s/step - loss: 5.4004 - accuracy: 0.1388 - precision: 0.5992 - recall: 0.0244
Epoch 6/10
1359/1359 [==============================] - 4725s 3s/step - loss: 5.3715 - accuracy: 0.1445 - precision: 0.6122 - recall: 0.0300
Epoch 7/10
1359/1359 [==============================] - 4785s 4s/step - loss: 5.3669 - accuracy: 0.1499 - precision: 0.6209 - recall: 0.0344
Epoch 8/10
1359/1359 [==============================] - 4762s 4s/step - loss: 5.3547 - accuracy: 0.1548 - precision: 0.6295 - recall: 0.0388
Epoch 9/10
1359/1359 [==============================] - 4424s 3s/step - loss: 5.3721 - accuracy: 0.1585 - precision: 0.6373 - recall: 0.0423
Epoch 10/10
1359/1359 [==============================] - 3413s 3s/step - loss: 5.3879 - accuracy: 0.1640 - precision: 0.6553 - recall: 0.0471
```

Ergebnisse Evaluieren

Zur Evaluierung der Ergebnisse wird die Performance jedes Modells anhand folgender Metriken gemessen: loss, accuracy, precision, recall

Damit die genannten Parameter ermittelt werden können wird ein bisher noch nicht betrachtetes Plenarprotokoll eingelesen und durchläuft den gleichen Preprocessing Prozess wie die Trainingsdaten.

Anschließend können die trainierten und geladenen Modelle, anhand einer Eingabe Sequenz aus dem Plenarprotokoll, die darauf folgenden Wörter vorhersagen. Die Anschließende Bewertung der Ergebnisse erfolgt im Fazit.

Erklärung Metriken

loss
Im Verhältnis zu den anderen Werten ist der Loss keine Prozentangabe. Loss ist die Summe von Fehlern, die für jedes Beispiel gemacht worden sind. Damit weist eine niedrigere Zahl auf eine bessere Performance hin, eine zu niedriger Loss auf Trainingsdaten kann auch ein Anzeichen von Overfitting sein.

accuracy
Treffer Wahrscheinlichkeit, wie oft wurde tatsächlich die richtige Klasse vorhergesagt

precision
Das Verhältnis von Positiv richtig vorhergesagten Werten zu den generell positiv vorhergesagten Werten.
True_Positiv/True_Positiv+False_Positiv

recall
Verhältnis von allen Positiv Richtig vorhersagen eins Worts zu den insgesamten Möglichkeiten eine richtige Vorhersage zutreffen
True_Positiv/True_Positiv+False_Negativ

In [20]:

```
model1 = keras.models.load_model('../Artefakt_LukasDech_9376/03_Modell_trained/Archi
model2 = keras.models.load_model('../Artefakt_LukasDech_9376/03_Modell_trained/Archi
model3 = keras.models.load_model('../Artefakt_LukasDech_9376/03_Modell_trained/Archi
```

In [21]:

```
with open("../Artefakt_LukasDech_9376/03_Modell_trained/Architektur1/word_to_int.spe
    word_to_int = pickle.load(file)

with open("../Artefakt_LukasDech_9376/03_Modell_trained/Architektur1/int_to_word.spe
    int_to_word = pickle.load(file)
```

In [79]:

```
dfs = []
#Einlesen der Plenarprotokolle as der Drive
for i in range(160,161):
    protokoll_path = f"../Artefakt_LukasDech_9376/06_TXT/{i}.csv"
    dfs.append(pd.read_csv(protokoll_path, encoding= 'unicode_escape',
                           sep = ",", error_bad_lines= False))
#Zusammenfügen der einzelnen Protokolle im Pandas Df
speeches_concat = pd.concat(dfs, ignore_index= True)
#Filtern der Daten, sodass nur notwendige Daten erhalten bleiben
filtered_column_speeches = speeches_concat[["speaker_party","text","top","type"]]
filtered_row_speeches = filtered_column_speeches[
                            (filtered_column_speeches["speaker_party"].notnull()) &
                            (filtered_column_speeches["type"] == "speech") &
                            (filtered_column_speeches["speaker_party"] == "cducsu")]
#Alle Reden als zusammenhängenden Text darstellen
all_speeches_series = pd.Series(list(filtered_row_speeches["text"]))
all_speeches_text = all_speeches_series.str.cat(sep=' ')
#Text in einezelne Wörter zerlegen
tokens_speeches = word_tokenize(all_speeches_text)
tokens_transformed = [word_to_int[word] for word in tokens_speeches if word in word_
X = []
y = []
# x = vektor an zahlen (satz)
# y = nächstes wort (One-Hot-Encoding)
for i in range(0, len(tokens_transformed) - SEQUENCE_LENGTH):
    X.append(tokens_transformed[i:i + SEQUENCE_LENGTH])
    y.append(tokens_transformed[i+SEQUENCE_LENGTH])
X = np.array(X)
y = np.array(to_categorical(y, num_classes = cv.max_features))
sentence = tokens_transformed[100:200]
print(" ".join([int_to_word[token] for token in sentence]))
sentence = np.array(tokens_transformed[100:200])
```

```
und Erfolg wÃ¼nschen , der es auch zu langsam geht , nÃ¤mlich unsere
Bundeskanzlerin . - Da darf der bei unserem Koalitionspartner ruhig et
was grÃ¶Ã  er sein ; denn ich gehe davon aus , dass auch Sie den Erfolg
wollen . Meine sehr verehrten Kolleginnen und Kollegen , ja , das , da
s Europa im in Griechenland zeigt , ist nicht das von Europa , das ich
mir in der ganzen Welt . Das , was wir in Griechenland erleben , ist a
ber das Ergebnis davon , dass Europa nicht schnell genug gehandelt hat
, und das ist das Ergebnis ausschlieÃ  lich
```

In [81]:

```
score1 = model1.evaluate([X,X], y, verbose=0)
score2 = model2.evaluate(X, y, verbose=0)
score3 = model3.evaluate(X, y, verbose=0)

print(f"{model2.metrics_names[0]}/{model2.metrics_names[1]}/
        {model2.metrics_names[2]}/{model2.metrics_names[3]}")
print(f"Modell_1: {score1[0]} / {score1[1]}  / {score1[2]}  / {score1[3]}")
print(f"Modell_2: {score2[0]}  / {score2[1]} / {score2[2]} / {score2[3]}")
print(f"Modell_3: {score3[0]} / {score3[1]} / {score3[2]}  / {score3[3]}")
```

```
                loss        /       accuracy      /      precision
/       recall
Modell_1: 4.854308128356934 / 0.2015209197998047  / 0.6295081973075867
/ 0.07684610784053802
Modell_2: 5.32698917388916  / 0.13047829270362854 / 0.4776632189750671
4 / 0.027816690504550934
Modell_3: 5.067773818969727 / 0.17590554058551788 / 0.6263157725334167
/ 0.047628577798604965
```

5.3 Nutzbarmachung

Damit die gewonnen Erkenntnisse oder Das Modell nutzbar gemacht werden können, muss nach Ziel des Projektes ausgewählt werden in welcher Form die Daten bereitgestellt werden sollen. Sobald das Ziel definiert ist, muss die Sowohl die technische Umsetzbarkeit geprüft werden als auch die Anwendbarkeit. Sollte beides gegeben sein könne die Analyse Ergebnisse bereitgestellt werden.
Da bisher nur die Metriken der Modelle geprüft worden sind, werden im Folgenden die Modelle für den eigentlichen nutzen vorbereitet und dazu verwendet Texte vorherzusagen. Dabei werden die vorhergesagten Klassen der Modelle zurück in die Vorhergesagten Wörter codiert, um zu erkennen wie Sinnvoll der generierte Inhalt und Satzstruktur ist.

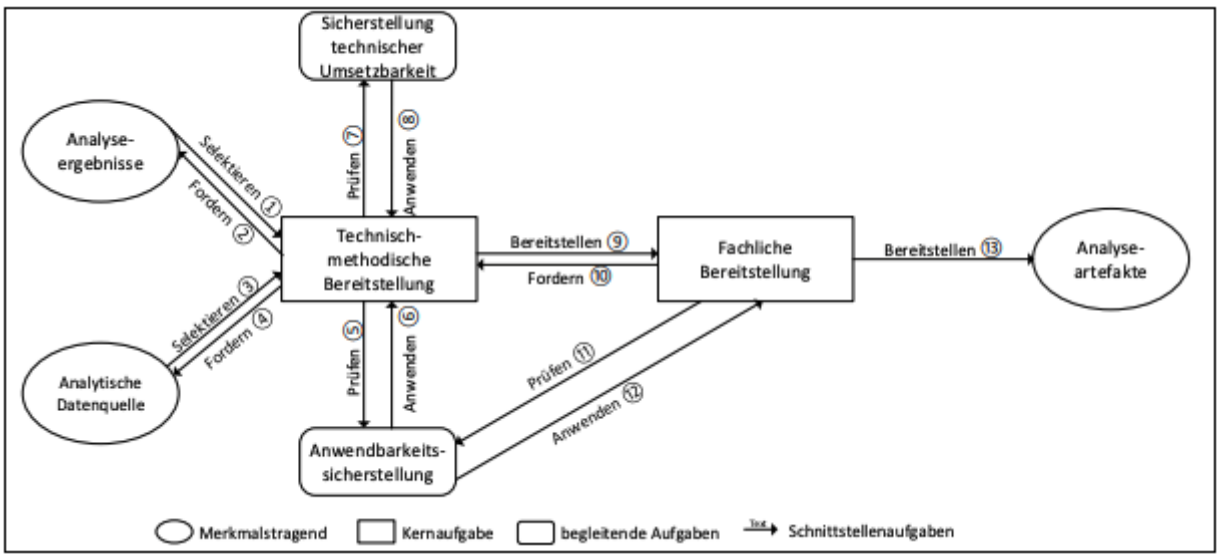


Abbildung 7 Schlüsselbereich Nutzbarmachung (Neuhaus, et al. 2020, S.53)

In [95]:

```
print(" \nPrediction Model_1: \n")
for i in range(0, 100):
    prediction = model1.predict([sentence.reshape(1, 100),
                                 sentence.reshape(1, 100)])
    word = np.random.choice(len(int_to_word), p=prediction[0])
    print(int_to_word[word], end=" ")
    sentence = np.append(sentence[1:], [word])
```

```
Prediction Model_1:

guten Aber wenn die indem die so kommt , dass man zu nutzen . Aber wir
brauchen den Mali ; denn kÃ¶nnen Sie uns mit der Verhandlungen gesagt
. Auch ich bin Herr verehrten Damen und Herren , die Sache , bei der a
ktuellen Montag sein kann . Dabei kÃ¶nnen wir eigentlich einmal genug
? Daher wurde das geht nicht so sorgen , dass die Zahl der Verteilung
und deutlich fÃ¼hren kÃ¶nnen . Die Industrie zeigen die TÃ¼rkei auch a
n eine Partner an Europa zu Schritt , meine Damen und Herren . Das The
ma des Verbraucherschutz im Osten . Ich
```

In [96]:

```
print(" \nPrediction Model_2: \n")
for i in range(0, 100):
    prediction = model2.predict([sentence.reshape(1, 100),
                                 sentence.reshape(1, 100)])
    word = np.random.choice(len(int_to_word), p=prediction[0])
    print(int_to_word[word], end=" ")
    sentence = np.append(sentence[1:], [word])
```

```
Prediction Model_2:

, ist ein auch . Deswegen Rahmen und Kolleginnen , Ihre Sie sowie kÃ¶n
nen das mir um heiÃ  t . Ich schutzbedÃ¼rftig - Damen glaube wir , auf
ein . Frau richtige mÃ¶chte sich eine mehr , diese ganz ihm , denn wir
in dem Verbesserung erreichen , , den nach geworden , den fÃ¼r uns es
viel es einmal , Ihren Fall ; Zur freue wir sind . Die Nach . ist scho
n aber , dass doch heute an , zu in die Wirtschaft - in Deutschland is
t es Vielen , und wenn ich etwas , ist eine . Wir
```

In [97]:

```
print(" \nPrediction Model_3: \n")
for i in range(0, 100):
    prediction = model3.predict([sentence.reshape(1, 100),
                                 sentence.reshape(1, 100)])
    word = np.random.choice(len(int_to_word), p=prediction[0])
    print(int_to_word[word], end=" ")
    sentence = np.append(sentence[1:], [word])
```

```
Prediction Model_3:

unserem schaffen wir mir ganz eben nicht so . Im hÃ¤tte doch richtig d
ieser Diskussion nur gesagt worden sein kÃ¶nnen , brauchen wir uns ja
auch nicht fÃ¼r das Recht was . In Vor hat jetzt glaube , Ihnen haben
wir viele stÃ¤rken , der wie sollen dieses andere sind , fÃ¼r die deut
schen Hilfe 1 . meiner bis Linke bin gut zwei Milliarden anderes schaf
fen . die dem GrÃ¼nen so , was man sagen man als Kollegen geworden ist
der wichtige Frage auf die Wirtschaft der Niedersachsen und richten wi
r gesagt , dass Â§ Thema , dass gemeinsam ,
```

5.4 Nutzung

Nach dem die Ergebnisse so weit aufbereitet worden sind, dass diese im laufenden Betrieb genutzt werden können, muss in der weiterführenden Anwendung gewährleistet werden, dass das Modell weiterhin überwacht wird. Die Überwachung ist notwendig da sich die Datenlage im Betrieb, im Verhältnis zur Testdatenlage signifikant geändert haben kann, wodurch die Genauigkeit des Modells im Livebetrieb gebärdet ist. Außerdem müssen die gewonnen Ergebnisse in der zuständigen Fachabteilung Verwendung finden.
Da das Ziel war die generierten Texte bereitzustellen und anhand dessen zukünftige Reden anzupassen, erfolgt die Nutzung durch das Generieren einer Text Datei. In die Text Datei werden von trainierten Architekturen, jeweils die nächsten 250 auf eine Eingabe Sequenz vorhergesagt. Der eigentliche Nutzen und Erfüllung des Ziel erfolgt demnach mit der Benutzung der Textdatei als Vorlage für den Nutzer.

Eine generiete Textdatei wurde dabei bereits mit als Prüfungsleistung abgegeben

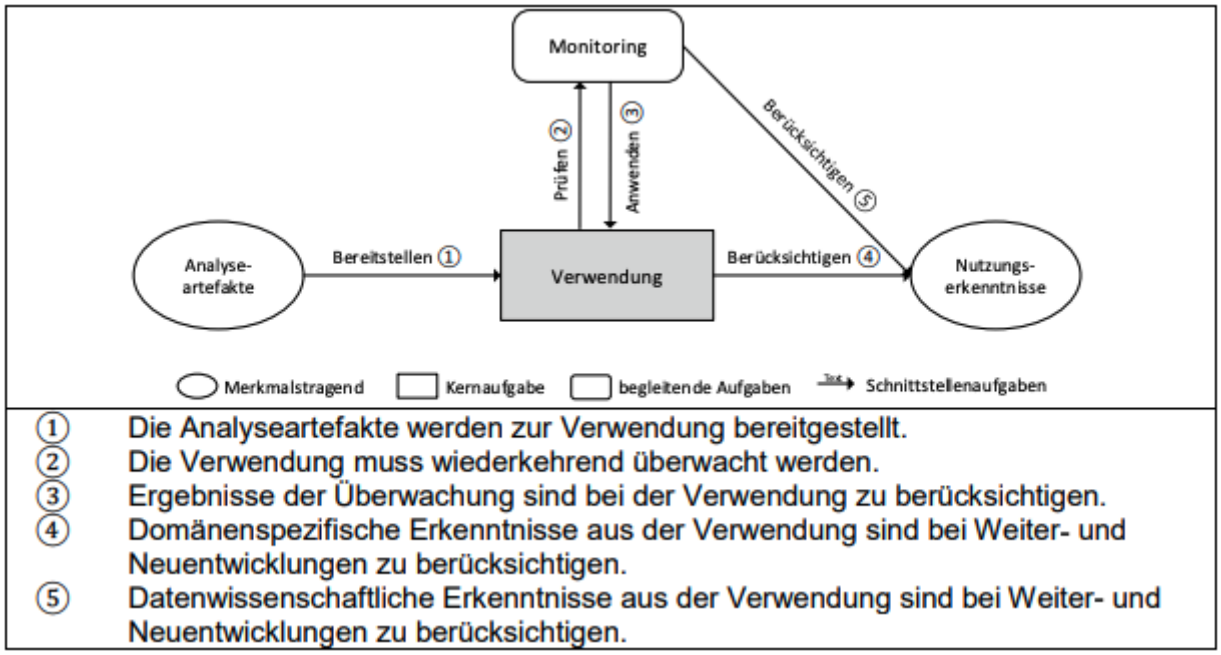


Abbildung 8 Schlüsselbereich Nutzung (Neuhaus, et al. 2020, S.66)

In [107]:

```
with open('generierter_Text.txt', 'w') as f:

    f.write("\nEingabe Sequenz:\n \n")
    f.write(" ".join([int_to_word[token] for token in sentence]))
    f.write("\n \n \n")

    f.write("\nText generiert mit der ersten Architektur: \n \n")
    for i in range(0, 250):
        prediction = model1.predict([sentence.reshape(1, 100)
                                     ,sentence.reshape(1, 100)])
        word = np.random.choice(len(int_to_word), p=prediction[0])
        f.write(int_to_word[word])
        f.write(" ")
        sentence = np.append(sentence[1:], [word])

    f.write("\n \n \n")
    f.write("\nText generiert mit der zweiten Architektur: \n \n")
    for i in range(0, 250):
        prediction = model2.predict([sentence.reshape(1, 100)
                                     ,sentence.reshape(1, 100)])
        word = np.random.choice(len(int_to_word), p=prediction[0])
        f.write(int_to_word[word])
        f.write(" ")
        sentence = np.append(sentence[1:], [word])

    f.write("\n \n \n")
    f.write("\nText generiert mit der dritten Architektur: \n \n")
    for i in range(0, 250):
        prediction = model2.predict([sentence.reshape(1, 100)
                                     ,sentence.reshape(1, 100)])
        word = np.random.choice(len(int_to_word), p=prediction[0])
        f.write(int_to_word[word])
        f.write(" ")
        sentence = np.append(sentence[1:], [word])
```

6. Fazit

Das Ziel des Artefaktes wurde insofern erreicht, da es möglich war ein Model zu entwickeln welches in der Lage ist eine Text basierend auf den eingelesenen Plenarprotokollen zu generieren.
Allerdings hat die Verwendung des Models gezeigt, dass die generierten Textstrukturen sowie die Grammatik nicht den Anforderung entsprechend sind, um das Model als Vorlage für zukünftig geschriebene Reden zu verwenden.

Die ermittelten Metriken deuteten bereits darauf hin, dass die Ergebnisse nicht den Anforderungen entsprechen werden. Zwar ist eine Accuracy von 15-20% bei einer Anzahl von 5000 Klasen weitaus besser als eine Zufällige Vorhersage, dennoch wird dabei nur jedes 5te Wort richtig vorhergesagt.
Sätze besitzen eine empfohlene durchschnittslänge von 15 Wörtern, damit sind pro Satz ohne Betrachtung der Satzeichen, nur 3 Wörter richtig vorhergesagt. Die damit Verbunde Anzahl an richtig vorhergesagten Worten reicht also nicht aus, um einen qualitativ hochwertigen Text zu generieren oder verständlichen Inhalt aufzuzeigen.

Zum einen lässt sich die mindere Qualität der generierten Texte auf das eingeschränkte Vokabular zurückführen, da die eingelesen Texte bereits 20000 verschiedene Wörter beinhalteten.
Eine Möglichkeit die Qualität der Texte zu steigern, wäre es das Vokabular zu erweitern. Damit würde allerdings die Trainingsdauer zunehmen und diese betrug bereits bei einen Vokabular von 5000 Wörtern und 10 Trainingsepochen, über 15 Stunden.

Wie bereits aufgezeigt beläuft sich das aktiv genutzte Vokabular von Muttersprachlern auf 5000 Wörtern. Allerdings ist dabei zu betrachten, dass es sich dabei um verschiedene Wörter handelt. Das Model hingegen stellt jede Änderung eines Wortes als eigene Klasse dar.
Schlussfolgernd daraus sind auch alle Konjugation von Verben eigenständige Wörter. Wodurch unter dem erlernten Vokabular auch etliche Wörter sind, welche angepasst an gegebene die gegeben Personalpronomen oder Zeitformen, mehrmals erlernt worden sind.

Damit besteht die Möglichkeit den generieten Text, bei gleichbleibender Trainingsdauer zu verbessern, indem alle Wörter im Trainingsdatensatz auf dessen Infinitiv heruntergebrochen werden. Allerdings würde damit auch die Grammatikalische Struktur des Textes verschlechtert werden.

Außerdem konnte ein Ergebnis bei der Betrachtung der unterschiedlichen Architekturen erzielt werden. Das erste Model wurde mit 2 Embedding Layer gebaut und ein 15% Dropout zwischen den LSTM Layern, abgesehen davon war es identisch mit der zweiten Architektur. Anhand der Metriken ist zu erkennen, dass die erste Architektur auf den Testdatensatz eine bessere Performance hatte, außerdem war auch die Performance, des ersten Modells auf den Trainingssatz besser. Resultierend daraus ist bessere Performance des ersten Models nicht auf Dropout zurückführen, sondern auf das Parallelschalten der Embedding Layer.

Die Idee dahinter war es neben den Pretrained Word Embedding, den tatsächlich vorkommenden Wörter im Text eine höhere Gewichtung zu verleihen.
Die Verbesserung der Performance ist insofern interessant, da auch nur mit der Verwendung des Pretrained Word Embedding die Gewichtung hinsichtlich der tatsächlich vorkommenden Wörter, mithilfe der Verbindungen zwischen den LSTM Layern, angepasst werden sollte.
Die Erkenntnis, dass es ein positiver Effekt auf ein Model haben kann, wenn zwei Embedding Layer genutzt werden, kann in zukünftigen Arbeiten von Vorteil sein.

Weiterhin ist festzustellen das die Performance der dritten Architektur auf dem Testdatensatz besser war als im Training. Die zweite Architektur hat hingegen deutlich an Performance auf dem Testdatensatz verloren. Resultierend daraus lässt sich schlussfolgern das der vermehrte Einsatz von Dropouts einen positiven Effekt bei einem solchem Problem darstellen kann.
Außerdem kann damit gesagt das die zweite Architektur leicht overfitted ist, wodurch keine Steigerung der

Performance auf den Testdaten, bei einer Steigerung der trainierten Epochen, zu erwarten ist. Allerdings wird daraus geschlussfolgert, dass die dritte Architektur mit einer Anzahl von 10 trainierten Epochen leicht underfitted ist und somit eine Steigerung der Epochen im weiteren Projektverlauf ratsam wäre. Leider war die Anpassung der Modelle im durch die Trainingsdauer und den zeitlich begrenzten Rahmen der Hausarbeit nicht mehr möglich.

die Performance der Modelle könnte weiterhin mit einer Änderung der Hyperparmeter gesteigert werden. Bedingt durch die lange Trainingsdauer ist dies nur bedingt passiert, die erste Architektur wurde im Laufe der Hausarbeit auf verschiedenen Batchgrößen und bis zu 25 Epochen trainiert, allerdings konnte dieses Model durch Hardware technische Probleme nicht gespeichert werden.

Allerdings konnte in den Trainingsprozess beobachtet werden das bei einer Größen Epochen Anzahl die Steigerung der Accuracy nur noch im 0,01 Bereich erfolgt ist. Weshalb alle weiteren Modelle auf 10 Trainingsepochen beschränkt worden sind. Außerdem brachte eine Änderung der Loss Function keine Performance Steigerung mit sich.

Abschließend ist zu sagen das die Aufgabe mit den gegebenen Ressourcen und den vorliegenden Architekturen deutlich zu komplex war, um ein befriedigendes Ergebnis zu erlangen. Die Generierung von Natürlicher Sprache besitzt zu viele Kombinationsmöglichkeiten, besonders bei der Wahl von Plenarprotokolle als Datengrundlage, da dort auch vermehrt Fachwörter vorkommen, die nicht im alltäglichen Sprachgebrauch relevant sind, aber dennoch für die Sinnhaftigkeit des Inhaltes erlernt werden müssen.

Weiterhin ist festzuhalten, dass während der Ausarbeitung aufgefallen ist, dass klassische Metriken die normalerweise zur Evaluierung der Ergebnisse herangezogen werden, wie accuracy, precision und recall keine große Aussagekraft darüber bieten wie gute der generierte Text wirklich ist. Viele Wörter besitzen ähnliche Bedeutung oder können in Abhängigkeit des Kontextes auch durch andere ersetzt werden. Damit kann auch bei der Vorhersage einer falschen Klasse ein grammatikalisch und inhaltlich Sinnvoller Text generiert werden. Bei der Recherche in laufe der Hausarbeit wurde keine zuverlässige Metrik gefunden, welches das Problem löst und den generierten Text qualitativ auf dessen Logik bewerten kann. Damit kann die Untersuchung von Zielführenderen Metriken in der Textgenerierung ein mögliches Thema weiterführende Forschung sein.

7. Literaturverzeichnis

- Ipsos. 2020. Wie groß ist Ihrer Meinung nach der Einfluss von Künstlicher Intelligenz auf die folgenden Bereiche? Bericht, VdTÜV.
- Neuhaus, Uwe, Michael Schulz, Jens Kaufmann, Badura Daniel, Kerzel Ulrich, Welter Felix, Prothmann Maik, et al. 2020. DASC-PM v1.0 - Ein Vorgehensmodell für Data-Science-Projekte. Hamburg.
- Römer, Christine. 2005. Lexikologie des Deutschen. Gunter Narr Verlag.
- Chigozie Enyinna Nwankpa, Winifred Ijomah, Anthony Gachagan, and Stephen Marshall. 2020. Activation Functions: Comparison of Trends in Practice and Research for Deep Learning.
- Emmert-Streib F, Yang Z, Feng H, Tripathi S and Dehmer M (2020) An Introductory Review of Deep Learning for Prediction Models With Big Data. Front. Artif. Intell. 3:4. doi: 10.3389/frai.2020.00004
- Bengio, Samy, and Georg Heigold. "Word embeddings for speech recognition." (2014).