KU LEUVEN

Dialogue Breakdown Detection in Chatbots

Mariya Hendriksen

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Speech and Language Technology

Thesis supervisor:
Prof. Dr. Marie-Francine Moens

Assessor:
Prof. Dr. Hugo Van Hamme

Mentor:
Artuur Leeuwenberg

Academic year 2018 – 2019

Preface

I would like to thank everyone who supported and helped me in writing the thesis.

First of all, I would like to express the genuine gratitude to my supervisor, Prof. Dr. Marie-Francine Moens and my mentor, Artuur Leeuwenberg for guiding me throughout the internship and helping me to structure my thoughts. I would also like to thank my assessor, Prof. Dr. Hugo Van Hamme for taking the time to read my thesis.

Secondly, a heartfelt thanks to my parents, brothers and friends for their continuous encouragement and understanding.

Last but not least, I thank my husband Rinke for his everlasting support, patience and love.

Mariya Hendriksen

Contents

Abstract

One of the biggest problems of human-chatbot interaction is miscommunication. Occurring mainly on behalf of the chatbot, miscommunication can lead to dialogue breakdown, i.e., a point when the dialogue cannot be continued. Detecting such points can facilitate breakdown prevention or recovery after breakdown happened.

The thesis represent a report for an internship undertaken at Language Intelligence & Information Retrieval Lab. The goal of the project is to build a multinomial sequence classifier to detect breakdowns in human-chatbot dialogues.

The project includes an analysis of the existing dialogue breakdown detectors for defining a baseline. Besides, it entails the development of nine LSTM models with different architectures and embedding types. The experiments allowed to investigate the relationship between model architecture, embedding type and performance. Besides, two best performing models were selected and compared with the baseline. Overall, the created models demonstrated improvement over the baseline.

List of Figures and Tables

List of Figures

List of Tables

List of Abbreviations and Symbols

Abbreviations

NB	Not a breakdown
PB	Possible breakdown
B	Breakdown
CNN	Convolutional Neural Network
CRF	Conditional Random Fields
DBDC	Dialogue Breakdown Detection Challenge
EMLo	Embeddings from Language Model
ETR	Extra Trees Regressor
GloVe	Global Vectors
LSTM	Long Short-term Memory Network
MaxEnt	Maximum Entropy classifier
MemNN	End-to-End Memory Network
NLTK	Natural Language Toolkit
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
sklearn	scikit-learn

Symbols

κ	Fleiss' kappa coefficient
y	true value
$\hat{y}$	predicted value

Chapter 1

Introduction

The thesis represents a summary of the research internship done at the Language Intelligence & Information Retrieval Lab. The project's main goal was to develop a model for dialogue breakdown detection in chat-oriented dialogue systems.

1.1 Motivation

A dialogue system is a computer system whose objective is to converse with a human. It can employ different communication modes such as speech, text, gestures, etc[55].

A chatbot is a dialogue system which conducts a conversation with a user via text or speech. The very first chatbots were introduced about half a century years ago. One of the most known examples was ELIZA[54]. Even though many people believed that ELIZA communicated with genuine understanding, it used pattern matching to simulate a Rogerian psychoterapist[1]. Overall, ELIZA was a rather simple chatbot in terms of both implementation and functionality.

Nowadays, more than fifty years after ELIZA was created, chatbots become an increasingly important part of our life[2]. They are used in various domains such as website help, education, customer service, e-commerce, entertainment. They appeal because of their ability to provide an instant 24-hours service.

The majority of modern chatbots are integrated into messaging apps (e.g., WeChat or Messenger), virtual assistants (e.g., Cortana or Google Assitant). Besides, chatbot can also be implemented as a part of individual application or website. Chatbots research is geared by such challenges as The Conversational Intelligence Challenge (CIC, part of NeurIPS 2018)[3] and the Workshops and Session Series on Chatbots

[1]person-center therapy, based on empathic understanding[44]

[2]Study:Chatbots gain popularity with consumers, especially millennials, URL: https://www.mobilemarketer.com/news/study-chatbots-gain-popularity-with-consumers-especially-millennials/447490/, last checked on 07-01-2019.

[3]The Conversational Intelligence Challenge 2, URL: http://convai.io/, last checked on 07-01-2019.

and Conversational Agents (WOCHAT)[4].

Naturally, the usefulness of a chatbot largely depends on its ability to interact with users. The smoother communication goes, the better the chatbot is. One of the major obstacles on the way to the goal is miscommunication. We define miscommunication as a situation when a chatbot does not understand a user correctly and returns an inappropriate reply. In other words, we assume that miscommunication mainly occurs on the behalf of chatbot.

Miscommunication leads to dialogue breakdown. Breakdown can be defined as a point in a dialogue when the interaction is interrupted with or without completion of the performed task [35]. In this perspective, successfully detecting breakdowns and potential breakdowns can contribute to the improvement of human-chatbot interaction. Such a detector could help to avoid system responses which cause breakdown and to identify breakdowns after they occur and launch the procedures necessary to get out of the breakdown situation.

For this purpose, we decided to investigate the dialogue breakdown detection during the internship. The idea is to build a dialogue breakdown detector and to compare its performance with the baseline model.

The problem is being actively discussed in the research community. In fact, to promote development in the field, The dialogue breakdown detection challenge (DBDC) was launched several years ago[47]. The organizers of the challenge provide training and evaluation datasets available both in English and Japanese language. Each system response is marked with one of the three labels:

- NB – not a breakdown: it is possible to continue the dialogue smoothly
- PB – possible breakdown: it is difficult to continue the dialogue smoothly
- B – breakdown: it is difficult to continue the dialogue

For instance, consider the piece of a dialogue taken from a TickTock (TKTK)[5] dataset:

S_1:	I felt I was 17 again.	(NB)
U_1:	Why's that?	
S_2:	Do you want to go to my party tonight?	(PB)
U_2:	Where are you from?	
S_3:	That is a lot to assume. Any thoughts?	(B)
U_3:	Not right now.	

In the example, user utterances are marked with U and system utterances are marked with S. In this case, according to the golden labels, the first system utterance does not cause any breakdown, whereas the second system response has a potential

[4]WOCHAT: Workshops and Session Series on Chatbots and Conversational Agents URL: http://workshop.colips.org/wochat/data/index.html, last checked on 07-01-2019.

[5]TickTock 100 URL: https://github.com/dbd-challenge/dbdc3/raw/master/data/dev/ticktock_100.zip, last checked on 07-01-2019.

to cause a breakdown. At the same time, after the third system response it is difficult to continue the conversation.

Hence, we define dialogue as a sequence of utterances. For accomplishing the task, each system utterance should be appropriately labelled. There are multiple labels available. Hence, we define the task as multinomial sequence classification.

1.2 Related Literature

The dialogue breakdown detection task was first introduced in [47]. Up to the current moment, three challenges were held[19, 20].

Dialogue breakdown detection can be used to re-rank responses of a chat-oriented dialogue system. In [25], the authors suggest three re-ranking approaches: classification, regression and probability-based approach. The classification technique was based on the classification of all the possible system responses. The regression method implied application of linear regression with the probability distribution of breakdown labels and response scores as a feature set. In case of the probability-based approach, the non-breakdown probability was used for re-ranking.

Other application examples include [50] where the author applies dialogue breakdown detection system for selecting tweets that can be used as responses of a chat-oriented dialogue system.

Another direction of research in the field is to investigate the errors causing a breakdown in chat-oriented dialogue systems. This is done in [18] where researchers present a taxonomy of this type of errors. Inspired by the Gricean maxims[6], the authors define utterance-level, environmental-level and cooperativeness error. The breakdown detector based on the taxonomy of errors is presented in [22].

Breakdown detection is not the only way to evaluate chatbot responses. For instance, [61] offer a similar technique for assessment of chatbot responses in non-task-oriented dialogues. In particular, they suggest to measure the appropriateness of utterances and customer satisfaction.

1.3 Research Questions & Contributions

Throughout the internship, the following questions were addressed:

Q_1**: What approaches to breakdown detection are available?** It was important to consider the models that already exist in the field. It helps to choose a baseline model and discover what methods work best. Besides, it prevents duplicating the work that has already been done.

Q_2**: What model is the most suitable for breakdown detection?** Answering this question helps to narrow down the search space and to define a more specific direction in which further research can develop.

Q_3**: Does the architecture of the model influence its performance?** Even after we define a general type of model suitable for the task, the space of all

[6]the principles of effective communication in standard social setting[17]

the possible model architectures is still rather big. Hence, we would like to to look deeper into different architecture options within the selected approach and find out if there is a relationship between architecture type and model performance in the task setting.

Q_4: **Does the embedding type of the detector influence its performance?** Another significant part of any model working with text is text representation. Therefore, we would like to explore different embedding types and investigate a relationship between embedding type and model performance if there is any.

In providing response to the questions mentioned above, the findings contribute to defining the major type of models which can be used for dialogue breakdown detection. Besides, the approach with the most promising results is investigated deeper. In particular, we find out what type of model architecture and word embedding type produce positive influence on the model's performance. Moreover, we introduce models for dialogue breakdown detection which outperform the baseline.

1.4 Outline

The report is organized as follows: in the chapter 2, we present the datasets and methods we use for the project. In the section 2.1, we describe the dataset for model training and evaluation. In the section 2.2, we present an overview of the methods used in the field and select the model for experiments. In the section 2.3, we give an overview of the embedding types. We choose several embedding type for incorporation into the model.

In the chapter 3, we discuss model implementation. The chapter comprises two major parts. In the first part (section 3.1), we describe the dataset preprocessing steps and tensors design. In the second part (section 3.2), the complete models' architecture and training procedure are discussed.

The chapter 4 is dedicated to model evaluation. In particular, we present the metrics for model assessment (section 4.1) and discuss the models' performance (section 4.2.

In the last chapter (chapter 5), we discuss the obtained results in connection to the research questions (section 1.3). We conclude the report by outlining the work that can be done in the future.

Chapter 2

Data and Methods

The first step for building a dialogue breakdown detector is to define the architecture of the model and to find or create a dataset suitable for the model's training and evaluation. This is the topic of the chapter.

First, we look at all the datasets suitable for the task. Fortunately, the dataset from the latest breakdown detection challenge, DBDC3[1] is available to public. Hence, we use it in the project. In the section 2.1, we give an overview of the dataset, in particular, the sources from which the dataset dialogues were collected, the format of the dataset, its annotation, and the dialogue structure. Besides, we delve deeper into dataset statistics and point out the observed labelling issues.

Second, in the section 2.2, we look into the existing dialogue breakdown detection models. After analyzing the models described in the literature, we define major types of implemented models and compare their performance. We do it in order to select a baseline and decide on model architecture for further experiments.

Another important aspect to consider when working with textual data is text representation. For this reason, in the section 2.3, we give an overview of the existing methods for distributed representation of words and select embedding approaches for the experiments.

2.1 Dataset

The English DBDC dataset comprised four different subsets: IRIS, TKTK, CIC, YI.

The IRIS dialogues were collected from Informal Response Interactive System. The system's details can be found in [2]. The TKTK dataset

The IRIS dialogues is a subset from the WOCHAT dataset. Initially, the dialogues were collected from Informal Response Interactive System. The system's details can be found in [2]. Just like IRIS, the TKTK dataset was also sampled from the WOCHAT dataset. The dialogues were gathered from a TickTock dialogue system described in [59]. The CIC dialogues were collected during the Conversational Intelligence Challenge (CIC). The YI dialogues were assembled by using a chatbot

[1]dbdc3: Dataset, URL: https://dbd-challenge.github.io/dbdc3/datasets, last checked on 07-01-2019.

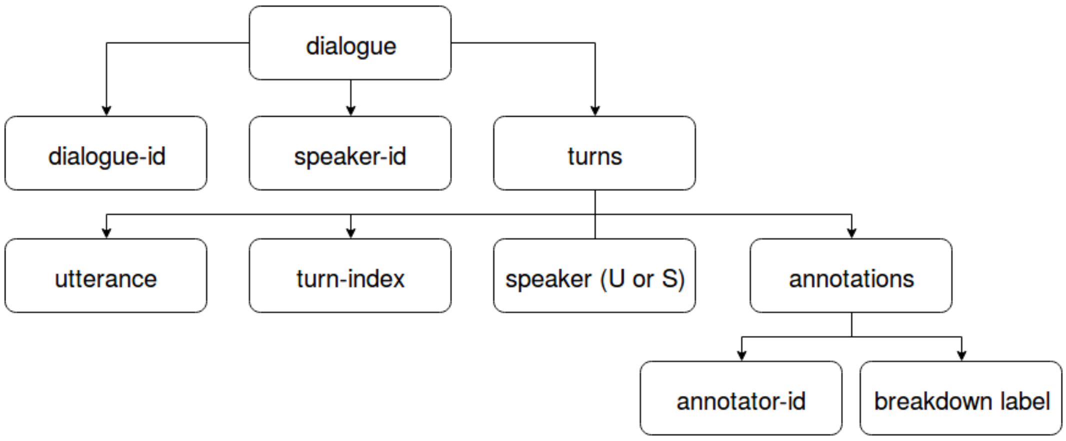


Figure 2.1: Dialogue representation in JSON: fields hierarchy

named Yura and Idris. It was participating in CIC and was developed by a team from Moscow Institute of Physics and Technology. The speakers from this dialogue were from the crowd-sourcing platform Amazon Mechanical Turk (AMT)[2].

Each dialogue is represented in JavaScript Object Notation (JSON) format. The dialogues characters are encoded with 8-bit Unicode Transformation Format (UTF-8). The fields hierarchy of dialogue representation in JSON is illustrated in figure 2.1.

Each dialogue is a sequence of utterances. The utterances can be of two types: user utterance and system response. The user and system generally exchange utterances in turns. We define the dialogue length as the number of utterances it has. Each dialogue in a dataset has a length of 20 or 21 utterances. For development set, $49,6\%$ dialogues have length 20, $50,4\%$ dialogues have length 21. In the evaluation set, 51% dialogues have length 20, 49% dialogues have length 21. This is due to the fact that sometimes a dialogue starts or ends with two user or system utterances in a row.

Each system utterance is labelled by thirty annotators. As was mentioned in the introduction (section 1.1), there are three labels available: 'NB' – not a breakdown, 'PB' – possible breakdown, 'B' – breakdown. Since the labelling is highly subjective, for each system utterance we define a golden label as the majority label.

The TKTK and IRIS dialogues were annotated by crowd workers from Crowd-Flower (currently Figure Eight)[3], whereas the CIC and YI datasets were annotated by crowd workers from AMT. In both cases, non-native speakers were asked to abstain from the task, however, this cannot be ensured.

The general dataset statistics are presented in the table 2.1. It can be noted, that the datset is split into training set and evaluation set. In total, the training set contains 415 dialogues, the evaluation set consist of 200 dialogues.

Since we work with a classification problem, it is important to consider how balanced our classes are. Operating with imbalanced dataset under the assumption that it is balanced leads to poor model performance[26]. The labels counts are

[2] Amazon Mechanical Turk, URL:https://www.mturk.com/, last checked on 07-01-2019.
[3] Figure Eight, URL:https://www.figure-eight.com/, last checked on 07-01-2019.

Table 2.1: The dataset statistics

	Development data				Evaluation data			
	TKTK	IRIS	CIC	YI	TKTK	IRIS	CIC	YI
Dialogues	100	100	115	100	50	50	50	50
Annotators	CF	CF	AMT	AMT	CF	CF	AMT	AMT
NB	35.1%	32.9%	28.9%	34.8%	44.3%	34.5%	29.1%	35.4%
PB	27.6%	27.8%	29.8%	36.1%	29.2%	29.3%	39.3%	40.3%
B	37.3%	39.4%	41.3%	29.1%	26.5%	36.2%	31.6%	24.3%
Fleiss' κ	0.14	0.11	0.05	0.01	0.13	0.09	0.001	-0.006

represented in the table 2.1 give us a reasonable idea concerning the class balance. However, we decided to make a general count of labels per training and evaluation datasets to ensure class balance. The development dataset contains 4150 labels in total, $34,7\%$ of them are labelled as NB, $43,7\%$ – PB, $21,6\%$ has label B. The evaluation set contains 2000 labels, $37,8\%$ of them are labelled as NB, $24,9\%$ – PB, $37,3\%$ – B. Overall, we conclude that the dataset classes are balanced.

Another important aspect to consider when working with labelled dataset is annotators agreement. Since each system utterance in the dialogues is labelled by thirty different annotators, we measure annotators agreement with Fleiss' κ. The measure is computed as follows:

$$\kappa = \frac{\bar{P} - \bar{P}_e}{1 - \bar{P}_e} \tag{2.1}$$

where $\bar{P} - \bar{P}_e$ represents the degree of agreement which was achieved above chance, whereas $1 - \bar{P}_e$ is a degree of agreement achievable above chance[10]. If $k = 1$, the annotators are in total agreement, whereas $k < 0$ signifies lack of agreement.

As can be seen in the table 2.1, κ is relatively low for the CIC and YI dialogues. In both cases, the dialogues were annotated by AMT workers. An analysis carried out in [20] allows concluding that it was due to the fact that some annotators used the same labels for most of the utterances. Removing such labels helped to increase the Fleiss' κ coefficient up to 0.1-0.2 for both dialogue subsets.

2.2 Methods Overview

After discussing the model's dataset, we proceed by presenting an overview of the dialogue breakdown detection methods. We group the methods into several classes based on the model type and compare the approaches. The survey is meant to help us understand what approaches work better than others in the field. Besides, it also precludes duplication of work.

2.2.1 Approaches

Following the analysis of the existing literature in the field, we define several types of models used for breakdown detection. The list includes Conditional Random Fields model (CRF), Extremely Randomized Trees (Extra Trees Regressor, ETR), Maximum Entropy model (MaxEnt), Support Vector Machines (SVM), Memory Networks (MemNN) and Recurrent Neural Networks (RNN). Besides, some systems feature attention modules as part of their architecture. Below, we give an overview of the models and compare their performance.

CRF One of the available approaches is to implement a CRF model. The model was first introduced in [31]. CRF was considered a baseline during DBDC3[47]. The model was making predictions on the words in target utterances and previous utterances as features.

ETR The other type of models was an ETR model[13]. This ensemble approach of a tree type was implemented in [29]. The authors calculate utterance similarities between adjoining utterances using term frequency vector and word embeddings. They present several models in their work. For the first model, authors employ geometric mean and maximum cosine similarity as features, whereas for the second model they experiment with arithmetic mean. In addition, they implement a model which uses cosine similarities of all the pairs of terms from the neighbouring utterances.

MaxEnt Another type of dialogue breakdown detector is MaxEnt classifier[39]. In [20] a MaxEnt-based breakdown detector was implemented as follows: each dialogue was represented as a sequence of exchanges, i.e., $(\textit{user utterance}, \textit{system response})$ pairs. All the dialogues are fed into the classifier which labels each system response.

SVM The dealogue breakdown detection system can also be SVM-based. In [33], authors present an SVM with a SpeDial feature set[36].

MemNN Besides, MemNN can be used to approach the task. In [24], the team implements character-level MemNN. Additionally, they use a convolutional neural network (CNN) for the purpose of memories reduction and as a part of the attention module. The network was trained with stochastic gradient descent (SGD). The authors present two models. The first model served as a dialogue breakdown detector of English First model was trained on Japanese dataset, whereas the second model was multilingual as the team expanded the dataset set by adding English data to it.

RNN The biggest number of dialogue breakdown detector models were of RNN type. For instance, in [40], the authors present an RNN model with attention. They use global vectors (GloVe) embeddings pretrained on Twitter data due to the proximity of the domains. The authors transform every utterance into a sentence embedding and operate on two main representations of the dialogue: target system utterance and the context vector. The context embedding and the target system utterance are fed to a feedforward neural network to obtain a prediction.

A subtype of RNN model is a **long short-term memory network** (LSTM). One of the examples of implementation of an LSTM model for dialogue breakdown detection is presented in [33] where authors demonstrate an LSTM with pretrained word2vec Google News embeddings. Another example of LSTM-based dialogue breakdown detector is demonstrated in [58]. There, authors create a hierarchical bidirectional LSTM (Hie-Bi-LSTM). They concatenate user utterances with the corresponding system responses thereby forming exchanges. Each word in exchange is represented with pretrained GloVe embeddings[4]. Next, an exchange is fed to a bi-directional LSTM encoder to obtain utterance representation. Afterwards, utterance representations to the second bidirectional LSTM to predict breakdown labels.

Models with attention Moreover, some dialogue breakdown detection models feature an attention module. For example, in [24], the MemNN's attention module is implemented with CNN and 'attention over attention'[7]. Besides, in [40], the team implements RNN with attention between sentences. Attention layer produces weights for the context vector. The weights help model to understand which parts of the context worth attention.

2.2.2 Model Selection

This section describes the comparison of the methods for dialogue breakdown detection and the process of model selection.

Comparison of the methods The results of the models' performances grouped by type are presented in table 4.1. We do not include models with attention as a separate model type in the table. This is done to avoid repetition as all the models with attention also belong to the other groups of models.

As can be seen, the highest accuracy was achieved by LSTM model with Google News embeddings. At the same time, the best $F_1(B)$ and $F_1(PB + B)$ scores were attained with MemNN with attention model. Therefore, we will consider these two models as the baseline.

Selected model After analyzing the models, we turn to the selection of the type of models we want to investigate further. Our choice fell on RNN type because all the models from this category demonstrate good performance both in terms of accuracy and F_1. Besides, the model with the best accuracy is also an RNN. The other argument in support of RNN is their ability to process sequential data. Since we framed the task as sequence classification, RNN seems like the optimal choice.

[4]Glove Wikipedia 2014 100d, URL: https://nlp.stanford.edu/data/glove.6B.zip, last checked on 07-01-2019.

Model type	Implementation description	Accuracy	$F_1(B)$	$F_1(PB+B)$
CRF	baseline model for DBDC3	0.420	0.354	0.762
MaxEnt	*MaxEnt, labelling exchanges	0.410	0.240	0.220
SVM	SVM + SpeDial feature set	0.340	0.350	0.840
MemNN	MemNN + attention	0.295	**0.364**	**0.874**
	MemNN + attention; trained on multiligual data	0.290	0.356	0.870
ETR	ETR + geometric mean	0.426	0.312	0.832
	ETR + cosine similarity of all word pairs	0.431	0.320	0.840
	ETR + arithmetic mean	0.420	0.302	0.835
RNN	RNN + attention between sentences, GloVe Twitter	0.360	0.208	0.346
	RNN + attention + GloVe Twitter + finetuning	0.210	0.210	0.340
	RNN + attention + GloVe Twitter + extra linguistic features	0.356	0.320	0.805
	LSTM + BoW, word embeddings	**0.440**	0.290	0.744
	LSTM + BoW, document embeddings	0.422	0.340	0.759
	Hie-Bi-LSTM + GloVe Wikipedia	0.429	0.321	0.763

Table 2.2: Models overview: classification results. The best performance is indicated in bold. An asterisk signifies submission without technical paper

However, standard RNN cannot deal with the problem of long-term dependencies and have some training issues, e.g., vanishing and exploding gradients[3]. Hence, we decided to use LSTM. The neural network was initially proposed by [21], later extra enhancements such as 'the forget gate' were offered in [12].

The LSTM cell is structured as follows. It has three types of gates and three weights. The gates are input, output and forget gates. They respectively filter input and output information as well as the information the cell should forget. The LSTM cell weights are input and output weights and internal state. The input weights are applied to the current time step, the output weights are weighing the output from the previous time step and the internal state is applied to the output of the current time step [27].

There are several types of LSTM available: vanilla, stacked, CNN, encoder-decoder, bidirectional and generative. Below, we discuss each LSTM type.

We define *vanilla LSTM* as a shallow network containing one layer of LSTM cells. This type of network proved to be useful when working with simple sequence classification or sequence prediction tasks, e.g., [57].

The second architecture type is *stacked (deep) LSTM*. As can be guessed from the name, this type of model has more than one LSTM layer. The approach proved to be useful when dealing with more cases where a more 'complex' representation might fit better, for instance, [62].

The next type of LSTM model is *CNN LSTM*. It can be used for extracting features from input sequences and generating sequences as output. This kind of model consists of two sub-models, a CNN on the front end and an LSTM. CNN acts as an encoder, whereas LSTM is used for sequence generation. CNN LSTM can be used, for instance in image description, video description, or activity recognition[9].

Encoder-decoder model is another LSTM type. This kind of network consists out of two parts: the first part is an encoder and the second part is a decoder. The encoder takes a sequence of a variable length as input and maps it to an embedding of fixed dimensionality. The decoder network, in turn, maps the embedding to a target sequence of a variable length[5]. The encoder-decoder is usually applied to statistical machine translation[5], conversation modeling[48] or image captioning[34] tasks.

The other LSTM architecture type is *bidirectional* LSTM. This type of model scans a sequence in two directions: from left to right (forward LSTM) and backwards (backward LSTM). In other words, the model employs information not only about the previous part of the sequence but also about the parts of the sequence coming afterwards[16]. This type of LSTM was initially created for speech recognition[15] but currently it is used in many other language-processing tasks such as sequence tagging[23] and named entity recognition (NER)[4].

Generative LSTM is the last type of LSTM discussed in the report. The network is trained on a given corpus. The objective is to learn the structural properties of the training data and to generate an output which would be representative of the training data. The network addresses the problem of language modelling [51, 14].

After considering the existing LSTM architectures, we decided to run a series of experiments with three types of model: vanilla, stacked and bidirectional LSTM.

2.3 Word Embedding Models

In this section, we discuss word embedding models. First, we give an overview of existing approaches to distributer representation of words. Next, we select the word embedding types for implementation of dialogue breakdown detector.

2.3.1 Overview

One of the earliest times the idea of representing a word as a vector was proposed in [46]. The concept was developed further and two principle methods of approaching the problem emerged: matrix factorization and local context window[41]. The example of the first method is latent semantic analysis[32], the second group includes `word2vec`[5] models such as skip-gram and continuous Bag of Words (CBOW)[37].

Both approaches have its advantages and disadvantages. For instance, matrix factorization methods are generally capable to leverage statistical information about the corpus from which representation was learnt. However, they do not perform very well in analogy task. On the contrary, the local context models do a better job with analogy task but cannot leverage corpus statistics. In an attempt to unify advantages of both approaches, GloVe model was presented. Representations obtained with GloVe showed increased performance in such tasks as NER, word similarity and word analogy[41].

One of the most recent advances in the field of word representation is the Embeddings from Language Model (ELMo)[42]. ELMo embeddings possess several unique properties. First, an ELMo's word embedding depends on context. That implies the ability of the model to handle polysemy. Second, the representations are deep, meaning that they combine all the hidden layers of the bidirectional language model they were learned from. Third, the embeddings are on the character level, what allows to leverage morphological cues for creating representations[11].

2.3.2 Embedding Selection

After looking at the available word embedding types, we turn to the selection of the word embeddings for the experiments. We consider only pretrained word embeddings due to limited data.

The first type of the word representations we use is the word2vec vectors pretrained on Google News corpora[6]. We use those vectors because they were featured in the LSTM model which serves us as accuracy baseline. The vectors were produced by a bag-of-words model (BoW) trained with negative sampling with window size 5. Each word is represented with an embedding of size 300.

The second type of word representation we use is GloVe trained on Twitter data[7]. We use this embedding type because of the proximity of the Twitter domain to the task domain[40]. The vectors were obtained by training on 2 billions of tweets with representations for 27 billion tokens and a vocabulary of 1,2 million. The vectors are presented in 25d, 50d, 100d and 200d, we decided to use 200d vectors. The words are uncased.

The third type of word embedding model is GloVe Common Crawl[8]. It contains representations for 840 billion tokens with the vocabulary of 2.2 million. The words are cased and the vectors dimensionality is 300d. We use this embedding type because it, unlike the GloVe Twitter, is domain-independent.

[5]word2vec, URL: `https://code.google.com/archive/p/word2vec/`, last checked on 07-01-2019.

[6]Google News 100B 3M words, URL: `https://github.com/3Top/word2vec-api`, last checked on 07-01-2019.

[7]GloVe: Twitter, URL: `https://nlp.stanford.edu/data/glove.twitter.27B.zip`, last checked on 07-01-2019.

[8]GloVe: Common Crawl, URL: `https://nlp.stanford.edu/data/glove.840B.300d.zip`, last checked on 07-01-2019.

2.4 Conclusions

In the chapter, we described the dataset which will be used for model training and evaluation.

Besides, we defined two baseline models. One model is an accuracy baseline, the other model is a baseline for F_1 scores. In addition, we selected a neural network type for dialogue breakdown detector and chose three architecture types for further experiments: vanilla LSTM, stacked LSTM and bidirectional LSTM. We plan to combine each LSTM architecture with three embedding types: word2vec pretrained on Google News corpora, GloVe pretrained on Twitter data and GloVe pretrained on Common Crawl. Hence, we intend to create nine different models in total.

Chapter 3

Model Implementation

After deciding on models' architecture and dataset, we can proceed by creating the models.

Implementation comprises two major steps. The first step is dataset preprocessing. It entails preparing the dataset for the model training and evaluation. The data is represented as tensors. For each model, we prepare two pairs of tensors, one pair for model training, one pair for model evaluation. Each pair of tensors consist of an input tensor with dialogue representations and a target tensor with corresponding labels.

The second step is definition, compilation and training of the models. We define models based on the discussion results from the previous chapter. Besides, we discuss such topics as cost function, optimization function and overfitting.

3.1 Data preprocessing

In this section, we discuss dataset processing and describe the process of tensors design.

3.1.1 Text Preprocessing

Before starting dataset processing, we had to decide whether it is needed to include user turns or not. The findings represented in [33] allow to conclude that including users turns in the training set helps to improve the accuracy of a dialogue breakdown detection system. Therefore, we choose to keep user turns.

If we keep the user turns in the dataset, how do we feed them? Initially, we considered concatenating user turns with the corresponding system response into exchanges pairs. Such approach would allow to avoid introduction of an extra label. However, results shown in [58] testify that such concatenation decrease model performance. Hence, we resolve to mark each user utterance with an extra label U and feed them to the models as a separate utterance.

After making the decisions, we can start dataset preprocessing. Since we use three different types of embedding models, we prepare one dataset for each of the types. We do it so that the particularities of the embedding type are taken into account. However, two major preprocessing steps were applied to all three datasets. The steps are tokenization and replacement of apostrophe contractions.

In general, all the datasets are tokenized with `TweetTokenizer`[1]. This tokenizer is a part of `casual` submodule of `nltk.tokenize` package, it was selected because the domain of its primary use is closely related to the domain of the task. In particular, `TweetTokenize` is able to handle emoticons the dataset contain.

In addition to tokenization, extra rules are applied to common apostrophes contractions. For example, the contraction "$that's$ is transformed to "$thatis$".

Besides the mentioned preprocessing steps, we remove punctuation signs from the dataset for word2vec Google News vectors because the model did not know any punctiation signs. Additionally, we lowercase all the words in the dataset for GloVe Twitter because the pretrained word vectors are uncased.

As mentioned in the section 2.1, dialogues length varies in both development and evaluation sets. In such a case, we can either truncate or pad the dialogues. Since the first approach implies a loss of certain parts of data, we consider padding as a more suitable option.

3.1.2 Representations

The following step of data preparation is obtaining word, utterance, dialogue and label representations. It is important to note that pretrained vectors we use for the experiments do not have the same dimensionality. Both word2vec vectors pretrained on Google News and GloVe vectors pretrained on Common Crawl have dimensionality of 300, whereas the dimensionality of GloVe vectors pretrained on Twitter data is 200. The difference impacts the shape of representations and tensors.

Word representations are acquired with pretrained word embedding vectors. We represent all the unknown words with the token unk.

Each sentence embedding is represented as the average of the token embeddings that comprise the sentence. Consequently, each utterance embedding has the same dimensionality as the word embedding.

A dialogue is represented as a sequence of user and system utterances. We pad dialogues to ensure that each dialogue has the length of 21 utterances. Therefore, each dialogue is represented as 21×300 matrix for representations created with word2vec Google News vectors and GloVe Common Crawl vectors and 21×200 for representations built with GloVe Twitter vectors.

Each system response in a dialogue is labelled with B (breakdown), PB (possible breakdown) or NB (not a breakdown) label. Besides, we introduce label U to mark user utterance. The labels are represented with one-hot encoding. Therefore, each sequence of dialogue labels is represented as 21×4 matrix.

[1] NLTK 3.4 documentation, URL: `http://www.nltk.org/api/nltk.tokenize.html`, last checked on 07-01-2019.

3.1.3 Tensors

After describing dialogue representation, we proceed by designing tensors for model training and evaluation.

We represent the dataset as four tensors. The development input tensor contains dialogues used for training. We build this tensor by stacking dialogue utterance representations on the top of each other. There are 415 dialogues in the training set, each dialogue is represented as 21×300 or 21×200 matrix. Hence, the resulting shape of the development input tensor is $415 \times 21 \times 300$ or $415 \times 21 \times 200$ depending on the pretrained word vectors used for its creation.

The second tensor is the development target tensor. It contains labels for the corresponding dialogues of the training set. The tensor is built by stacking dialogue labels representations. Since each sequence of labels is represented with one-hot encoding, the final shape of the tensor is $415 \times 21 \times 4$.

Similarly to the above-mentioned method, we obtain evaluation input and target tensors by stacking dialogues and labels representations. Since the evaluation dataset consist out of 200 dialogues, the resulting input tensor has a shape of $200 \times 21 \times 300$ or $200 \times 21 \times 200$, whereas the resulting target tensor's dimensionality is $200 \times 21 \times 4$. The input and target tensors representations are shown in the figure 3.1.

3.2 Model development

In this section, we discuss the software we use for models implementation, describe the models' architecture and the training procedure.

3.2.1 Software

For the project, we chose `Python 3.6` because it provides all the libraries required for model implementation and evaluation: `NumPy`, `NLTK`, `gensim`, `matplotlib`, `sklearn`, `json`. The model was built with `Tensorflow`[1] as a backend and `Keras` machine learning library[6].

3.2.2 Models Description

In this subsections, we describe the models we create in more details.

Each vanilla LSTM has 4 layers with 64 cells in each. The first layer is an `LSTM`[2] layer, the next two hidden layers are `Dense`[3]. The output layer is `Dense`, too. Hidden layers have rectified linear unit (ReLU) activation function, whereas the output layer's activation function is softmax. The model is illustrated on figure B.1 (in the appendix B).

[2]Keras Documentation: Recurrent Layers, URL: https://keras.io/layers/recurrent/#lstm, last checked on 07-01-2019.

[3]Keras Documentation: Core Layers, URL: https://keras.io/layers/core/#dense, last checked on 07-01-2019.

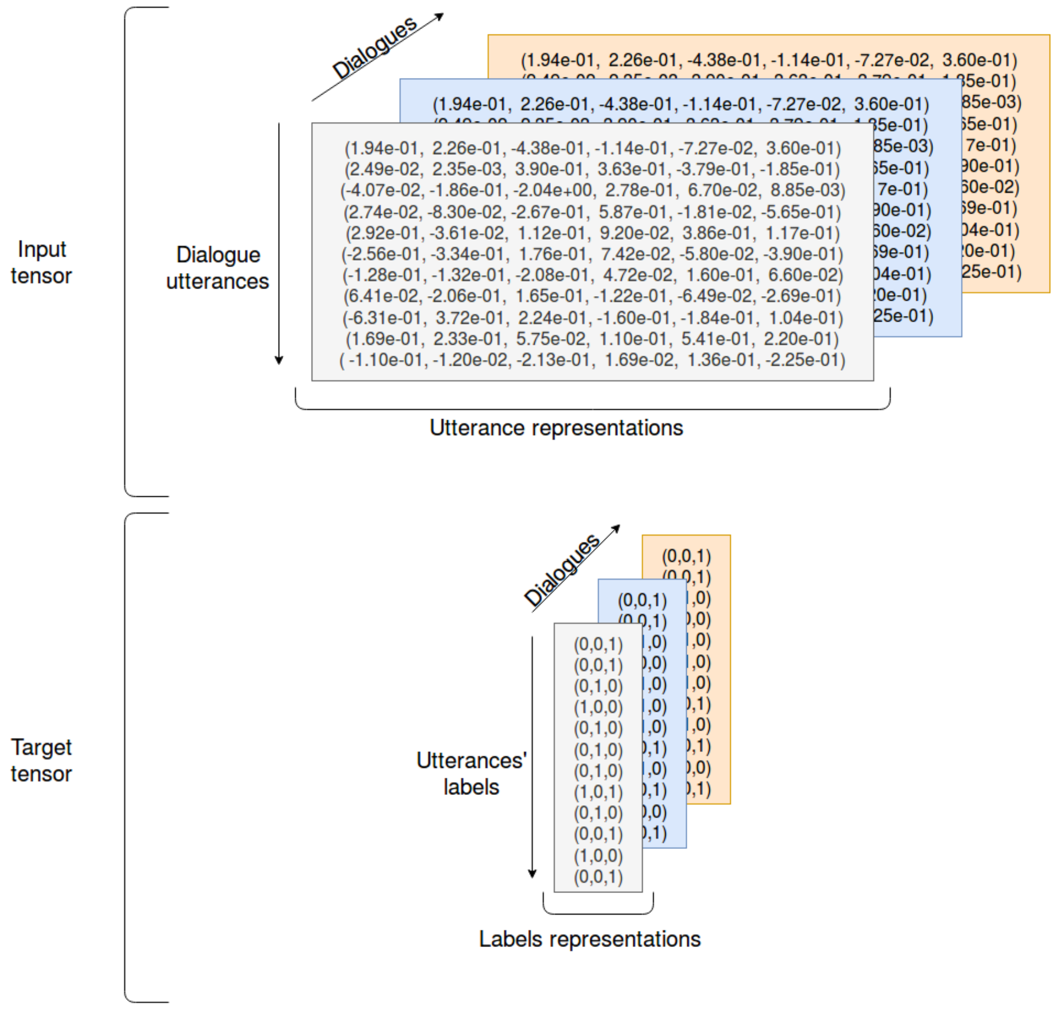

Input tensor
Dialogues
Dialogue utterances
(1.94e-01, 2.26e-01, -4.38e-01, -1.14e-01, -7.27e-02, 3.60e-01)
(2.49e-02, 2.35e-03, 3.90e-01, 3.63e-01, -3.79e-01, -1.85e-01)
(-4.07e-02, -1.86e-01, -2.04e+00, 2.78e-01, 6.70e-02, 8.85e-03)
(2.74e-02, -8.30e-02, -2.67e-01, 5.87e-01, -1.81e-02, -5.65e-01)
(2.92e-01, -3.61e-02, 1.12e-01, 9.20e-02, 3.86e-01, 1.17e-01)
(-2.56e-01, -3.34e-01, 1.76e-01, 7.42e-02, -5.80e-02, -3.90e-01)
(-1.28e-01, -1.32e-01, -2.08e-01, 4.72e-02, 1.60e-01, 6.60e-02)
(6.41e-02, -2.06e-01, 1.65e-01, -1.22e-01, -6.49e-02, -2.69e-01)
(-6.31e-01, 3.72e-01, 2.24e-01, -1.60e-01, -1.84e-01, 1.04e-01)
(1.69e-01, 2.33e-01, 5.75e-02, 1.10e-01, 5.41e-01, 2.20e-01)
(-1.10e-01, -1.20e-02, -2.13e-01, 1.69e-02, 1.36e-01, -2.25e-01)
Utterance representations
Target tensor
Dialogues
Utterances' labels
(0,0,1)
(0,0,1)
(0,1,0)
(1,0,0)
(0,1,0)
(0,1,0)
(0,1,0)
(1,0,1)
(0,1,0)
(0,0,1)
(1,0,0)
(0,0,1)
Labels representations

Figure 3.1: Input and target tensors representations. Input tensor consist out of stacked dialogues representations, target tensor comprises corresponding one-hot encoded labels

The second model's architecture type is a stacked LSTM. Just like the previous model, it has four layers, the only difference being that the second layer is also `LSTM`. The model is presented in the figure B.2 (in the appendix B).

The third model is a bidirectional LSTM. Like the previous models, it also has four layers, however, its first layer contains bidirectional LSTMs. In `Keras` this layer can be implemented with a `Bidirectional` wrapper[4]. The model's acrhitecture is illustrated in the figure 3.2.

We combine each of the three LSTM architectures with three different types of word embeddings: word2vec vectors pretrained on Google News corpora, GloVe

[4]Keras Documentation: Layer Wrappers, URL:`https://keras.io/layers/wrappers/#bidirectional`, last checked on 07-01-2019.

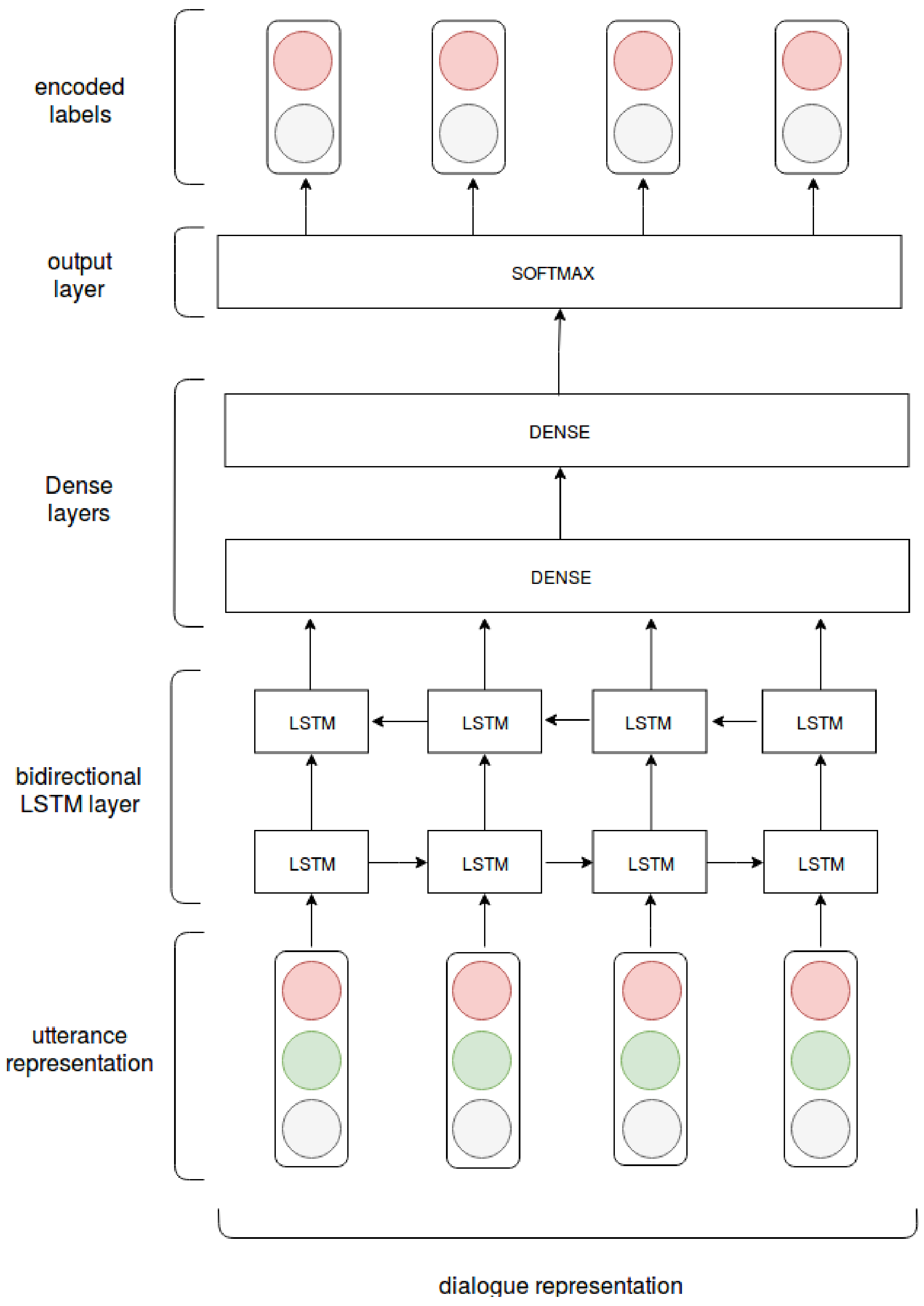


Figure 3.2: Bidirectional LSTM architecture

pretrained on Twitter data and GloVe pretrained on Common Crawl.

3.2.3 Training

In this section, we discuss the aspects of training the neural network such as loss function, model optimization algorithm, number of epochs and batch size and a potential problem of overfitting.

Loss Function Loss function type depends on the task the model has to complete. In the given case, we are working on labelling dialogue utterance, each turn can have one of the four labels: NB, PB, B, U. For each utterance in the dialogue, the network predicts a probability for each of the labels and compares it with the ground truth. Hence, the model should use a loss function that would compare the labels probability distribution with the ground truth and penalize incorrect label prediction. One of the suitable objective function for this task is cross-entropy function[8]. The function measures the difference between two probability distributions. The function is computed as follows:

$$H(y, \hat{y}) = -\sum_{i=1} y_i log(\hat{y}_i) \tag{3.1}$$

where y - ground truth label, $\hat{y}$ - predicted label.

Optimization Function We optimize models with gradient descent (GD). There are several types of GD to consider. Below, we discuss each approach and select the one which fits to the experiment setting better.

The first approach is batch GD, which calculates the gradient of the objective function after the entire training dataset fed into the model:

$$\theta = \theta - \eta \nabla_\theta J(\theta) \tag{3.2}$$

If the error surface is convex, batch gradient descent converges to a global minimum. However, this cannot be guaranteed if the error surface is non-convex[45]. Besides, this optimization type generally takes more time to converge, especially when working with big datasets[56].

The second option is stochastic gradient descent (SGD) which updates model parameters for each point from a training dataset $x^{(i)}$ and label $y^{(i)}$ [45]:

$$\theta = \theta - \eta \nabla_\theta J(\theta; x^{(i)}; y^{(i)}) \tag{3.3}$$

When processing large training sets, SGD converges faster than batch GD. On the other side, SGD is also more sensitive to the noisy data. This algorithm also tends to fluctuate more than batch GD what implies both advantages and downsides. Fluctuating behaviour might allow the model to converge to the global minimum which batch GD can overlook. At the same time, it can also lead to constant overlooking of the minimum.

The third principal option is mini-batch GD, which updates the model parameters for each mini-batch of size with k training datapoints:

$$\theta = \theta - \eta \nabla_{\theta} J(\theta; x^{(i:i+k)}; y^{(i:i+k)}) \tag{3.4}$$

Mini-batch GD combines advantages of both SGD and batch GD. On one hand, it is less computationally expensive than SGD. On the other hand, it is faster then batch GD.

After considering all the available options, we decided to use mini-batch GD. The size of a batch was set to 32. It implies that in every epoch (forward and backwards pass) 32 random training data points are sampled for training. The dataset is shuffled before sampling.

There exist several extensions of the standard GD optimization such as Nesterov accelerated gradient[38], Adaptive Moment Estimation (Adam)[30], Adadelta[60]. After considering all the available optimization algorithms, we decided to choose Root Mean Square Propagation (RMSProp)[52]. The optimization technique is an extension of Resilient Backpropagation (Rprop)learning[43]. It combines robustness of Rprop, the efficiency of mini-batches and the effective averaging of gradients over mini-batches.

Regularization To prevent overfitting, we employ two regularization techniques: early stopping and dropout. Early stopping monitors model performance on the validation set. The assumption is that overfitting occurs as soon as model performance on the training set continue to increase whereas its performance on the validation set does not improve. We implement early stopping[5] which monitors validation loss with patience 5, i.e., the model will stop after five consecutive epochs with no improvement. Another regularization technique is a dropout. The idea behind this technique is to randomly drop some of the network units during training. This is done to prevent 'co-adaptation' of the model units, i.e., a situation when a model relies on the presence of any specific unit for making predictions[49]. We applied both dropout[6] and recurrent dropout (for `LSTM` layers). In both cases, the coefficients were set to 0.1.

3.3 Conclusions

In the chapter, we explained dataset preprocessing and models implementation. In particular, we described the preprocessing steps we took for each embedding type and the shape of the resulting tensors.

Besides, we discussed the software we use for model development, models' architecture and the training procedure. All the models were trained in the same conditions. The categorical cross-entropy was used as a loss function, the model's

[5]Keras Documentation: Callbacks, URL:https://keras.io/callbacks/#earlystopping, last checked on 07-01-2019.

[6]Keras Documentation: Core, URL:https://keras.io/layers/core/#dropout, last checked on 07-01-2019.

parameters were optimized with RMSProp. The batch size was set to 32. The maximum number of epochs was 100. In practice, however, this number has never been reached due to early stopping.

Chapter 4

Evaluation

In this chapter, we discuss models evaluation. In particular, we describe evaluation metrics and present quantitative analysis of models performance. Next, the best-performing models are compared with the baseline and relationship between model architecture, word embedding type and model performance are explored. We conclude the chapter by an analyzing the best models' vulnerabilities and suggesting solutions to the identified issues.

4.1 Evaluation Metrics

Evaluation of classification model is based on the idea of a contingency table, where each cell represents a possible outcomes[28]. The table has two dimensions: 'gold standard labels' and 'system output labels'. With respect to one particular class, each instance can be classified as a member or non-member of the class. Therefore, four outcomes are possible: true positive (TP) – model correctly labels datapoint as belonging to the class; false positive (FP) – model labels datapoint whereas it does not belong to the class, true negative (TN) – model correctly rejects class membership; false negative (FN) – model rejects class membership, whereas the instance belongs to the class.

A classifier can be evaluated with such metrics as accuracy, precision, recall and F-measure.

Accuracy is a fraction of instances the model classified correctly:

$$Accuracy = \frac{TP + TN}{TP + FP + TN + FN} \tag{4.1}$$

The drawback of the accuracy metric is that it does not account for class imbalance.

For this reason, precision and recall are more common to use when evaluating classifier performance. Precision is the percentage of the relevant instances (TP) in the total amount of instances labelled as belonging to the class (TP + FP). Precision, therefore, can be calculated as follows:

$$Precision = \frac{TP}{TP + FP} \tag{4.2}$$

Recall measures how many of the items belonging to the class were labelled by the system as such. Recall is defined as follows:

$$Recall = \frac{TP}{TP + FN} \tag{4.3}$$

Both precision and recall are useful metrics for classifier performance as they emphasize the significance of TP. There are multiple ways how they can be combined into one metric. One of the most common examples is F-score[53]. The formula is given below:

$$F_\beta = \frac{(\beta^2 + 1)PR}{\beta^2 P + R} \tag{4.4}$$

The β coeffiecient is regulating the significance of precision and recall. $\beta < 1$ gives extra importance to precision, whereas $\beta > 1$ favours recall. When $\beta = 1$ precision and recall are equally relevant. In this case the metric is called F_1 or balanced F-score. Hence, the formula looks as follows:

$$F_1 = \frac{2PR}{P + R} \tag{4.5}$$

Since we do not have any indication towards precision or recall being more important, we choose F_1 as the metric.

Since we work with multinomial classification, we use a confusion matrix to assess the classifier performance. The matrix shows how many instances of a given class were classified correctly as well as how many instances were misclassified and to which other classes the instances were assigned. We calculate precision, recall and F_1 for each class separately and find their average.

4.2 Evaluation Results

After explaining the metrics we use for performance evaluation, we proceed by evaluating the models we built and choosing the models with the best performance in terms of accuracy and F_1 scores. Afterwards, we proceed by comparing the selected model(s) with our baseline models. The chapter concludes by a deeper analysis of the relationship between performance metrics and model architecture and embedding type.

4.2.1 Quantitative Analysis

In this part of the chapter, we discuss the scores models obtained, choose the best performing models and compare them with the baseline. Besides, we explore relationship between model architecture, embedding type and model performance.

Table 4.1: Models performance results. The best performance is indicated in bold

Model	Embedding	Accuracy	$F_1(B)$	$F_1(PB+B)$
Vanilla LSTM	Google News	0.42	0.38	0.84
	GloVe Twitter	0.41	0.31	0.78
	GloVe Common Crawl	0.44	**0.46**	**0.93**
Stacked LSTM	Google News	0.39	0.30	0.77
	GloVe Twitter	0.42	0.41	0.82
	GloVe Common Crawl	0.42	0.45	**0.93**
Bi-LSTM	Google News	0.42	0.33	0.75
	GloVe Twitter	0.44	0.34	0.83
	GloVe Common Crawl	**0.46**	0.37	0.85

Table 4.2: Comparison of the created models with the baseline models. The best performance is indicated in bold

Model type	Description	Accuracy	$F_1(B)$	$F_1(PB+B)$
Accuracy baseline	LSTM + word2vec Google News	0.44	0.29	0.74
Best accuracy model	Bi-LSTM + GloVe Common Crawl	**0.46**	0.37	0.85
F_1 baseline	MemNN + attention	0.29	0.36	0.87
Best F_1 model	Vanilla LSTM + GloVe Common Crawl	0.44	**0.46**	**0.93**

Results Overview

First, we evaluate the created models. We do this by using the metrics described in the previous section (section 4.1), namely accuracy and $F_1(B)$ and $F_1(PB+B)$. The results are shown in the table 4.1.

The best accuracy is achieved with Bi-LSTM with GloVe Common Crawl, however, the performance in terms of $F_1(B)$ and $F_1(PB+B)$ is attained by the *Vanilla LSTM with GloVe Common Crawl*. Therefore, we select Bi-LSTM with GloVe Common Crawl as the model with the best accuracy and Vanilla LSTM with GloVe Common Crawl as the model with the best $F_1(B)$ and $F_1(PB+B)$ scores.

Comparison with the Baseline Models

After choosing the best-performing models, we compare them with the baseline selected in the chapter 2. The comparison is shown in the table 4.2. As can be seen in the table both created models perform better than the baseline models.

Table 4.3: Average metric scores for every LSTM type

LSTM type	Accuracy	$F_1(B)$	$F_1(PB + B)$
Vanilla LSTM	0.42	0.38	0.85
Stacked LSTM	0.41	0.39	0.84
Bi-LSTM	0.44	0.35	0.81

Table 4.4: Average metric scores for every embedding type

Embedding type	Accuracy	$F_1(B)$	$F_1(PB + B)$
word2vec Google News	0.41	0.34	0.78
GloVe Twitter	0.42	0.35	0.81
GloVe Common Crawl	0.44	0.43	0.90

Relationship Exploration

After comparing our models with the baseline models, we move to the exploration of the relationship between performance and model architecture type and performance and embedding type.

In order to investigate which type of LSTM produces the best performance, we compare the metrics results. We do this by calculating the average metric score for each $Model \times Metric$ pair across three embedding types. The results are presented in the table 4.3. Overall, it can be concluded that, given that all the metrics have equal importance, the best performance is obtained by vanilla LSTM, stacked LSTM is the second best, and Bi-LSTM is the worst.

Next, we turn to the investigation of relationship between model performance and its embedding type. In analogy with the above-mentioned idea, we calculate an average performance score for each metric. The results presented in the table 4.4, allow to conclude that GloVe Common Crawl demonstrate the best performance, the GloVe Twitter being the second best, the word2vec Google News is the worst.

4.2.2 Qualitative Analysis

Apart from analyzing the models from the quantitative perspective, comprehension of patterns in the type of errors that the best models make could provide additional insight for their improvement.

Earlier in the chapter we showed that embedding type impacts model performance. In particular, during the experiments we found out that there is a positive correlation between the proportion of the tokens the embedding model knows in the dataset and model performance. Hence, we investigate the type of tokens the model does not know. In general, GloVe pretrained on Common Crawl does not know 883 tokens in the dataset (or 375 unique tokens).

Emoticons One of the biggest part of unknown tokens were emoticons. The emoticons are the strong cues how the user feels and hence can help to understand when the conversation goes to the wrong direction. Therefore, they should be taken into account. The model knows basic emoticons such as ☺ or ☹ but is not aware of more complex ones such as 😵 or 😎. Therefore, it might be helpful to replace the complex emoticons with their basic counterparts to facilitate the model's understanding.

Apostrophe Contractions Many of the tokens were not recognized because they represented apostrophe contractions. The examples included both straightforward cases like *isn't* (*is not*) and ambiguous situations such as *he's* which depending on context can be interpreted either like *he has* or *he is*. In general, the problem can be resolved by introduction of extra contraction replacement rules. In the case of ambiguous situations, it might be better to apply a disambiguation procedure first.

Abbreviations Another significant group of unknown tokens include abbreviations, e.g., *ConvAI* (*conversational Artificial Intelligence*). The issue can be addressed by abbreviation expansion. For instance, we could create a dictionary containing most common abbreviations.

Misspellings Some other words are not recognized because of misspelling. The most common type of mistakes is skipping white-space and thereby merging words. E.g., 'questionWho' and 'wait.Where'. Splitting these words while preprocessing the datasets can help to address the issue. Moreover, there were standard spelling mistakes such as 'seee' or 'tommorow'. Such standard mistakes can be resolved by adding a spellchecker.

Extra Besides, another type of unknown tokens include meaningless sequence of characters (for example, *MAMAXMAMAXMMAMAXMA*) or tokens representing words from different language. These type of tokens can be either replaced by *unk* or removed from the dataset. Further experiments are required to decide which approach is more suitable.

4.3 Conclusions

In the chapter, we evaluated the created models. We selected a model which showed best performance in terms of accuracy and a model with the best F_1 scores. Next, we compared the created models to the baseline models and came to conclusion that the models we had built performed better. Besides, we found out how model architecture and embedding type influence model performance.

Moreover, we presented qualitative analysis of the best-performing models. In particular, we investigated what kind of tokens the corresponding embedding model did not know and suggested solutions to the identified issues.

Chapter 5

Discussion & Conclusions

5.1 Summary

First, we explored the challenge dataset and gave an overview of existing models built for dialogue breakdown detection and compared their performance. Besides, we discussed word embedding types. (section 2)

Second, we discussed data processing and model development. In particular, we explained how the datasets were preprocessed and how tensors were built. Besides, we motivated and described the architecture of the neural network we were going to implement and explained the training procedure. (section 3).

Third, we discussed the metrics we used for the model evaluation and presented the evaluation results. (section 4). The general model pipeline is presented in figure 5.1.

5.2 Answers to research questions

Q_1**: What approaches to breakdown detection are available?** After analyzing the literature in the field, we defined several types of models for breakdown detection and grouped them into several categories: CRF, ETR, MaxEnt, SVM, MemNN, RNN and models with attention. We compared the models' performances in order to choose a baseline model.

Q_2**: What model is the most suitable for breakdown detection?** After analyzing the existing approaches, we defined the model that appears to be most promising for accomplishing the task and run a series of experiments to test our hypothesis.

Q_3**: Does the architecture of the model influence its performance?** After defining the most suitable model, we turned to investigating the possible model architectures. We run several experiments with different model architectures and compared the results. This was done to investigate the relationship between model architecture and its performance. We found out that some architecture types works

Q_4**: Does the embedding type of the detector influence its performance?** Another important part of any model working with text is text repre-

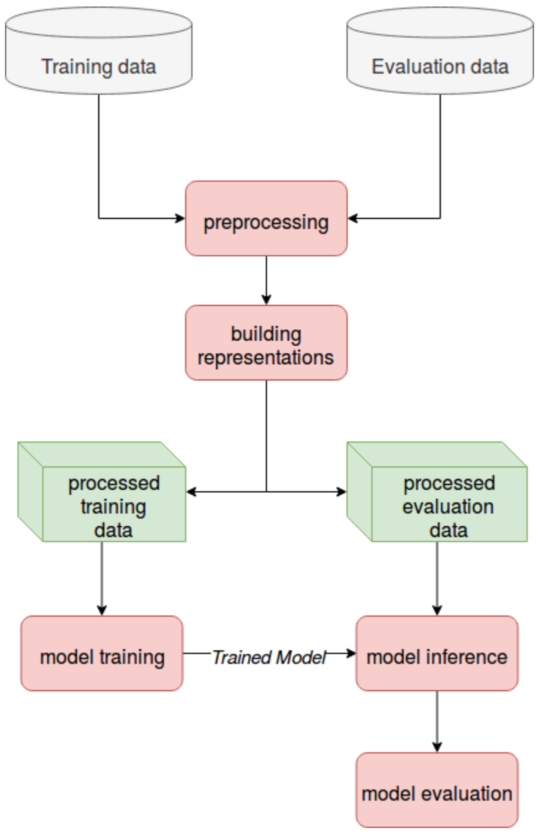


Figure 5.1: Model development and evaluation pipeline

sentation. There are different types of word embedding models available. We run experiments with different embedding type to find out if there is any relationship between embedding type and model performance.

5.3 Future Work

The future research in the field can be carried out in several directions.

First, it would be interesting to expand the dataset. The current dataset contains dialogues only in two languages – English and Japanese. It would be interesting to add extra dialogues to expand the current dataset so that it includes different languages and modalities.

Besides enlarging the dataset, we could think about improving architecture of the model. For example, we could investigate whether the performance of our models can be improved by adding an attention module. Moreover, we could experiment with hierarchical LSTM and attempt to improve the results presented in [58]. Apart from that, we could try to develop a character-level LSTM and experiment with multilingual data.

Besides, it is possible to explore how our model works with the the other embedding types. In the research, we explored three types of the word embedding such as word2vec with Google News, GloVe Twitter and GloVe Common Crawl. In the future, it would be interesting to try representing words with the embedding from EMLo to see how it affects the performance.

Lastly, from the perspective of empirical research, another possible direction to take is looking into the taxonomy of errors which cause dialogue breakdown. Such a taxonomy already exist for Japanese[18], however, there is no analogy for the English language yet. It would be interesting to compare English and Japanese taxonomies and implement a breakdown detector for English based on it.

Appendices

Appendix A

Models code

In the appendix, we demonstrate the code used for models definition, compilation and training. The following constants were used for all the models:

```
# constants
NUM_CELLS = 64
NUM_EPOCHS = 100
MAX_UTTERANCE_LENGTH = 21
BATCH_SIZE = 32
PATIENCE = 5
DROPOUT_COEFF = 0.1
UTTERANCE_DIMENSIONALITY = 300
```

A.1 Vanilla LSTM

```
# imports
from keras.models import Sequential
from keras.layers import LSTM, Dense

# model definition
vanilla = Sequential()
# input layer
lstm1 = LSTM(NUM_CELLS, input_shape=(MAX_UTTERANCE_LENGTH,
↪  UTTERANCE_DIMENSIONALITY), return_sequences=True, dropout =
↪  DROPOUT_COEFF, recurrent_dropout = DROPOUT_COEFF)
vanilla.add(lstm1)
# hidden layers
vanilla.add(Dense(NUM_CELLS, activation = 'relu'))
vanilla.add(Dense(NUM_CELLS, activation = 'relu'))
# output layer
outputs = Dense(NB_LABELS, activation = 'softmax')
vanilla.add(outputs)
```

```
# model compilation
vanilla.compile(
    optimizer = 'rmsprop',
    loss = 'categorical_crossentropy',
    metrics=['accuracy'])

# model training
vanilla_history = vanilla.fit(
    X_train, y_train,
    batch_size=BATCH_SIZE,
    validation_data = (X_test, y_test),
    callbacks = [EarlyStopping(monitor='val_loss', patience=5)],
    epochs=NUM_EPOCHS)
```

A.2 Stacked LSTM

```
# imports
from keras.models import Sequential
from keras.layers import LSTM, Dense

# model definition
stacked = Sequential()
# input layer
lstm1 = LSTM(NUM_CELLS, input_shape=(MAX_UTTERANCE_LENGTH,
↪   UTTERANCE_DIMENSIONALITY), return_sequences=True, dropout =
↪   DROPOUT_COEFF, recurrent_dropout = DROPOUT_COEFF)
stacked.add(lstm1)
# hidden layers
vanilla.add(Dense(NUM_CELLS, activation = 'relu'))
vanilla.add(Dense(NUM_CELLS, activation = 'relu'))
# output layer
stacked = Dense(NB_LABELS, activation = 'softmax')
stacked.add(outputs)

# model compilation
stacked.compile(
    optimizer = 'rmsprop',
    loss = 'categorical_crossentropy',
    metrics=['accuracy'])

# model training
stacked_history = stacked.fit(
    X_train, y_train,
    batch_size=BATCH_SIZE,
    validation_data = (X_test, y_test),
    callbacks = [EarlyStopping(monitor='val_loss',
    ↪  patience=PATIENCE)],
    epochs=NUM_EPOCHS)
```

A.3 Bidirectional LSTM

```
# imports
from keras.models import Sequential
from keras.layers import LSTM, Dense, Bidirectional

# model definition
bi_lstm = Sequential()
# input layer
lstm1 = Bidirectional(LSTM(NUM_CELLS, return_sequences=True, dropout
↪  = DROPOUT_COEFF, recurrent_dropout = DROPOUT_COEFF),
↪  input_shape=(MAX_UTTERANCE_LENGTH, UTTERANCE_DIMENSIONALITY),
↪  merge_mode = 'sum')
bi_lstm.add(lstm1)
# hidden layers
lstm2 = LSTM(NUM_CELLS, input_shape=(MAX_UTTERANCE_LENGTH,
↪  UTTERANCE_DIMENSIONALITY), return_sequences=True, dropout =
↪  DROPOUT_COEFF, recurrent_dropout = DROPOUT_COEFF)
bi_lstm.add(lstm2)
bi_lstm.add(Dense(NUM_CELLS, activation = 'relu'))
# output layer
bi_lstm = Dense(NB_LABELS, activation = 'softmax')
bi_lstm.add(outputs)

# model compilation
bi_lstm.compile(
    optimizer = 'rmsprop',
    loss = 'categorical_crossentropy',
    metrics=['accuracy'])

# model training
bi_lstm_history = bi_lstm.fit(
    X_train, y_train,
    batch_size=BATCH_SIZE,
    validation_data = (X_test, y_test),
    callbacks = [EarlyStopping(monitor='val_loss',
    ↪  patience=PATIENCE)],
    epochs=NUM_EPOCHS)
```

Appendix B

Models Architecture

B.1 Vanilla LSTM

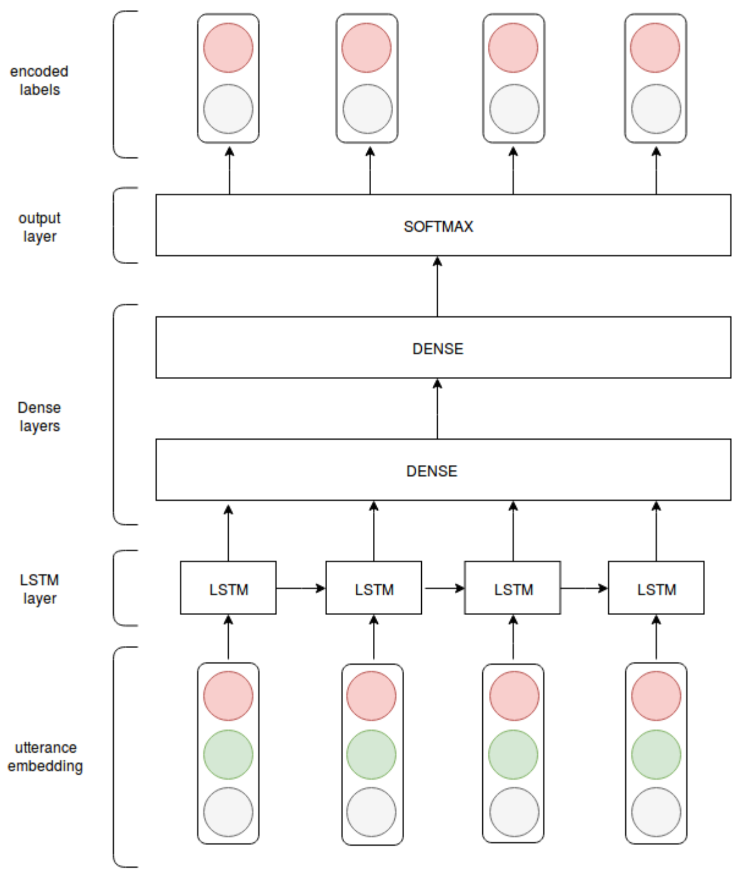


Figure B.1: Vanilla LSTM architecture

B.2 Stacked LSTM

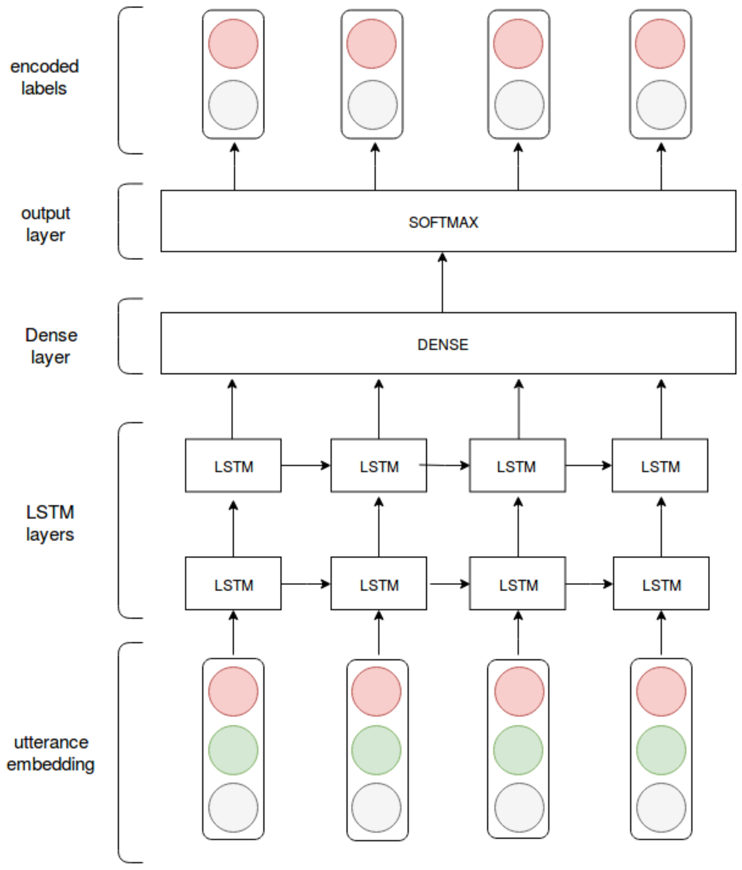
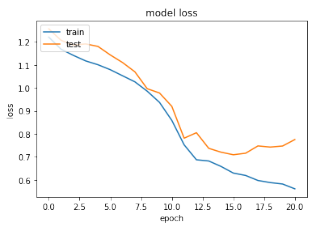


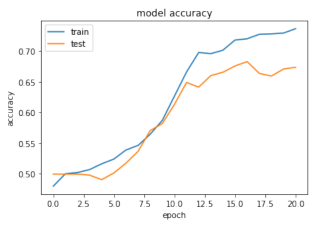
Figure B.2: Stacked LSTM architecture

Appendix C

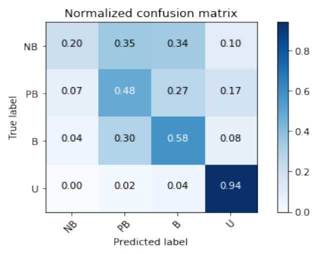
Models' training graphs



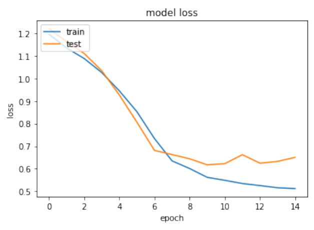
(a) Google News: Loss



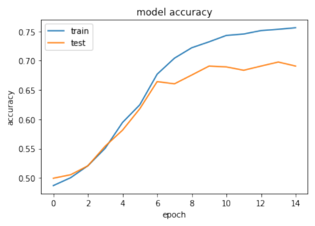
(b) Google News: Accuracy



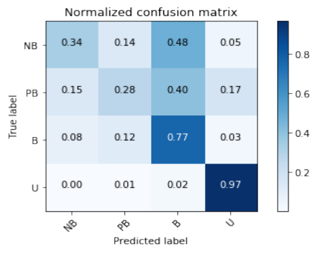
(c) Google News: Confusion matrix



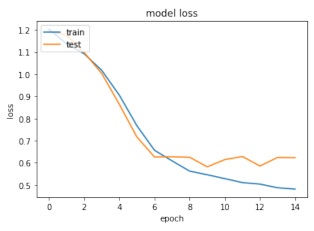
(d) GloVe Twitter: Loss



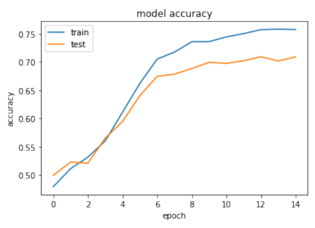
(e) GloVe Twitter: Accuracy



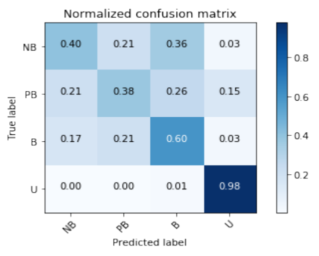
(f) GloVe Twitter: Confusion matrix



(g) Common Crawl: Loss

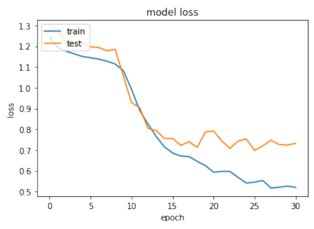


(h) Common Crawl: Accuracy

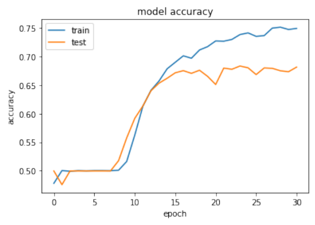


(i) Common Crawl: Confusion matrix

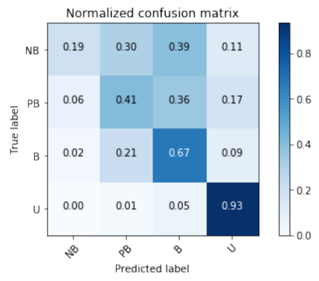
Figure C.1: Training loss, accuracy and confusion matrix for each vanilla LSTM with different embedding types



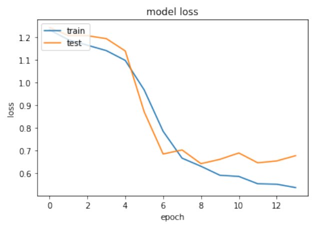
(a) Google News: Loss



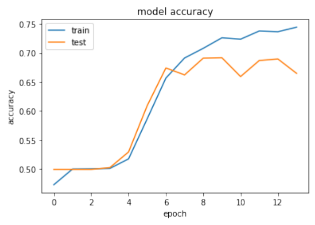
(b) Google News: Accuracy



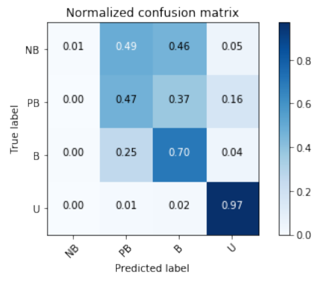
(c) Google News: Confusion matrix



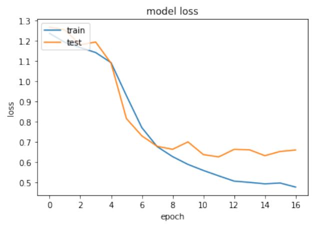
(d) GloVe Twitter: Loss



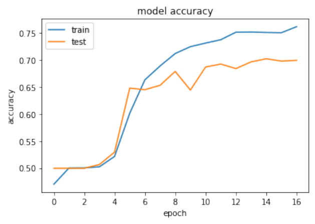
(e) GloVe Twitter: Accuracy



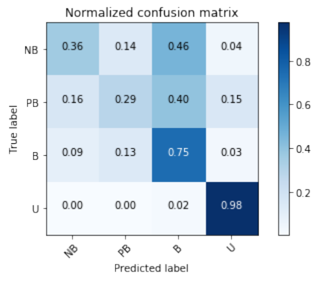
(f) GloVe Twitter: Confusion matrix



(g) Common Crawl: Loss

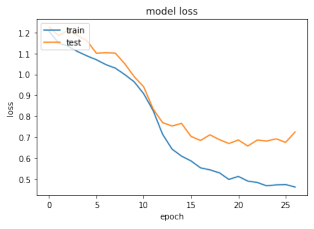


(h) Common Crawl: Accuracy

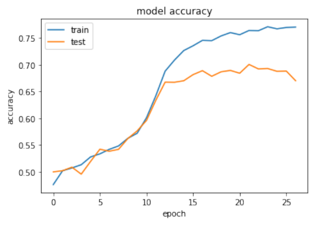


(i) Common Crawl: Confusion matrix

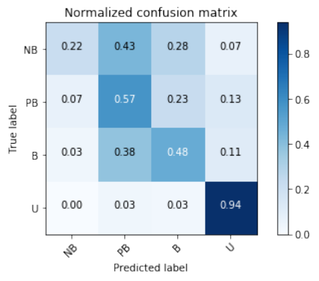
Figure C.2: Training loss, accuracy and confusion matrix for each stacked LSTM with different embedding types



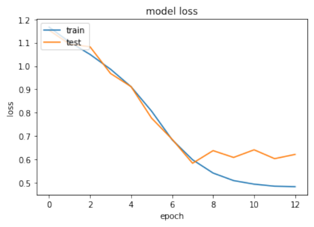
(a) Google News: Loss



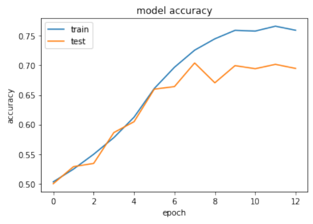
(b) Google News: Accuracy



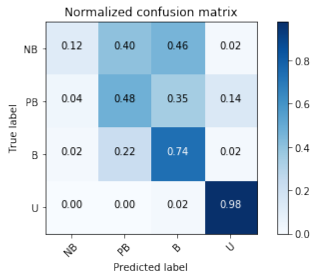
(c) Google News: Confusion matrix



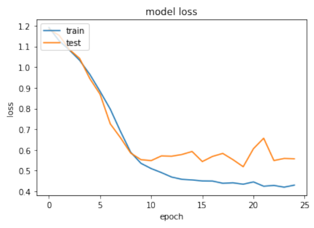
(d) GloVe Twitter: Loss



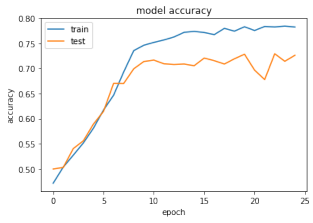
(e) GloVe Twitter: Accuracy



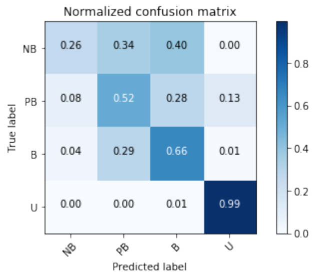
(f) GloVe Twitter: Confusion matrix



(g) Common Crawl: Loss



(h) Common Crawl: Accuracy



(i) Common Crawl: Confusion matrix

Figure C.3: Training loss, accuracy and confusion matrix for each bidirectional LSTM with different embedding types

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] R. E. Banchs and H. Li. Iris: a chat-oriented dialogue system based on the vector space model. In *Proceedings of the ACL 2012 System Demonstrations*, pages 37–42. Association for Computational Linguistics, 2012.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[4] J. P. Chiu and E. Nichols. Named entity recognition with bidirectional lstm-cnns. *arXiv preprint arXiv:1511.08308*, 2015.

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

[6] F. Chollet et al. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k*, 7(8), 2015.

[7] Y. Cui, Z. Chen, S. Wei, S. Wang, T. Liu, and G. Hu. Attention-over-attention neural networks for reading comprehension. *arXiv preprint arXiv:1607.04423*, 2016.

[8] P.-T. De Boer, D. P. Kroese, S. Mannor, and R. Y. Rubinstein. A tutorial on the cross-entropy method. *Annals of operations research*, 134(1):19–67, 2005.

[9] J. Donahue, L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[10] J. L. Fleiss. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378, 1971.

[11] A. I. for Artificial Intelligence. Deep contextualized word representations. URL: https://allennlp.org/elmo, last checked on 07-01-2019.

[12] F. A. Gers, J. Schmidhuber, and F. Cummins. Learning to forget: Continual prediction with lstm. 1999.

[13] P. Geurts, D. Ernst, and L. Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[14] A. Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013.

[15] A. Graves, N. Jaitly, and A.-r. Mohamed. Hybrid speech recognition with deep bidirectional lstm. In *Automatic Speech Recognition and Understanding (ASRU), 2013 IEEE Workshop on*, pages 273–278. IEEE, 2013.

[16] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5-6):602–610, 2005.

[17] H. P. Grice. Logic and conversation. *1975*, pages 41–58, 1975.

[18] R. Higashinaka, K. Funakoshi, M. Araki, H. Tsukahara, Y. Kobayashi, and M. Mizukami. Towards taxonomy of errors in chat-oriented dialogue systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 87–95, 2015.

[19] R. Higashinaka, K. Funakoshi, M. Inaba, Y. Arase, and Y. Tsunomori. The dialogue breakdown detection challenge 2. *Proceedings of SIG-SLUD*, 2016.

[20] R. Higashinaka, K. Funakoshi, M. Inaba, Y. Tsunomori, T. Takahashi, and N. Kaji. Overview of dialogue breakdown detection challenge 3. *Proceedings of Dialog System Technology Challenge*, 6, 2017.

[21] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[22] T. Horii and M. Araki. A breakdown detection method based on taxonomy of errors in chat-oriented dialogue. 2015. (in Japanese).

[23] Z. Huang, W. Xu, and K. Yu. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*, 2015.

[24] T. Iki and A. Saito. End-to-end character-level dialogue breakdown detection with external memory models.

[25] M. Inaba and K. Takahashi. Improving the performance of chat-oriented dialogue systems via dialogue breakdown detection.

[26] N. Japkowicz. The class imbalance problem: Significance and strategies. In *Proc. of the Intl Conf. on Artificial Intelligence*, 2000.

[27] B. Jason. *Long Short-Term Memory Networks With Python.* Del Rey (reprint), 2017. ISBN-13: 978-0-12-394424-5.

[28] D. Jurafsky and J. H. Martin. *Speech and language processing*, volume 3. Pearson London, 2014.

[29] S. Kato and T. Sakai. Rsl17bd at dbdc3: Computing utterance similarities based on term frequency and word embedding vectors. In *Proceedings of DSTC6. http://workshop. colips. org/dstc6/papers/track3_paper13_kato. pdf*, 2017.

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

[31] J. Lafferty, A. McCallum, and F. C. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. 2001.

[32] T. K. Landauer, P. W. Foltz, and D. Laham. An introduction to latent semantic analysis. *Discourse processes*, 25(2-3):259–284, 1998.

[33] J. Lopes. How generic can dialogue breakdown detection be? the kth entry to dbdc3.

[34] J. Lu, C. Xiong, D. Parikh, and R. Socher. Knowing when to look: Adaptive attention via a visual sentinel for image captioning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, volume 6, page 2, 2017.

[35] B. Martinovsky and D. Traum. The error is the clue: Breakdown in human-machine interaction. Technical report, Institute for Creative Technologies, University of Southern California, 2006.

[36] R. Meena, J. Lopes, G. Skantze, and J. Gustafson. Automatic detection of miscommunication in spoken dialogue systems. In *Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 354–363, 2015.

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

[38] Y. Nesterov et al. Gradient methods for minimizing composite objective function, 2007.

[39] K. Nigam, J. Lafferty, and A. McCallum. Using maximum entropy for text classification. In *IJCAI-99 workshop on machine learning for information filtering*, volume 1, pages 61–67, 1999.

[40] C. Park, K. Kim, and S. Kim. Attention-based dialog embedding for dialog breakdown detection. In *Proceedings of the Dialog System Technology Challenges Workshop (DSTC6)*, 2017.

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

[42] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[43] M. Riedmiller and H. Braun. A direct adaptive method for faster backpropagation learning: The rprop algorithm. In *Neural Networks, 1993., IEEE International Conference on*, pages 586–591. IEEE, 1993.

[44] C. R. Rogers. *Counselling and psychotherapy*, volume 298. Houghton Mifflin Boston, 1942.

[45] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[46] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533, 1986.

[47] H. Ryuichiro, K. Funakoshi, Y. Kobayashi, and I. Michimasa. The dialogue breakdown detection challenge: Task description, datasets, and evaluation metrics.

[48] I. V. Serban, A. Sordoni, R. Lowe, L. Charlin, J. Pineau, A. C. Courville, and Y. Bengio. A hierarchical latent variable encoder-decoder model for generating dialogues. In *AAAI*, pages 3295–3301, 2017.

[49] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[50] H. Sugiyama. Utterance selection based on sentence similarities and dialogue breakdown detection on ntcir-12 stc task. In *NTCIR*, 2016.

[51] I. Sutskever, J. Martens, and G. E. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[52] T. Tieleman and G. Hinton. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[53] C. J. Van Rijsbergen. *Information Retrieval.* Butterworth-Heinemann, 1979.

[54] J. Weizenbaum. Eliza – a computer program for the study of natural language communication between man and machine. *Communications of the ACM*, 9(1):36–45, 1966.

[55] Wikipedia. Dialogue system. URL: https://en.wikipedia.org/wiki/Dialogue_system, last checked on 07-01-2019.

[56] D. R. Wilson and T. R. Martinez. The general inefficiency of batch training for gradient descent learning. *Neural Networks*, 16(10):1429–1451, 2003.

[57] Y. Wu, M. Yuan, S. Dong, L. Lin, and Y. Liu. Remaining useful life estimation of engineered systems using vanilla lstm neural networks. *Neurocomputing*, 275:167–179, 2018.

[58] Z. Xie and G. Ling. Dialogue breakdown detection using hierarchical bi-directional lstms. In *Proceedings of the Dialog System Technology Challenges Workshop (DSTC6)*, 2017.

[59] Z. Yu, Z. Xu, A. W. Black, and A. Rudnicky. Strategy and policy learning for non-task-oriented conversational systems. In *Proceedings of the 17th Annual Meeting of the Special Interest Group on Discourse and Dialogue*, pages 404–412, 2016.

[60] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[61] Z. Zeng, C. Luo, L. Shang, H. Li, and T. Sakai. Test collections and measures for evaluating customer-helpdesk dialogues. *Proceedings of EVIA*, 2017.

[62] W. Zhu, C. Lan, J. Xing, W. Zeng, Y. Li, L. Shen, X. Xie, et al. Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks. In *AAAI*, volume 2, page 6, 2016.

Master thesis filing card

Student: Mariya Hendriksen

Title: Dialogue Breakdown Detection in Chatbots

UDC: 681.3*I20

Abstract:

One of the biggest problems of human-chatbot interaction is miscommunication. Occurring mainly on behalf of the chatbot, miscommunication can lead to dialogue breakdown, i.e., a point when the dialogue cannot be continued. Detecting such points can facilitate breakdown prevention or recovery after breakdown happened. The thesis represents a report for an internship undertaken at Language Intelligence & Information Retrieval Lab. The goal of the project is to build a multinomial sequence classifier to detect breakdowns in human-chatbot dialogues. The project includes an analysis of the existing dialogue breakdown detectors for defining a baseline. Besides, it entails the development of nine LSTM models with different architectures and embedding types. The experiments allowed to investigate the relationship between model architecture, embedding type and performance. Besides, two best performing models were selected and compared with the baseline. Overall, the created models demonstrated improvement over the baseline.

Thesis submitted for the degree of Master of Science in Artificial Intelligence, option Speech and Language Technology

Thesis supervisor: Prof. Dr. Marie-Francine Moens

Assessor: Prof. Dr. Hugo Van Hamme

Mentor: Artuur Leeuwenberg