

The University of Southern Denmark

The Maersk Mc-Kinney Moller Institute

Health and Welfare Technology

Master's thesis

Extraction of bleeding occurrences from
electronic health records for automatic
thromboprophylaxis decision support using
natural language processing

Authors:

Martin Sundahl

LAURSEN

Jannik Skyttegaard

PEDERSEN

Supervisors:

Thiusius Rajeeth

SAVARIMUTHU

Thomas

SCHMIDT

June 2, 2020



Title page

Master's thesis
Health and Welfare Technology
The Maersk Mc-Kinney Møller Institute
Faculty of Engineering
University of Southern Denmark

Title

(En.) Extraction of bleeding occurrences from electronic health records for automatic thromboprophylaxis decision support using natural language processing

(Da.) Udtræk af blødningshændelser fra elektroniske patientjournaler til automatisk beslutningsstøtte til administration af tromboseprofylakse ved brug af natural language processing

Number of signs incl. spaces: 217,846

Number of normal pages: 91

Appendices: A-R

We, Martin Sundahl Laursen and Jannik Skyttegaard Pedersen, declare that this thesis and the work presented in it is our own and has not been submitted for any other degree. The work has been equally distributed between us. All sources have been acknowledged.

Signature

Signature

Date

Date

Martin Sundahl Laursen
malau15@student.sdu.dk

Jannik Skyttegaard Pedersen
jannp15@student.sdu.dk

Section authors

1 Pedersen
2 - 4 Laursen
5 Pedersen and Laursen
6 - 6.2.1 Pedersen
6.2.2 - 6.2.3 Laursen
6.3 - 6.3.1 Pedersen
6.3.2 - 6.3.3.1 Laursen
6.3.3.2 - 6.3.4 Pedersen
6.4 Laursen and Pedersen
6.5 - 6.5.2.2 Laursen
6.5.2.3 Pedersen
6.6 - 6.6.1 Laursen
6.6.2 Pedersen
6.7 - 6.8 Laursen
7 - 8.4 Pedersen
8.5 - 8.7 Laursen
9 - 10 Pedersen
11 - 11.6 Laursen
11.7 - 11.8 Pedersen
11.8 - 13.2.2 Laursen
13.2.3 - 13.2.5 Pedersen
13.3 - 13.3.1 Laursen
13.3.2 - 13.3.4 Pedersen
13.4 - 14 Laursen
15 Pedersen

University of Southern Denmark

Abstract

The Faculty of Engineering
The Maersk Mc-Kinney Moller Institute

Extraction of bleeding occurrences from electronic health records for automatic thromboprophylaxis decision support using natural language processing

By Martin S. Laursen and Jannik S. Pedersen

Venous thromboembolism (VTE) is a life-threatening condition. The risk of developing VTE is up to 100 times higher in hospitalized patients than in the general population. Administering anticoagulant medicine as thromboprophylaxis decreases the risk of VTE, but at the same time it increases the risk of bleeding. Scoring systems can support the decision of which type, if any, thromboprophylaxis should be administered to patients. The process of utilizing these scoring systems is time consuming because the doctor needs to look through the full electronic health record (EHR) to look for e.g. previous bleeding occurrences.

This thesis investigates whether bleeding occurrences can be extracted from EHRs automatically. This has the potential to reduce the time spent by doctors on the time-consuming task and improve the administration of thromboprophylaxis. Using a convolutional neural network, an accuracy of 89.2 % was achieved on a balanced test set consisting of 1,178 sentences from EHRs collected from Odense University Hospital. The CNN was moreover tested on a note-level using an EHR of 220 notes which yielded a sensitivity of 0.976 and a specificity of 0.620.

Acknowledgements

We would like to thank Thiusius Rajeeth Savarimuthu and Pernille Just Vinholt for being the administrative driving forces behind this thesis and its objective, and for their support, continued guidance, and belief in us.

We would like to thank Thomas Schmidt for his help and guidance, and for sharing his knowledge of Clinical Decision Support Systems with us.

Without the help from our 12 annotators from Odense University Hospital, this thesis would not have been possible. Mentioned in alphabetical order they are: Anne Alnor, Anne-Sofie Faarvang Thorsen, Camilla Brødsgaard Nielsen, Cathrine Brødsgaard, Charlotte Gils, Eline Sandvig Andersen, Ina Mathilde Kjær, Kristian Voss Bjerre, Kristina Bjerg Appel, Lou-Ann Andersen, and Rasmus Søgaaard Hansen. Thank you for your effort!

We would also like to thank Christian Hardahl, Morten Krogh Danielsen, and Rasmus Davidsen from SAS for introducing us to their machine learning platform and for their guidance in that regard.

Lastly, we would like to thank Leon Derczynski from the IT University of Copenhagen for facilitating the use of their pre-trained GloVe word embeddings.

Table of contents

1	Introduction	1
2	Problem definition	2
2.1	Balancing the risk of venous thromboembolism and bleeding	2
2.2	Scoring systems for venous thromboembolism and bleeding	3
2.2.1	ImPACT-ILL	3
2.2.2	Khorana	4
2.2.3	IMPROVE	4
2.2.4	Tables for balancing VTE and bleeding risks	6
2.3	Treatment problems	7
2.4	Chapter Summary	9
3	Objective	11
3.1	Overall research objective	11
3.2	Thesis objective	11
4	Pathology of venous thromboembolism and bleeding	13
4.1	Venous thromboembolism	13
4.2	Bleeding	14
4.3	Chapter Summary	15
5	Related work	16
5.1	Search methodology and inclusion criteria	16
5.2	Presentation of related work	18
5.3	Key takeaways	25
5.4	Chapter summary	25
6	Natural language processing with deep learning	26
6.1	Introduction to text classification	26
6.2	Word embeddings	28
6.2.1	Word2Vec	28
6.2.2	Global Vectors	32
6.2.3	Differences between Word2Vec and Global Vectors	36
6.3	Neural networks for text classification	37

6.3.1	Convolutional neural networks	37
6.3.2	Simple recurrent neural networks	43
6.3.3	Extensions of the simple recurrent neural network.....	45
6.3.4	Modifications to recurrent architectures	50
6.4	Backpropagation	52
6.4.1	An example of backpropagation.....	52
6.4.2	Backpropagation for a simple RNN	54
6.4.3	Backpropagation for a simple CNN	60
6.5	Vanishing and exploding gradients	62
6.5.1	The problem of vanishing and exploding gradients	62
6.5.2	Avoiding vanishing and exploding gradients	63
6.6	Regularization techniques	68
6.6.1	Weight regularization.....	68
6.6.2	Dropout	69
6.7	Optimizers.....	70
6.7.1	Adam.....	70
6.7.2	SGD with momentum	72
6.8	Chapter summary	73
7	Data acquisition and annotation	75
7.1	Data acquisition.....	75
7.2	Data annotation.....	76
7.2.1	Extraction to Word.....	76
7.2.2	Pre-annotation cleaning	77
7.2.3	Annotation in Word.....	77
7.2.4	Inter annotator agreement	78
7.3	Chapter summary	79
8	Generation of datasets	80
8.1	Structure of EHR text	81
8.2	Division of text into samples	81
8.2.1	Paragraph-based samples.....	81
8.2.2	Sentence-based samples.....	82

8.2.3	Comparison of the sample methods	85
8.3	Extraction of samples	86
8.3.1	Extraction of annotated samples	86
8.3.2	Extraction of random samples	86
8.4	Data de-identification	87
8.4.1	Pre-de-identification cleaning.....	87
8.4.2	De-identification method.....	87
8.5	Sample cleaning	88
8.6	Distribution of datasets	89
8.6.1	Training objective	89
8.6.2	Distribution.....	90
8.7	Chapter summary	91
9	Test of word embeddings and datasets	92
9.1	Preprocessing	92
9.2	Test of word embeddings	92
9.3	Test of datasets.....	94
9.4	Chapter summary	96
10	Exploratory data analysis and preprocessing.....	97
10.1	Quantitative analysis of data	97
10.2	Qualitative analysis	99
10.3	Data preprocessing	101
10.4	Chapter summary.....	102
11	Training.....	103
11.1	Rule-based classification baseline	103
11.2	Training strategy for neural networks.....	104
11.3	Convolutional model.....	105
11.3.1	Architecture hyperparameters.....	107
11.3.2	Optimizer and learning rate.....	109
11.3.3	Setting hyperparameters	110
11.4	Recurrent model.....	112
11.5	Combination model	112

11.6	Selection of best model.....	114
11.7	Augmentation.....	114
11.7.1	Input dropout.....	115
11.7.2	Backtranslation.....	115
11.8	Final model ensemble.....	116
11.9	Chapter summary.....	117
12	Results.....	119
12.1	Test set performance.....	119
12.2	Real-life EHR performance.....	120
13	Discussion.....	122
13.1	Analysis of results.....	122
13.1.1	Analysis of test set performance.....	122
13.1.2	Analysis of EHR test performance.....	123
13.2	Analysis of model.....	126
13.2.1	Convolutional vs. recurrent model.....	126
13.2.2	Filter sizes and stacked layers.....	127
13.2.3	Inner workings of the convolutional neural network.....	128
13.2.4	Effect of spelling mistakes.....	134
13.2.5	Gender and age bias of the model.....	134
13.3	Improvement of performance.....	135
13.3.1	Improvement of data quality.....	136
13.3.2	Increase amount of data.....	137
13.3.3	Other models.....	138
13.3.4	Word embeddings.....	138
13.4	Implementation of a Clinical Decision Support System.....	139
13.4.1	Clinical Decision Support System for relevant bleeding.....	139
13.4.2	Communication mechanism.....	140
13.4.3	Keeping the model up to date.....	141
13.5	Chapter summary.....	141
14	Future work.....	143
14.1	Extraction of bleeding occurrences.....	143

14.2	Supporting decisions on thromboprophylaxis.....	144
15	Conclusion	145
	Appendix A: Explanation of the derivation of GloVe.....	146
	Appendix B: Differentiation of cross entropy loss with softmax.....	149
	Appendix C: Transposing ht	152
	Appendix D: Math of upper bounded terms of RNN.....	155
	Appendix E: ICD-10 codes of patient groups.....	158
	Appendix F: Annotation guide.....	162
	Appendix G: Code for inserting HTML tags in annotated EHRs.....	163
	Appendix H: Test of Stanza and NLTK tokenizer.....	169
	Appendix I: Code for extracting sentence samples	178
	Extract and de-identify annotated sentences	178
	Extract and de-identify random sentences	180
	Appendix J: Code for extracting paragraph samples	183
	Extract annotated paragraphs.....	183
	Extract random paragraphs.....	186
	De-identify extracted paragraphs	186
	Appendix K: List of stop-words	188
	Appendix L: Training of recurrent neural network.....	189
	Appendix M: Code for rule-based classifier.....	197
	Appendix N: Code for training CNN and RNN	201
	Appendix O: Code for augmentation	211
	Input dropout method	211
	Backtranslation method.....	212
	Appendix P: Code for Grad-CAM	213
	Appendix Q: Time schedule	216
	Appendix R: Attached files	219
	References.....	220

List of Figures

Figure 4.1 Illustration of DVT and PE.....	13
Figure 5.1: Flowchart of identified articles for related work.....	18
Figure 6.1: Neural text classification process.....	26
Figure 6.2: Illustration of the principle behind the CBOW model.	29
Figure 6.3: Illustration of the CBOW model	29
Figure 6.4: Visualization of vector differences representing underlying concepts of words.....	34
Figure 6.5: Weighting function for GloVe loss function.....	35
Figure 6.6: A 1x3 filter slides over a sentence.....	38
Figure 6.7: Illustration of a filter sliding across a sentence.....	38
Figure 6.8: Illustration of a conv-layer with three filters	39
Figure 6.9: Illustration of three different pooling operations	41
Figure 6.10: Illustration of the Grad-CAM method.....	42
Figure 6.11: Different types of RNNs.....	43
Figure 6.12: Simple RNN cell.....	43
Figure 6.13: Unrolled simple RNN architecture.....	45
Figure 6.14: The Gated Recurrent Unit.....	46
Figure 6.15: The LSTM cell	48
Figure 6.16: Illustration of a bilateral LSTM.....	51
Figure 6.17: Illustration of a 3-layer stacked RNN	52
Figure 6.18: Simple example of backpropagation.....	53
Figure 6.19: Gradient flow for a simple RNN with three inputs.....	59
Figure 6.20: The ReLU, sigmoid, and tanh activation functions	64
Figure 6.21: The ReLU, sigmoid, and tanh activation function derivatives	64
Figure 6.22: Left: Fully connected layers without dropout.....	69
Figure 6.23: Illustration of the SGD with momentum update.....	72
Figure 7.1: Flowchart showing the pipeline of data aquisition and annotation.....	75
Figure 7.2: Example of the extracted unstructured text of an EHR.....	76
Figure 7.3: Example of an unstructured text cleaned for HTML tags	77
Figure 7.4: Example of an unstructured text with annotations tags.....	78

Figure 8.1: Pipeline of the generation of sentence and paragraph dataset	80
Figure 8.2: Example of the structure of an EHR	81
Figure 8.3: An example of an unstructured text cleaned pre-tokenization for extra dots	83
Figure 10.1: Sentence length of the dataset	97
Figure 11.1: General structure of a CNN and the hyperparameters associated	105
Figure 11.2: Test of number of dense layers for CNN model	108
Figure 11.3: Test of filter sizes for CNN model	108
Figure 11.4: Test of convolutional layers and filters for CNN model	109
Figure 11.5: Test of optimizer and learning rate for CNN model	110
Figure 11.6: Visualization of the training process for the best CNN model	112
Figure 11.7: Architecture of the combination model	113
Figure 11.8: An example of backtranslation	116
Figure 11.9: Visualization of the performance of different ensemble sizes	117
Figure 13.1: A correctly classified EHR note	124
Figure 13.2: Correctly classified EHR note with some sentence misclassifications	125
Figure 13.3: Grad-CAM visualization of each conv-layer	128
Figure 13.4: Grad-CAM visualization of words leading towards a negative prediction for a false positive	132
Figure 13.5: Graph of the loss and accuracy for different sizes of the training set	137
Figure A.1: Weighting function for GloVe loss function	148
Figure L.1: General structure of a RNN and the hyperparameters associated	189
Figure L.2: Test of recurrent layer type	191
Figure L.3: Test of RNN dense layers	192
Figure L.4: Test of recurrent layers	193
Figure L.5: Test of recurrent layer size	193
Figure L.6: Test of optimizer and learning rate	194
Figure L.7: Rate of learning for Adam and SGD	195

List of Tables

Table 2.1: ImPACT-ILL risk factors and scores for VTE.....	3
Table 2.2: Khorana risk factors and scores for chemotherapy-associated VTE	4
Table 2.3: IMPROVE risk factors and scores for bleeding	5
Table 2.4: RADS recommendations for thromboprophylaxis for medical patients	6
Table 2.5: RADS recommendations for thromboprophylaxis for cancer patients in chemotherapy	6
Table 5.1: Related work summerized	24
Table 6.1: Example of co-occurrence matrix.	33
Table 6.2: Probabilities and probability ratios describing the target words	34
Table 6.3: Summary of the Word2vec and GloVe methods and acquired pre-trained embeddings.....	37
Table 7.1: Distribution of extracted EHRs	75
Table 8.1: Sentence tokenization test between Stanza and NLTK.....	84
Table 8.2: Example of text before and after deidentification.....	88
Table 8.3: Number of samples for the paragraph dataset	89
Table 8.4: Number of samples for the sentence dataset.....	89
Table 8.5: Train, validation, and test distribution of paragraph dataset .	90
Table 8.6: Train, validation, and test distribution of sentence dataset	91
Table 9.1: Result of embedding test.....	93
Table 9.2: Comparision of fastText and GloVe word embeddings.....	93
Table 9.3: Token count statistics of sentence and paragraph datasets.....	95
Table 9.4: Result of dataset test	95
Table 10.1: Token count statistics for sentence dataset	98
Table 10.2: Frequent words in sentence training dataset	99
Table 10.3: Examples of sentence structures in the sentence dataset.....	100
Table 10.4: Examples of bleeding indicating words	101
Table 11.1: Final CNN structure	111
Table 11.2: Overview of the results of the recurrent, convolutional, and combination model.....	114

Table 11.3: Results of increasing the dataset with input dropout augmentation	115
Table 11.4: Results of increasing the dataset with backtranslation augmentation	116
Table 12.1: Validation and test set results	119
Table 12.2: Confusion matrix on the test set.....	120
Table 12.3: Measures on the test set.....	120
Table 12.4: Confusion matrix on the real-life EHR test	121
Table 12.5: Measures on the real-life EHR test	121
Table 13.1: Grad-CAM visualization of a true positive	129
Table 13.2: Grad-CAM visualization of a true negative	130
Table 13.3: Grad-CAM visualization of a false negative.....	131
Table 13.4: Grad-CAM visualization of words leading towards negative prediction of a true positive.....	131
Table 13.5: Grad-CAM visualization of a false positive.....	132
Table 13.6: Grad-CAM visualization of a false positive.....	133
Table 13.7: Test of the model's reaction to spelling mistakes.....	134
Table 13.8: Softmax scores of sentences with gender and age weighted tokens.....	135
Table 13.9: Selected words and their 3 most similar words.....	139
Table E.1: ICD-10 codes of patient groups	161
Table H.1: Sentence tokenization test between Stanza and NLTK	177
Table L.1: Final RNN model architecture and hyperparameters.....	196

List of abbreviations

Bi-RNN	Bidirectional Recurrent Neural Network
CBOW	Continuous Bag of Words
CNN	Convolutional Neural Network
Conv-layer	Convolutional layer
DG	Downstream Gradient
DVT	Deep Vein Thrombosis
EHR	Electronic Health Record
GloVe	Global Vectors
Grad-CAM	Gradient-weighted Class Activation Mapping
GRU	Gated Recurrent Unit
ICU	Intensive Care Unit
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
OOV	Out of Vocabulary
PE	Pulmonary Embolism
PPV	Positive Predictive Value
RADS	The Danish Council for the Use of Expensive Hospital Medicines
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
SVM	Support Vector Machine
VTE	Venous Thromboembolism

Abbreviations will be presented the first time they are used in every chapter and again in the chapter summaries.

Chapter 1

1 Introduction

Venous thromboembolism (VTE) is a life-threatening condition. The risk of developing VTE is up to 100 times higher in hospitalized patients than in the general population. Administering anticoagulant medicine as thromboprophylaxis has shown to decrease the risk of VTE, but at the same time it increases the risk of bleeding. To balance the risk of VTE and bleeding, scoring systems can support the decision of which, if any, thromboprophylaxis should be administered. These scoring systems rely on information from electronic health records (EHR) such as previous bleeding occurrences. This information is not easily available, e.g. because the doctor must look through the full EHR for previous bleeding occurrences. This is very time consuming, and therefore this thesis investigates whether bleeding occurrences can automatically be extracted from EHRs. It has the potential to reduce the time spent by doctors on a time-consuming task and improve the administration of thromboprophylaxis.

The problem will first be described and analyzed after which the objective is stated. After presenting the objective, the pathology of VTE and bleeding occurrences are described. The thesis then presents the relevant related work.

The theory of extracting text from EHRs using deep learning is then described, including sections covering word embeddings, neural networks, and backpropagation.

The thesis then describes how data was acquired, annotated, and split into two different datasets. A test to decide the best word embedding and dataset is then presented. It is then described how the best performing dataset was preprocessed before being input to a convolutional neural network, a recurrent neural network, and a combination of the two. The best performing neural network is then presented and the results are discussed, including how the results can be improved and eventually used in a Clinical Decision Support System.

Chapter 2

2 Problem definition

This chapter introduces the challenges of balancing the risk of venous thromboembolism (VTE) and bleeding in hospitalized patients. It then presents the scoring systems to balance the risks when deciding on thromboprophylaxis treatment, i.e. preventive medicine. Afterwards, the problems regarding the treatment are presented.

2.1 Balancing the risk of venous thromboembolism and bleeding

Hospitalized patients have an increased risk of developing VTE which is the occurrence of either a deep vein thrombosis (DVT) or a pulmonary embolism (PE) [1]. The risk of VTE in hospitalized patients is up to 100 times higher than that of the general population of same sex and age, and one third of the occurrences of VTE happen in the months after hospitalization [2], [3]. Studies [4], [5] show that VTE occurs in up to 15 % of hospitalized patients who are not given anticoagulant treatment. Another study [6] shows that the 30-day mortality risk for VTE patients is 3.0 % for DVT and 31 % for PE versus 0.4 % in the comparison cohort.

The risk of developing VTE can be decreased by administering anticoagulant treatment as thromboprophylaxis. A study by Samama et al. [5] showed that administering thromboprophylaxis decreased the incidence of VTE from 14.9 % in the placebo group to 5.5 % in the treatment group. A study by Leizorovicz et al. [4] showed that administering thromboprophylaxis reduced the incidence from 4.96 % in the placebo group to 2.77 % in the treatment group.

While anticoagulant treatment decreases the risk of VTE, it increases the risk of bleeding [7]. Bleeding occurs in 3.2 % of medical patients and approximately one third are considered major bleedings which in rare cases can be fatal [8]. Therefore, when thromboprophylaxis is administered, it must be balanced against the risk of bleeding. For this purpose, The Danish Council for the Use of Expensive Hospital Medicines (RADS) propose scoring systems

that assesses the combined risk of VTE and bleeding to predict which patients will benefit from thromboprophylaxis. The RADS scoring systems are based on the ImPACT-ILL, Khorana, and IMPROVE scores.

2.2 Scoring systems for venous thromboembolism and bleeding

The ImPACT-ILL scoring system is used for VTE in the general population, the Khorana scoring system is used for VTE in cancer patients, and the IMPROVE scoring system is used for bleeding [9]. The scoring systems identify risk factors of VTE or bleeding. They were developed using multiple regression analysis and assign points to the risk factors by interpreting the regression coefficients [8], [10], [11].

The risk factors and the assigned points are shown for each scoring system below.

2.2.1 ImPACT-ILL

The ImPACT-ILL scoring system uses seven different risk factors which each has a score ranging from 1 to 3, where 3 indicates a high risk. Previous VTE is identified as the largest risk factor for VTE, being assigned 3 points. Thrombophilia, lower-limb paralysis, and current cancer each count 2 points. Immobilization for 7 days or more, stay at the Intensive or Cardiac Care Unit (ICU/CCU), and an age over 60 each count 1 point. See Table 2.1.

Risk factor	Risk score
Previous VTE	3
Known thrombophilia	2
Current lower-limb paralysis	2
Current cancer	2
Immobilized prior to or during admission ≥ 7 days	1
ICU/CCU stay	1
Age > 60 years	1

Table 2.1: ImPACT-ILL risk factors and scores for VTE [10].

When summing the points, a score of 0-1 is considered a low risk of VTE, 2-3 is considered moderate, and 4 and above is considered high [10].

2.2.2 Khorana

The risk of VTE is elevated in cancer patients. For this reason, the Khorana score is specifically used to predict chemotherapy-associated VTE. A cancer in a very high risk site like stomach or pancreas counts 2 points. Cancer in high risk regions like lungs or bladder counts 1 point. As do different biomarkers as e.g. a BMI of $35 \frac{\text{kg}}{\text{m}^2}$ or more. See Table 2.2.

Risk factors	Risk score
Very high risk site of cancer (stomach, pancreas)	2
High risk site of cancer (lung, lymphoma, gynecologic, bladder, testicular)	1
Pre-chemotherapy platelet count $350 \times 10^9/\text{L}$ or more	1
Hemoglobin level less than $100\text{g}/\text{L}$ or use of red cell growth factors	1
Pre-chemotherapy leukocyte count more than $11 \times 10^9/\text{L}$	1
BMI $35 \text{ kg}/\text{m}^2$ or more	1

Table 2.2: Khorana risk factors and scores for chemotherapy-associated VTE [11].

When summing the scores, a score of 0 is considered low risk, a score of 1-2 is considered intermediate risk, and a score of 3 or above is considered high risk [11].

2.2.3 IMPROVE

The IMPROVE score identifies several risk factors for bleeding in hospitalized medical patients, the three most important being an active gastroduodenal ulcer, a platelet count below $50 \cdot 10^9$ cells/L, and a previous bleeding occurring within 3 months before the admission. The study defines bleeding as either a major bleeding or a non-major but clinically relevant bleeding occurring in a medical patient. This means that bleedings attributed to surgery are not considered.

Major bleedings are defined as:

- Bleeding event contributing to death
- Bleeding within a critical organ (e.g. including intracranial, adrenal gland, or spinal bleeding)

- Clinically overt¹ bleeding associated with a fall in hemoglobin level of 2 g/dL or leading to transfusion of at least 2 units of packed red blood cells

Non-major but clinically relevant bleedings are defined as:

- Overt gastrointestinal bleeding
- Visible blood in the urine lasting more than 1 day
- Substantial nosebleed that required intervention and was recurrent and/or lasted at least 5 minutes
- Extensive hematoma² or bruising
- Intraarticular bleeding
- Heavy menstrual bleeding or intermenstrual bleeding
- Bleeding important enough to be recorded on the hospital chart

All risk factors and their assigned points can be seen Table 2.3.

Bleeding risk factors	Risk score
Moderate renal failure (GFR 30-59 mL/min/m ²)	1
Male	1
Age 40-84 years	1.5
Current cancer	2
Rheumatic disease	2
Central venous catheter	2
ICU/CCU stay	2.5
Severe renal failure (GFR <30 mL/min/m ²)	2.5
Hepatic failure (INR<1.5)	2.5
Age≥85 years	3.5
Platelet count <50x10 ⁹ cells/L	4
Bleeding within 3 months before admission	4
Active gastroduodenal ulcer	4.5

Table 2.3: IMPROVE risk factors and scores for bleeding [8].

¹ A bleeding that is visible to the patient or clinician. May manifest as vomiting of blood or blood in the feces [97]

² Large leakage often from large blood vessels, causing blood to pool [98]

When summing the points, a patient with a score below 7 is considered to not have a significantly elevated bleeding risk from anticoagulant thromboprophylaxis. Patients with a score of 7 or above should be given anticoagulant treatment with caution [8].

2.2.4 Tables for balancing VTE and bleeding risks

RADS propose the following tables for deciding thromboprophylaxis, balancing the risk of VTE and bleeding. Table 2.4 is for the general population [9]:

	Risk of VTE (ImPACT-ILL score)		
Risk of bleeding (IMPROVE score)	Low risk of VTE (0-1)	Moderate risk of VTE (2-3)	High risk of VTE (≥ 4)
Low bleeding risk (<7)	None	None	Low-molecular-weight heparin
High bleeding risk (≥ 7)	None	None	Mechanical prophylaxis

Table 2.4: RADS recommendations for thromboprophylaxis for medical patients

Table 2.5 is for cancer patients in chemotherapy [9]:

	Risk of VTE (Khorana score)		
Risk of bleeding (IMPROVE score)	Low risk of VTE (0)	Moderate risk of VTE (1-2)	High risk of VTE (≥ 3)
Low bleeding risk (<7)	None	None	Low-molecular-weight heparin
High bleeding risk (≥ 7)	None	None	Mechanical prophylaxis

Table 2.5: RADS recommendations for thromboprophylaxis for cancer patients in chemotherapy

Table 2.4 and Table 2.5 show that a high risk of VTE combined with a low risk of bleeding is treated with low-molecular-weight heparin which is a class of anticoagulant medication. A high risk of VTE combined with a high risk of bleeding is treated with mechanical prophylaxis, e.g. compression stockings.

All other combinations of VTE and bleeding risks should not be treated with any thromboprophylaxis.

2.3 Treatment problems

This section presents the problems associated with using the scoring systems to decide if and which thromboprophylaxis should be administered.

Insufficient or incorrect use of guidelines

Several studies show that a large part of hospitalized patients do not get or get the wrong prophylactic treatment against guideline indications. Amin et al. [12] found that 66.1 % of discharged patients from US medical centers did not receive appropriate thromboprophylaxis. 38.4 % received no prophylaxis despite indication, 4.7 % only mechanical prophylaxis when other treatment was indicated, 6.3 % an inappropriate dosage, and 16.7 % an inappropriate duration according to guidelines.

Rwabihama et al. [13] found that in a geriatric setting, 18.9 % received inadequate thromboprophylaxis - 11.1 % were undermedicated and 7.9 % were overmedicated.

Another study by Amin et al. [14] found that many cancer patients do not get the appropriate thromboprophylaxis. 73.0 % of discharges did not receive the appropriate prophylaxis - among medical discharges 46.0 % did not receive treatment despite indication, 11.8 % received only mechanical prophylaxis when pharmacologic prophylaxis was indicated, 11.5 % an inappropriate dose, and 3.5 % an inappropriate duration.

The results of the studies suggest that the guidelines, e.g. the scoring systems mentioned in section 2.2, are not always used in practice to decide on a treatment with prophylaxis. Often prophylaxis is underutilized which might be because of the fear of bleeding [15].

Manual calculation of score

One limiting factor of the usage of the scoring systems might be that today, the scores are calculated manually. The manual calculation is time consuming and it poses a further problem if the state of a patient changes during admission. A patient might e.g. be transferred to the ICU or the platelet count

could drop below the threshold of the IMPROVE scoring system after the initial calculation of the score. This might demand a change in the prophylaxis treatment that would need to be calculated manually again.

No easy access to data

Some of the risk factors used in the three different scoring systems are readily available in the EHR: Age, if the patient is staying at the ICU, immobilization period, cancer site, platelet count, BMI, e.g. Other risk factors, like previous VTE or a relevant medical bleeding occurring 3 months or less before the admission are more difficult to extract information about. In these cases, the medical staff have to look through the whole medical history of a patient to check for previous VTE or through the last 3 months of medical history to look for bleeding. This takes up a large amount of time as EHRs can be very long and the text is unstructured and includes lots of irrelevant information for a given problem. Moreover, the EHR system of Odense University Hospital, called COSMIC, only allows to open 40 notes at a time.

Diagnosis codes (ICD-10) could be a shortcut to the information as they are used to code the different diagnoses of an admission but they usually only code the main diagnosis. VTE is usually considered a primary problem of an admission but if e.g. a bleeding occurs as a secondary problem, it will usually not be coded. Additionally, the diagnosis codes are first created when the patient is discharged so they would only be of use for a subsequent admission. It is not possible, though, to search for diagnosis codes, nor free text, via COSMIC which limits the usability of diagnosis codes for use during admissions. Another limiting factor is the fact that Valkhoff et al. [16] found that a sample from the Aarhus University Hospital Database of 158 EHRs with ICD-10 codes related to upper gastrointestinal bleeding only had a positive predictive value of 77 %. This means that 23 % of the EHRs that had been coded as upper gastrointestinal bleeding were false positives. This indicates that the diagnosis codes are not always accurate.

The above points suggest that currently, the only way of extracting data for the thromboprophylaxis decision support table of RADS is to manually look through the unstructured text of the medical record.

Differentiating relevant and irrelevant bleedings

Apart from the difficulty in extracting the necessary data from the EHR, there is an added difficulty in assessing the bleeding for the IMPROVE scoring system. One important factor of the score is the occurrence of a major or non-major but clinically relevant medical bleeding within 3 months prior to admission. It can be hard for doctors to differentiate a medical bleeding as defined in section 2.2.3 from a surgical or traumatic bleeding which would not count in the score. It could e.g. be debated if blood in a catheter is caused by the surgery where it was inserted or from a simultaneous medical condition.

2.4 Chapter Summary

This chapter introduced the challenges of balancing the risk of venous thromboembolism (VTE) and bleeding when administering thromboprophylaxis to hospitalized patients.

Hospitalized patients have up to 100 times larger risk of developing VTE than the general population of same sex and age, and it occurs in up to 15 % who are not given anticoagulant treatment as prophylaxis.

Anticoagulant treatment decreases the risk of VTE but increases the risk of bleeding which occurs in 3.2 % of medical patients. Therefore, thromboprophylaxis must be balanced against the risk of bleeding. For this purpose, RADS proposes the use of scoring systems to predict which patients will benefit from thromboprophylaxis. The prediction is calculated based on the patient's risk of VTE and bleeding. Individual scoring systems for VTE and bleeding assign points to several risk factors so that a patient with e.g. a prior relevant bleeding up to 3 months before admission has a higher predicted risk of bleeding.

A high VTE risk combined with a low bleeding risk is treated with anticoagulant medication, and a high VTE risk combined with a high bleeding risk is treated with mechanical prophylaxis, e.g. compression stockings.

There are problems associated with thromboprophylaxis administration. Several studies show that a large part of hospitalized patients do not get or get the wrong prophylactic treatment against guideline indications. Often prophylaxis is underutilized which might be because of the fear of bleeding.

Another problem is that some of the risk factors used for the VTE and bleeding risk scores are not easily accessible in the EHR. To search for prior VTE or bleeding occurrences, medical staff must look through the whole medical history of a patient. This takes up a large amount of time.

Chapter 3

3 Objective

This chapter introduces the overall objective of the research, of which a part is undertaken in this thesis. It then presents the objective of the thesis.

3.1 Overall research objective

The overall end goal of the research, of which a part is undertaken in this thesis, is a solution for supporting the decision of thromboprophylaxis treatment by automatically detecting the risk of venous thromboembolism (VTE) and bleeding for hospitalized patients. The risk assessments of VTE and bleeding are made with the ImPACT-ILL, Khorana and IMPROVE scoring systems. The scoring systems are given information extracted from the electronic health record (EHR) such as prior VTE or bleeding, patient age, sex, blood test results, and other biomarkers to calculate the scores. Table 2.4 and Table 2.5 proposed by RADS is used to support the decision of which, if any, thromboprophylaxis should be given, balancing the risk of bleeding and VTE.

As described in section 2.3, information for the scoring system is not readily available. Automating the extraction of this data has the potential to decrease time spent by doctors and increase the use and accuracy of the decision support for administration of thromboprophylaxis.

The solution can be used to automatically calculate the risk scores and the indication of thromboprophylaxis at admission, and for surveilling changes in the state of the patient during admission that would demand a reassessment of the indication.

3.2 Thesis objective

The objective of this thesis is to use natural language processing, specifically deep learning techniques, to extract information from the unstructured text of the EHR that indicates if a patient has had a bleeding occurrence.

The task of extracting the simpler biomarkers will not be the focus of this thesis. The time perspective of the bleeding occurring 3 months or less before the hospitalization will not be considered. Neither will the extraction of occurrences of VTE.

The objective is moreover limited to extraction of any bleeding occurrence from the EHR, i.e. there is no differentiation between any bleeding and a major or non-major but clinically relevant medical bleeding as required by the IMPROVE score. The annotated EHR data provided for the thesis does not support the differentiation.

Chapter 4

4 Pathology of venous thromboembolism and bleeding

This chapter describes the pathology of venous thromboembolism (VTE) and bleeding.

4.1 Venous thromboembolism

Hospitalized patients have an increased risk of developing VTE. VTE is divided into two types based on where in the body it occurs. A deep vein thrombosis (DVT) is a blood clot that develops in the deep veins, usually the legs, and reduces the flow of blood. A pulmonary embolism (PE) is more life threatening than DVT. It occurs when a piece of a blood clot, possibly a DVT, breaks loose, travels to the lungs, and obstructs the blood flow [17], [18]. DVT and PE are visualized on Figure 4.1.

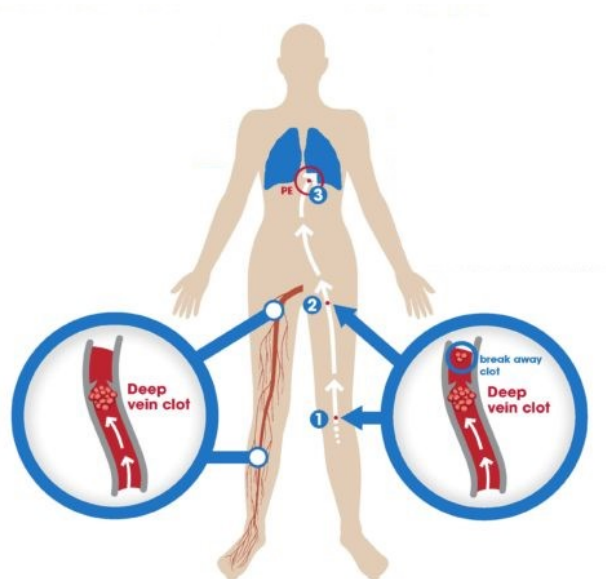


Figure 4.1 Illustration of DVT and PE [19]

The symptoms of DVT are swelling, tenseness, redness, and warmth of the affected body part. DVT can also occur without any noticeable symptoms [20]. PE on the other hand, gives symptoms such as unexplainable shortness of breath, chest pain that worsens when taking deep breaths or coughing, and high pulse [21].

In general, VTEs can happen if the blood has an increased tendency to coagulate, if there is damage on the inside of the vein, or if the blood flow slows down [22]. People in increased risk of developing a VTE are those with certain hereditary blood diseases, pregnant women, users of contraceptive pills, and people who are overweight or have had a prior VTE. Also cancer patients have an increased risk of VTE because the disease damages tissue in the body which triggers clotting [23].

Directly associated to the hospital setting, immobility for longer periods because of e.g. being bedridden after a surgery or a broken leg also increases the risk of VTE [20], [23]. Inactivity can cause the blood to pool in the lower parts of the body because muscle movement is needed to activate the venous pump that pushes blood back to the heart. The pooling of blood increases the risk of a clot. Inactivity can be caused by surgery, but also surgery itself is a risk factor because foreign matter like tissue debris, collagen and fat is at risk of being released to the bloodstream during the surgery. The blood responds by thickening which can cause a clot. Additionally, surgical tissue removal will make the blood respond by clotting [17], [24].

4.2 Bleeding

A bleeding can either be internal or external. Internal blood loss occurs when blood leaks from a damaged blood vessel or organ. An external bleeding is when the blood exits the body through a break in the skin. Blood from damaged tissue can also exit through natural body openings like the mouth or nose.

Bleedings can be caused by either trauma (including surgery) or a medical condition. When the bleeding is traumatic, the cause is usually apparent, like e.g. wounds or surgical incisions, and blood loss is expected. In medical patients, bleedings are caused by an underlying medical condition, and it is not always obvious what causes it. Conditions like cancer, bleeding disorders like

Von Willebrand disease³ or hemophilia⁴, or a low blood platelet count can cause medical bleedings [25].

4.3 Chapter Summary

This chapter described the pathology of venous thromboembolism (VTE) and bleeding.

VTE is divided into two types based on where in the body it occurs. A deep vein thrombosis is a blood clot that develops in the deep veins and reduces the flow of blood. A pulmonary embolism occurs when a piece of a blood clot breaks loose and travels to the lungs and obstructs the blood flow.

One of the reasons for VTE occurring in a hospital setting is the immobility of bedridden patients which can cause the blood to pool in the lower parts of the body, thus increasing the risk of a clot.

A bleeding can either be internal, i.e. a leak from a blood vessel or organ, or external, i.e. the blood exits through a break in the skin. They can be caused by either a trauma (including surgery) or a medical condition like a bleeding disorder.

³ A disease caused by a deficiency of Von Willebrand factor. This is a type of protein that helps the blood to clot [99].

⁴ An inherited bleeding disorder in which a person lacks or has low levels of clotting factors. This reduces the blood's ability to clot properly [100].

Chapter 5

5 Related work

This chapter provides an overview of relevant studies that have used natural language processing (NLP) algorithms to extract health related information from the unstructured text of electronic health records (EHR).

First, the search methodology and inclusion criteria for the articles are presented. Afterwards, the articles are described and summarized.

5.1 Search methodology and inclusion criteria

The electronic databases IEEE and PubMed were used to find relevant articles. For IEEE, the search was executed on October 1st 2019 and PubMed was searched on November 8th 2019. The two databases were searched with slightly different purposes, and therefore their search string and inclusion criteria differ. IEEE was chosen to include papers that provide state-of-the-art research in relation to information extraction, whereas PubMed was searched to find papers that have extracted information related to bleeding and venous thromboembolism (VTE) in EHRs. Even though the scope of this thesis only is extraction of bleeding occurrences, VTE was included to encompass the full domain of not readily available information extraction for thromboprophylaxis indication.

For IEEE, articles were considered eligible if the authors used deep learning techniques for extracting information from the unstructured text of EHRs. Moreover, the article should describe a concrete problem and it should describe the algorithms used in detail. Based on this, the following search string was created:

((((medical OR patient OR health OR hospital) ONEAR/1 (record OR journal*)) OR EMR OR EMJ OR EPR OR EPJ OR EHR OR EHJ) AND ("Natural language processing" OR NLP OR LSTM OR "Recurrent neural network*" OR RNN OR transformer))*

For PubMed, the health-related eligibility criteria were stricter, as it was chosen only to include studies related to either bleeding or VTE. In contrast, criteria of the technical aspects were relaxed to include all kinds of technical systems that extract knowledge from EHRs. Based on these criteria, the following search string was created:

((((("embolism and thrombosis"*[MeSH Terms])) OR *"hemorrhage"*[MeSH Terms])) AND *"medical records"*[MeSH Terms]) AND *"computing methodologies"*[MeSH Terms]*

PubMed was searched using MeSH terms meaning that e.g. “hemorrhage” will include words such as bleeding, hemothorax, hyphema etc., which are all related to bleedings.

In total, 460 articles were found using the above search strings. The 460 articles were first screened for eligibility using their title which excluded 327 articles. It was possible to exclude this high number of articles because the title often revealed that the article concerned structured or numerical information from EHRs. Afterwards, the remaining 133 articles were screened using their abstract which excluded 109 additional articles. This resulted in 24 articles that were assessed using their full text. 11 out of these 24 articles were included and are described in this section. Figure 5.1 shows a flowchart of the process.

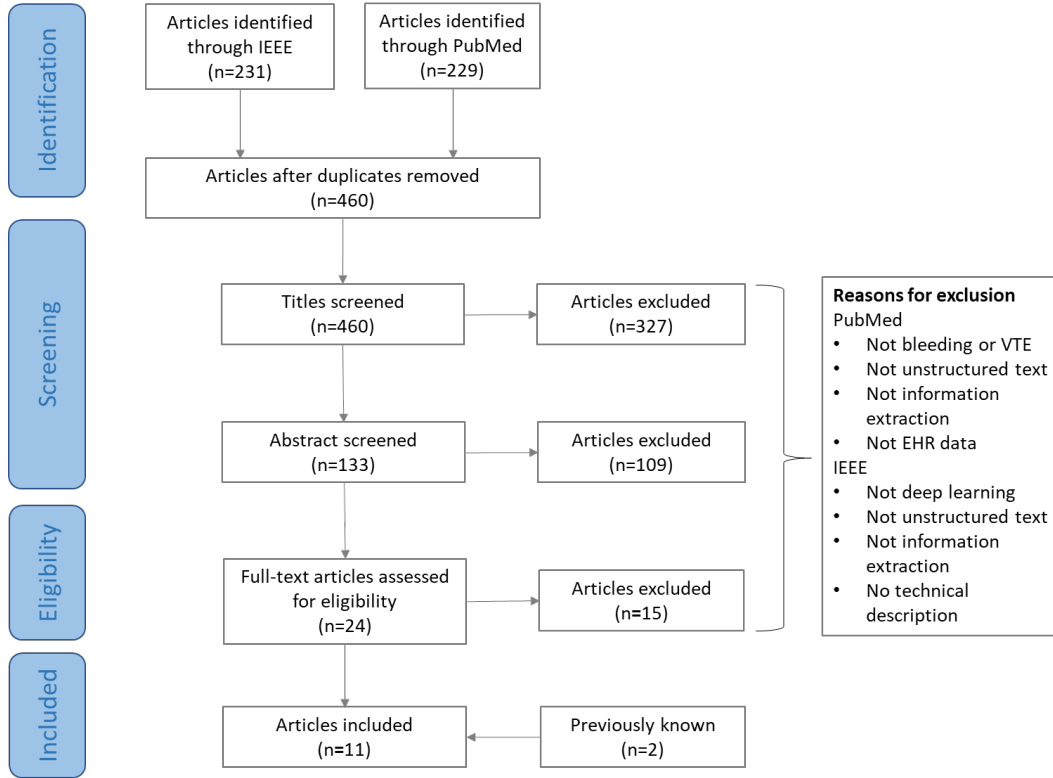


Figure 5.1: Flowchart of identified articles for related work. n : Number of articles.

Two additional articles are described in the section below. These articles were known beforehand and they are included because they are relevant for this thesis.

5.2 Presentation of related work

This section presents the articles that were found in the litteratur research.

Taggart et al. [26] used a rule-based approach and three different machine learning techniques to classify clinical notes as ‘bleeding present’ or ‘bleeding absent’. For the rule-based approach, 990 training notes were used to create hand engineered rules, they e.g. specified that bleeding and hemorrhage are targets and the word *denies* is a modifier of the target. The machine learning approaches were a support vector machine (SVM), a tree-based algorithm, and a convolutional neural network (CNN). These models were trained on a balanced dataset consisting of 450 clinical notes. The SVM and the tree algorithms were trained using bag-of-words feature representation while the CNN used GloVe embeddings. The CNN consisted of 4 hidden layers alternating

between a 1-dimensional convolutional layer and a max pooling layer. All models were tested on an unbalanced test set consisting of 660 notes of which 146 were ‘bleeding present’. The best performing model was the rule-based method which achieved an F-score of 0.743 and an accuracy of 86.1 %. The SVM achieved an F-score of 0.710 and an accuracy of 87.1 %. The tree-based algorithm achieved an F-score of 0.640 and an accuracy of 76.7 %. The CNN achieved an F-score of 0.395 and an accuracy of 0.582.

Rumeng et al. [27] used a neural network model to classify whether sentences from EHRs contained a bleeding event. The model consisted of a combination of a supervised CNN and an unsupervised bidirectional Long Short-Term Memory (LSTM) autoencoder. The model was trained by simultaneously passing the sentences to the CNN and LSTM, after which each of the representations of the sentences were concatenated. This concatenated feature vector was passed to a softmax function which predicted if the input contained a bleeding event. Data from 878 EHR notes that contained 1451 positive sentences was used. 1451 positives and 1451 negative sentences were used to train the model. The word embeddings were pre-trained on 4.7 million EHR notes using the GloVe model and fine-tuned during training. The model achieved an F1 score of 0.938. Moreover, the model was tested using 6 unprocessed EHR notes with 2345 sentences of which 64 sentences were positive. Here, the model achieved an overall accuracy of 93.8 %.

Tian et al. [28] used a rule-based approach for classifying narrative radiology reports for indications of VTE. They selected a random sample of 4000 reports which were manually coded by clinical experts as either positive or negative for deep vein thrombosis (DVT) or pulmonary embolism (PE). 2788 were designated as training set and 1212 as test set. Near perfect inter annotator agreement ($K=0.98$) was achieved on a subset of 20% of the reports. Preprocessing consisted of breaking into sentences at periods, converting to lowercase, lemmatization, and removing punctuation. For each of DVT and PE, a list of unigrams indicating DVT or PE (e.g. *DVT*), and bigrams of pathologic manifestation and anatomic reference (e.g. *clot - brachial*) was created. A list of negation modifiers was created to not classify e.g. historical or

negative cases as positives. Lastly, a set of classification rules were defined to determine if a given sentence contained a positive reference to DVT or PE using the unigrams, bigrams and negations. The accuracy of the classifiers on the test set gave 0.94 sensitivity and 0.96 specificity for DVT, and 0.94 sensitivity and 0.96 specificity for PE.

Santiso et al. [29] used a LSTM architecture to detect adverse drug reactions in EHRs. The architecture consisted of two separate bidirectional LSTM layers where max pooling was applied to one of them and attention pooling to the other. The outputs from the two different pooling layers were concatenated and given as input to a softmax function. Santiso et al. used 2 different datasets to train and evaluate the results. The first dataset contained 75 EHRs (~41,500 words) and the other contained 267 EHRs (~158,000 words). The datasets were preprocessed by changing words to lower cases and replacing digits with *DG*. Moreover, data was lemmatized and they trained their own word embeddings using GloVe on a separate dataset of 190,130 EHRs (107,884,773 words). The network achieved an F-score of 0.719 on positive cases and 95.4 on negative cases on the first dataset. On the second dataset an F-score of 0.733 on positive cases and 0.965 on negative cases was achieved.

Rocheftort et al. [30] trained two different SVM models to classify narrative radiology reports as either VTE present or VTE absent. The first SVM model was trained to classify DVT, and the second to classify PE. The training set consisted of 2000 reports of which 324 (16.2 %) included DVT cases and 154 (7.7 %) included PE cases. Rocheftort et al. used a bag-of-words approach with unigrams and bigrams to create the features of the model. For preprocessing, they lowercased and removed punctuation and extra whitespaces. The DVT classifier gave a sensitivity of 0.8 and a positive predictive value (PPV) of 0.89. The PE classifier gave a sensitivity of 0.79 and a PPV of 0.84.

Chen [31] used an Attention-based bidirectional LSTM architecture for assertion detection (e.g. negation) in clinical text. The model considered the following assertions for clinical concepts: ‘Present’, ‘Absent (negation)’, ‘hypothetical’, ‘possible’, ‘conditional’, and ‘associated with someone else’. The

model consisted of an embedding layer with pre-trained PubMed+ embeddings, a bidirectional LSTM layer, an attention layer, and an output layer.

The dataset consisted of 426 clinical texts annotated with the 6 assertions. Using five-fold cross validation the model reached a micro-F1 score of 0.922.

Chen et al. [32] classified breast cancer as benign or malignant by extracting information separately from three types of diagnosis reports (ultrasound, X-ray, MRI reports) and two types of context information (medical history, personal information).

Preprocessing consisted of removing periods, parenthesis, and quotation. For each of the three types of diagnosis reports, the authors used two stacked bidirectional LSTMs with attention.

The two types of context information were individually extracted by two bidirectional LSTMs and their outputs concatenated to a single feature. Then, the three features from diagnosis reports and the concatenated feature from context information were combined using elementwise multiplication. Finally, a recurrent neural network layer performed the classification of the breast cancer. The model reached an accuracy of 86.76%.

Deng et al. [33] used an attentive bidirectional LSTM model to extract 8 different obesity-related terms from outpatient records. The dataset consisted of 305 outpatient notes in the form of German free text. There was an inter annotator agreement of 0.79 kappa rate.

Preprocessing consisted of separating the text into phrases. The authors used Conditional Generative Adversarial Networks to generate more positive training data per label than in the original dataset. The data was split in 90% for 5-fold cross validation and 10% for testing. The embeddings were pre-trained on the German Wikipedia corpus and the text corpus. The sequence of embedding vectors were processed by the bidirectional LSTM layer (size 150) followed by an attention layer. A single feed-forward layer then calculated the probability that a phrase belongs to a label. The last layer detected the dependencies between subsequent phrase labels (transition probabilities). It output through the softmax function the label to each word in the phrase. The Viterbi algorithm was applied to find the best label sequence looking at

the probabilities for each label and the transition probability to the following. The model reached an F1 score of 0.609.

Fei li et al. [34] utilized the BERT model for a text classification task on EHRs where named entity mentions (e.g. dyspnea on exertion) were mapped to a term, e.g. ‘60845006: Dyspnea on exertion’. Fei Li et al. trained and evaluated their model on three different annotated datasets. The first dataset called MADE had ~380,000 classes and contained 1089 EHR notes. The second and third dataset were both from PubMed and consisted of 743 and 1500 abstracts, respectively, both with ~11,000 classes. The BERT-model achieved an F1 score of 0.679 on the MADE dataset and 0.894 and 0.931 on the PubMed datasets.

Tien et al. [35] derived and validated an automated electronic search algorithm for identifying occurrences of postoperative DVT, PE, or myocardial infarction from EHRs. Two independent subsets of patients who underwent total hip or knee arthroplasty were randomly selected as training set (n=418) and validation set (n=2857). Using regular expressions to build queries, the EHRs were classified as positive or negative for DVT, PE, or myocardial infarction. Synonyms, abbreviations, medical acronyms, and most common symptoms associated with each complication were entered. A list of exclusion words like ‘rule out’ and ‘negative for’ was made. The algorithm was iteratively refined on the training set. On the validation cohort, the classifier achieved 0.97 sensitivity and 0.99 specificity on venous thrombosis, 0.97 sensitivity and 1.0 specificity on pulmonary embolism, and 1.0 sensitivity and 0.99 specificity on myocardial infarction.

Rajput et al. [36] used a singlechannel and multichannel CNN model to predict disease status from free text of EHRs. The singlechannel model consisted of a Word2Vec embedding layer which learned the word embeddings during training, a convolutional layer of size 8, a maxpooling layer, a flattening layer, a fully connected layer of 10 neurons, and a single-neuron fully connected output layer.

The multichannel model consisted of 3 different singlechannel models of different sizes of their convolutional layer (4, 6, and 8). The 3 models were then concatenated at their flattening layers. A fully connected layer of size 10 followed by a single-neuron fully connected layer gave the output.

The dataset consisted of (I) smoking status detection, (II) co-morbidity of obesity, and (III) cardiovascular disease risk prediction. Preprocessing consisted of removing stop words, conversion to lowercase, stemming, and tokenization. Training was performed using a binary cross entropy loss function with the Adam optimizer with 4-fold cross validation.

The singlechannel model performed with accuracies of (I) 85.62 %, (II) 89.88 %, and (III) 80.75 % on the three datasets, while the multichannel performed with accuracies of (I) 85.62 %, (II) 88.06 %, and (III) 79.28 %.

Table 5.1 summarizes the findings of the literature review.

Author, year (Database)	Subject (number of classes)	Main methods	Dataset	Results
Taggert et al., 2008 (PubMed)	Bleeding detection (2 class)	I) Rule-based II) SVM III) Decision tree III) CNN	990 clinical notes	I) F1: 0.743 II) F1: 0.710 III) F1: 0.640 III) F1: 0.395
Runmeng et al., 2019 (known beforehand)	Bleeding detection (2 class)	Combined CNN-LSTM network	878 EHR notes	F1: 0.938
Tian et al., 2017 (PubMed)	I) DVT (2 class) II) PE (2 class)	Rule-based	4000 radiology reports	I) Sensitivity: 0.94 Specificity: 0.96 II) Sensitivity: 0.94 Specificity: 0.96
Santiso et al., 2019 (IEEE)	Drug reaction detection (2 class)	Bidirectional LSTM	I) 75 EHRs II) 267 EHRs	I) F1 pos. cases: 0.72 F1 neg. cases: 0.95 II) F1 pos. cases: 0.73 F1 neg. cases: 0.97

Rocheffort et al., 2014 (PubMed)	I) VTE detection II) PE detection (2 class)	SVM	2000 clinical reports	I) Sensitivity: 0.8 PPV: 0.89 II) Sensitivity: 0.79 PPV: 0.84
Chen, 2019 (IEEE)	Assertion detection (6 class)	Bidirectional LSTM	426 clinical text	Micro-F1: 0.922
Chen et al., 2018 (IEEE)	Breast cancer detection (2 class)	Bidirectional LSTM	Not provided	Acc.: 86.8 %
Deng et al., 2019 (IEEE)	Extract obesity-related terms (8 class)	Bidirectional LSTM	305 outpatient notes	F1: 0.609
Fei li et al., 2019 (known beforehand)	Text classification	BERT	I) 1089 EHR notes II) 743 PubMed abstracts III) 1500 PubMed abstracts	I) F1: 0.679 II) F1: 0.894 III) F1: 0.931
Tien et al., 2015 (PubMed)	I) DVT (2 class) II) PE (2 class) II) MI (myocardial infarction) (2 class)	Rule-based	3275 EHRs	I) Sensitivity: 0.97 Specificity: 0.99 II) Sensitivity: 0.97 Specificity: 1 II) Sensitivity: 1 Specificity: 0.99
Rajput et al., 2018 (IEEE)	I) Smoking status detection II) Co-morbidity of obesity III) Cardiovascular disease risk prediction	a) Singlechannel CNN b) multichannel CNN)	Unknown number of EHRs	I.a) Acc.: 85.62 % II.a) Acc.: 89.88 % III.a) Acc.: 80.75 % I.b) Acc.: 85.62 % II.b) Acc.: 88.06 % III.b) Acc.: 79.28 %

Table 5.1: Related work summerized

5.3 Key takeaways

The related work research yielded two papers that have investigated bleeding detection in EHRs using NLP algorithms. Taggert et al. tested a rule-based method, a SVM, a decision tree, and a CNN. The rule-based method yielded the best result (F1: 0.743) while the CNN yielded the worst (F1: 0.395). The other study similar to the objective of this thesis, by Runmeng et al., used a combined CNN-LSTM network which gave a F1-score of 0.938.

Other studies that do not examine bleeding extraction, though in the same medical domain as this thesis, have mainly used bidirectional LSTMs to yield good results.

The take-away from the related work research is that rule-based methods should not be underestimated as they are often used with good results. CNNs, LSTMs, bidirectional LSTMs, and a single BERT based model are the deep learning techniques used in the researched literature. It is further noticed that good results were achieved on relatively small datasets. The two studies most similar to this thesis used datasets of 878 and 990 EHRs or clinical notes.

The major difference between this study and the researched literature is the language of the EHR notes. The two similar-to-this studies are both developed on English text while the target language of this thesis is Danish. NLP resources are far more developed for English than Danish, and therefore it cannot be assumed that the same results can be achieved for Danish EHRs although the same methods are used. Nonetheless, both results and methods are relevant for this study, as they indicate that NLP algorithms are capable of extracting bleeding events from EHRs with a high accuracy.

5.4 Chapter summary

This chapter introduced literature that have examined problems similar to the one of this thesis. It was found that two other studies have investigated the problem of extracting bleeding occurrences from EHRs with promising results. Other papers show good results on extracting other information than bleeding from EHRs that is in the same domain as this thesis. Primarily rule-based systems, CNNs and bidirectional LSTM have been used to extract information.

Chapter 6

6 Natural language processing with deep learning

This chapter describes the theory of the natural language processing (NLP) methods used for this thesis. It first presents an introduction which gives an overview of the building blocks used for text classification with deep learning. Then, word embeddings and different methods for encoding them are presented. This is followed by a description of four different neural networks for text classification. It then explains how to avoid vanishing gradients during backpropagation, and overfitting. Lastly, two optimizers are described.

In the following sections, bias weights of the network are only included in the explanation if deemed relevant for the understanding.

6.1 Introduction to text classification

Text classification is the task of assigning input text to a predefined class. This can be done using a neural network model which is trained using training samples with associated labels. In the training process, the model uses the training samples to iteratively adjust its internal weights in a way that makes it better at predicting the class of the samples and generalize to unseen samples. Figure 6.1 illustrates the process.

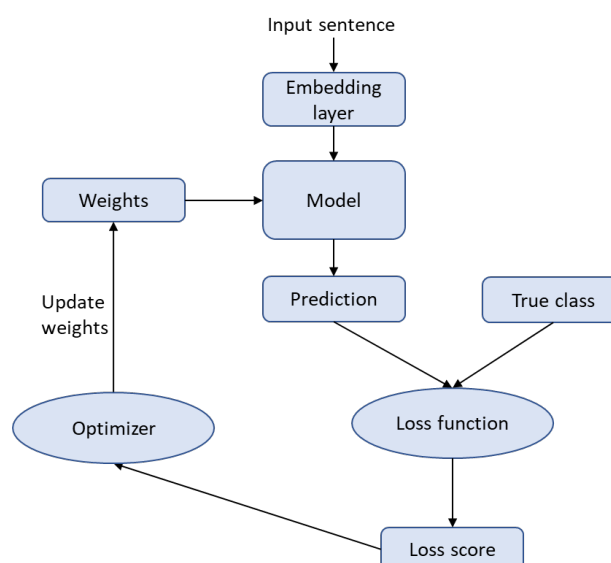


Figure 6.1: Neural text classification process [37]

The sequence of input tokens to the network is first sent through an embedding layer which transforms each token into a numerical representation. This is passed to the model which calculates a class score for each class. The class score is transformed into a measure of the model's certainty on each class by a function, e.g. softmax. The class with the highest score is the prediction of the model. The loss function quantifies the agreement between the predicted class and the true class by calculating a loss score. The gradient of the loss is backpropagated through the network and is used by the optimizer which is responsible for updating the weights in a way that reduces the loss, thereby improving the classification of the model.

For this thesis, the loss score is calculated using the cross-entropy loss function which is coupled with a softmax function.

The softmax function is used on each class score to make the output a probability distribution:

$$\text{softmax}(z) = \frac{e^{z_i}}{\sum_{j=1}^c e^{z_j}} \text{ for } i = 1 \dots c \quad (6.1)$$

where z_i is the i 'th class score, and c is the number of classes.

The cross-entropy loss is calculated as

$$L = - \sum_{j=1}^c y_j \cdot \log(\hat{y}_j) \quad (6.2)$$

where \mathbf{y} is the true probability distribution (a one-hot encoded vector), and $\hat{\mathbf{y}}$ is the predicted probability distribution from the softmax calculation, and c is the number of classes. As the true probability distribution is a one-hot vector, the loss function can be rewritten as

$$L = -\log(\hat{y}_t) \quad (6.3)$$

where \hat{y}_t is the predicted probability of the true class. Substituting $\hat{\mathbf{y}}$ with the softmax function gives

$$L = -\log(\hat{y}_t) = -\log\left(\frac{e^{z_t}}{\sum_{j=1}^c e^{z_j}}\right) \quad (6.4)$$

where z_t is the score of the true class. The gradient of the loss is backpropagated through the network and used by the optimizer to update the weights of the model.

The following sections will describe the building blocks of Figure 6.1 in more detail, first by introducing word embeddings which are used by the embedding layer.

6.2 Word embeddings

One of the most important aspects of NLP is word embeddings. In order to make a machine understand and process free text, it must be converted into numerical values. Word embeddings are real valued vector representations of words, which encode their meaning with similar words having similar vectors.

This section describes two different approaches to transform words into word embeddings. First, the Word2Vec method is introduced which includes the Continuous Bag of Words (CBOW) model. Afterwards the GloVe model is presented.

6.2.1 Word2Vec

Word2Vec was introduced in 2012 by Mikolov et. al. [38]. It learns the meaning of words in an unsupervised way, using a single layer neural network on a large corpus of unlabeled text. The model maps each word to a predefined word embedding size, often between 100 and 300 dimensions, where each point in the embedding is associated with the meaning of the word. The meaning of a word is embedded in the vector by predicting word occurrences near target words using a local window of context. This can be done using the CBOW model.

6.2.1.1 CBOW

The CBOW model encodes the meaning of words in the embeddings by predicting the center word given its context words. Figure 6.2 illustrates this using the sentence:

‘The patient had many bleedings last night’

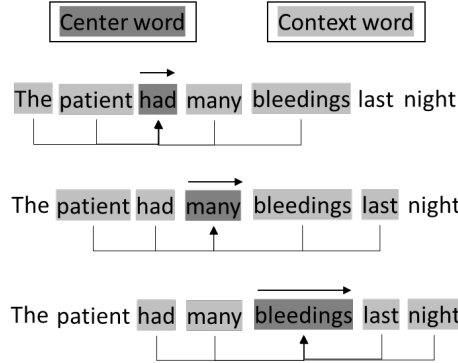


Figure 6.2: Illustration of the principle behind the CBOW model.

In total, the sentence produces seven training samples using a window size of two which consists of one center word paired with its context words.

In theory, using this small example, the CBOW model will output high probabilities for e.g. the word ‘bleedings’ if it is given the context words ‘had’, ‘many’, ‘last’ and ‘night’. The neural network architecture of the CBOW model is illustrated in Figure 6.3.

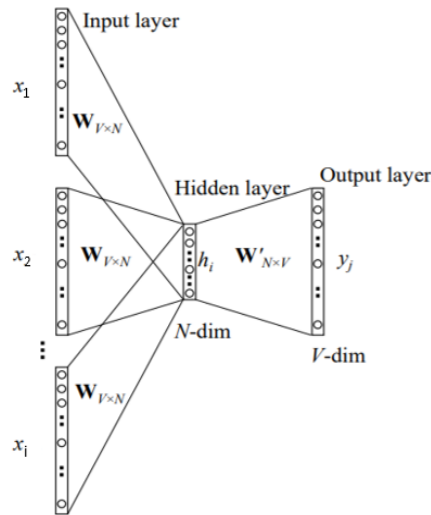


Figure 6.3: Illustration of the CBOW model [39].

The inputs to the network are one-hot encoded context words, $x_i = x_{c-m}, \dots, x_{c-1}, x_{c+1}, \dots, x_{c+m} \in R^{V \times 1}$, where c is the center word index, m is the context window size, and V is the size of the vocabulary. The one-hot encoded context words are pre-multiplied with the transposed context matrix $W \in$

$\mathbb{R}^{|V| \times N}$, where N is the size of the word embeddings, which gives the input word embeddings $v_i \in \mathbb{R}^{N \times 1}$:

$$v_{c-m} = W^T x_{c-m}, v_{c-m+1} = W^T x_{c-m+1}, \dots, v_{c+m} = W^T x_{c+m} \quad (6.5)$$

The hidden layer, $h \in \mathbb{R}^{N \times 1}$, is created by averaging the context words:

$$h = \frac{v_{c-m} + v_{c-m+1} + \dots + v_{c+m}}{2m} \quad (6.6)$$

The output layer, $\hat{y} \in \mathbb{R}^{V \times 1}$ is calculated by pre-multiplying the averaged context vector with the transposed center word matrix, $W' \in \mathbb{R}^{N \times V}$. The softmax function is used to make it a probability distribution:

$$\hat{y} = \text{softmax}(W'^T h) \quad (6.7)$$

The output is a probability distribution over all words of the vocabulary, where input context words will produce high probabilities for words that they are often centered around.

During training, the context words of the weight matrix W and the center words of the weight matrix W' are updated iteratively. Information about the words are encoded in the weights for the network to predict correctly. As semantically similar words are trained to predict the same center words, their word embeddings will be similar as well. Both weight matrices contain all words in the vocabulary and therefore they are averaged to produce the final word embeddings.

The network is trained using a loss function with negative sampling. Negative sampling is implemented in order not to update all the parameters of the network for a single word pair which is very inefficient. The update will be very sparse because most words do not co-occur. Instead of updating all weights, n negative words are chosen to update the weights for. These words are chosen using:

$$P(w_i) = \frac{f(w_i)^{\frac{3}{4}}}{\sum_j^n f(w_j)^{\frac{3}{4}}} \quad (6.8)$$

$P(w_i)$ is the probability of choosing a word w_i from a list of all tokens in a corpus. $f(w_i)$ is the frequency count for word w_i and the denominator sums the total number of tokens. The nominator and denominator are raised to the $\frac{3}{4}$ th power to increase the probability of updating less frequent words and decrease the probability of more frequent words.

Using negative sampling, the loss is calculated by the equation:

$$J(\theta) = \ln(\sigma(w_c^T h)) + \sum_{i=1}^k \mathbb{E}_{j \sim P(w)} [\ln(\sigma(-w_j^T h))] \quad (6.9)$$

where $J(\theta)$ is the objective function which is maximized, w_c is a center word, h is the averaged context words, σ is the sigmoid function, k is the number of random negative samples, and w_j is a random word. The first part of the equation seeks to maximize the probability of co-occurrence between a center word and the averaged context words. The second part minimizes the probability of the averaged context words with a randomly drawn word [39].

The CBOW method uses two additional methods to improve the performance of the model. Firstly, words that often occur in context of each other, e.g. *Barack* and *Obama* are considered as a single term. This is because *Barack* will occur with *Obama* with a high probability, and the predictions of this does not provide much knowledge. Secondly, the authors used subsampling of frequent words because common words like *the* and *I* do not carry significant information. Therefore, during training some frequent words are removed from samples to create more meaningful connections between words [40].

6.2.1.2 Pre-trained fastText embeddings

FastText is an open-source library developed by Facebook [41]. FastText has released Danish word vectors of size 300 trained on Common Crawl [42] and Wikipedia [43] using a modification of the CBOW method. FastText represents words as bags of character n-grams and the word character sequence itself which is surrounded by the boundary symbols $<$ and $>$. Using

an n-gram size of 3, the character n-grams of the word *where* is represented by $\langle wh, whe, her, ere, re \rangle$, and the sequence of the word itself is $\langle where \rangle$. The model is trained to learn a representation for both the n-grams and the word, where the word is represented as the sum of the character n-grams

$$w_i = \sum_{n=1}^N z_n \quad (6.10)$$

where z_n is the n'th n-gram vector of a word and N is the total number of n-gram representations for the word w_i . The model is then trained in the same way as the CBOW model, but the similarity function, s , that was seen in equation (6.9) as

$$s = w_c^T h \quad (6.11)$$

is now not a direct dot product of the two vectors, but the dot product of two vectors represented as a sum of n-grams [44].

The advantage of this model is that it considers the internal structure of a word, its *morphology*, because each subpart of the word is treated as an embedding. This differs from the Word2Vec model which represents the meaning of the word as a whole. Rare words are also modelled better, as their n-gram representations are learned using other words as well. Moreover, the model can create word embeddings for out of vocabulary (OOV) words by representing them as the sum of their n-grams which are learned during training.

The Danish word embeddings are trained using character n-grams of size 5, a window size of 5, and 10 random words are used in the negative sampling objective function.

6.2.2 Global Vectors

The Global Vectors [38] (GloVe) method for creating word embeddings was proposed by Pennington et al. While CBOW uses a sliding window technique to predict the center word from local context words, GloVe utilizes the global statistics of the text corpus to create meaningful word embeddings.

GloVe collects the global statistics about the corpus in a single sweep of the text, creating a co-occurrence matrix which contains the number of times

words co-occur. It typically uses a window size of 5-10. An example of constructing the matrix using a window size of 1 is given in Table 6.1 for the sentence:

‘Patient is bleeding. Patient is treated.’

	Patient	is	bleeding	.	treated
Patient	0	2	0	1	0
is	2	0	1	0	1
bleeding	0	1	0	1	0
.	1	0	1	0	1
treated	0	1	0	1	0

Table 6.1: Example of co-occurrence matrix.

If this co-occurrence matrix is produced from a very large text corpus, it could be said that the probability of the word *is* occurring in the context of *Patient* is

$$P_{i,j} = P_{Patient,is} = P(is|patient) = \frac{X_{Patient,is}}{\sum_k X_{Patient,k}} = \frac{2}{3} \quad (6.12)$$

where $P_{i,j}$ is the probability of the j 'th word occurring in the context of the i 'th word, and X is the co-occurrence matrix where $X_{i,j}$ is the number of times that the j 'th word co-occurs with the i 'th word.

Rather than being based on probabilities, GloVe is based on probability ratios. This is explained in the example below, inspired by the original publication [38]. In the example, centered around the subject of thermodynamic phase, *ice* and *steam* are the target words and $k = solid, gas, water, and fashion$ are the context words used to describe the target words. The goal is to discriminate the target words using the context words.

The example co-occurrence probabilities and probability ratios are shown in Table 6.2.

Probability and Ratio	k = solid	k = gas	k = water	k = fashion
P(k ice)	$1.9 \cdot 10^{-4}$	$6.6 \cdot 10^{-5}$	$3.0 \cdot 10^{-3}$	$1.7 \cdot 10^{-5}$

$P(k steam)$	$2.2 \cdot 10^{-5}$	$7.8 \cdot 10^{-4}$	$2.2 \cdot 10^{-3}$	$1.8 \cdot 10^{-5}$
$P(k ice)/P(k steam)$	8.9	$8.5 \cdot 10^{-2}$	1.36	0.96
$P(k steam)/P(k ice)$	0.11	12	0.74	1.1

Table 6.2: Probabilities and probability ratios describing the target words

It is seen that probability ratios distinguish *ice* and *steam* better than the probabilities do. The ratios show that both the in-domain word *water* and out-of-domain word *fashion* are equally non-significant in discriminating the target words, having a ratio close to 1. The words *solid* and *gas* are shown to be good discriminators.

Apart from probability ratios, Pennington et al. also sought to encode analogies in the vector space, like in Word2Vec, making e.g. the following arithmetic operation on vectors roughly true:

$$king - queen = man - woman$$

This captures dimensions of meaning, i.e. some dimensions of the vectors capture specific characteristics of the word, e.g. gender. The vector differences shown on Figure 6.4 are roughly equal and visualize the underlying concept differentiating e.g. *man* from *woman*, *king* from *queen*, and *brother* from *sister*. This underlying concept could be interpreted as gender.

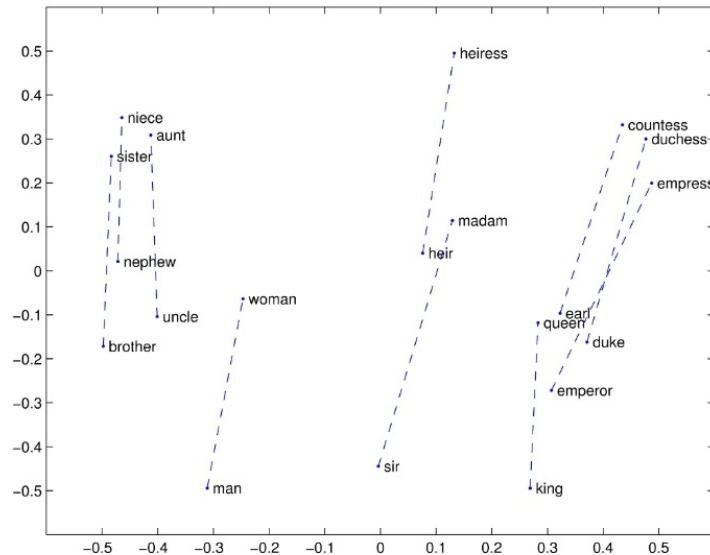


Figure 6.4: Visualization of vector differences representing underlying concepts of words [45]

The GloVe loss function is a least squares function, of which the derivation is explained in details in Appendix A, that sums over all pairs of words in the vocabulary:

$$L = \sum_{i,k=1}^V f(X_{i,k}) (u_i^T c_k + b_i + b_k - \log(X_{i,k}))^2 \quad (6.13)$$

where V is the size of the vocabulary, u_i is the i 'th word embedding of the target matrix u , c_k is the k 'th word embedding of the context matrix c , b_i and b_k are trainable biases, and $X_{i,k}$ is the number of times that the k 'th word co-occurs with the i 'th word. The function f is a weighting function that makes sure that when $X_{i,k} = 0$ then $f(X_{i,k}) = 0$ which means that the loss for that pair will not be calculated:

$$f(x) = \begin{cases} (x/x_{\max})^{\frac{3}{4}}, & x < x_{\max} \\ 1, & x \geq x_{\max} \end{cases} \quad (6.14)$$

Here, x is the evaluation of $X_{i,k}$ and x_{\max} is the cutoff, which is set to 100 in the original paper. The weighting function eliminates the risk of getting an undefined result from $\log(X_{i,k}) = \log(0) = \text{NaN}$ and improves the training speed because training is now only on non-zero values.

Lastly, the weighting function also weights less the co-occurrences that are very infrequent and therefore noisy, as well as the very frequent ones like *it is* that should not dominate the loss over less occurring pairs [46]. The function is plotted on Figure 6.5.

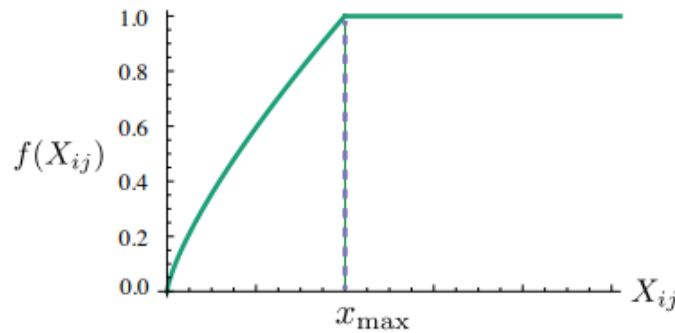


Figure 6.5: Weighting function for GloVe loss function [38]

When the weight matrices, \mathbf{u} and \mathbf{v} , have been optimized to give the lowest loss, they are added to give the final word embeddings.

Summarizing, the training objective of GloVe is to go through all pairs of co-occurring words, minimizing the distance between the dot product of the two word embeddings and the log of their co-occurrence count. Because the difference of logarithms equals the logarithm of a ratio, word embedding differences are associated with co-occurrence probability ratios [45]. These probability ratios are connected to the meaning of words as exemplified earlier.

6.2.2.1 Pre-trained GloVe embeddings

A set of Danish pre-trained GloVe word embeddings was acquired from Rasmussen and Berggrein [47]. The embeddings are trained on 323,122 de-identified EHRs in Danish [48].

6.2.3 Differences between Word2Vec and Global Vectors

Table 6.3 summarizes the differences between the Word2vec and GloVe methods for learning word embeddings as well as the acquired pre-trained word embeddings.

Word2Vec	GloVe
General approach to learning embeddings	
Utilizing sliding window prediction of center word from context words or vice versa.	Utilizing global corpus statistics on co-occurrence.
Words that share context are located closely in the word embedding space.	Differences between words in the embedding space are associated with co-occurrence probability ratios.
Same training examples (windows) might be seen several times and training scales with corpus size.	Efficient usage of statistics ensures fast training after one pass over the corpus collecting co-occurrence counts.

Encodes dimensions of meaning.	Encodes dimensions of meaning.
Acquired pre-trained embeddings	
FastText by Facebook trained on Common Crawl and Wikipedia.	Trained by Rasmussen and Berggrein on 323,122 de-identified EHRs in Danish
Embeddings have 300 dimensions	Embeddings have 100 dimensions
Trained with a modification of the CBOW method, also representing words as bags of character n-grams. Can create embeddings for OOV words.	Trained with standard GloVe objective

Table 6.3: Summary of the Word2vec and GloVe methods and acquired pre-trained embeddings

6.3 Neural networks for text classification

To understand the meaning of a sentence, one must study the words' relation to each other, e.g. by exploring the word order and proximity which can reveal specific patterns. This can be done using two types of information filtering: spatial filtering and temporal filtering. In spatial filtering, relations are studied across space, while temporal filtering studies relations across time. In spatial filtering, data can be viewed through a constant width window using a filter, e.g. by looking at two words a time. With temporal filtering, data can extend in an unlimited amount and be analyzed in its full context. Text can be processed spatially using convolutional neural networks (CNNs) and temporally using recurrent neural networks (RNNs). CNNs and three types of RNNs are described below.

6.3.1 Convolutional neural networks

The main building block of a CNN is the convolutional neural network layer (conv-layer). It uses a set of learned filters to recognize patterns in input data. These filters consist of weights that are learned during training using back-propagation. The filters slide across sentences and extract features that they have learned to be important. Filters convolve over an input sentence in a one-dimensional manner as illustrated in Figure 6.6.

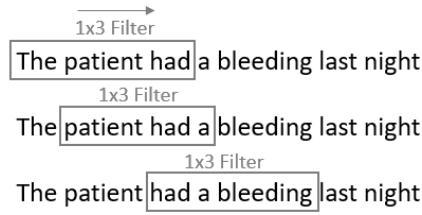


Figure 6.6: A 1x3 filter slides over a sentence

As shown, a filter of size 1x3 slides across a sentence with a stride of 1, meaning that it moves one word to the right at every step. The shape of the filter is described as one-dimensional because it convolves across the sentence in a one-dimensional way from left to right. The term *one-dimensional* does not refer to the actual shape of the filter, though, because words are represented as word embeddings, which by themselves have a dimension. For the example in Figure 6.6, this means that the shape of the filter would be 300x3 if the words are represented with 300-dimensional word embeddings. Figure 6.7 illustrates the process in more detail for a word embedding of 4 dimensions and a filter size of 3.

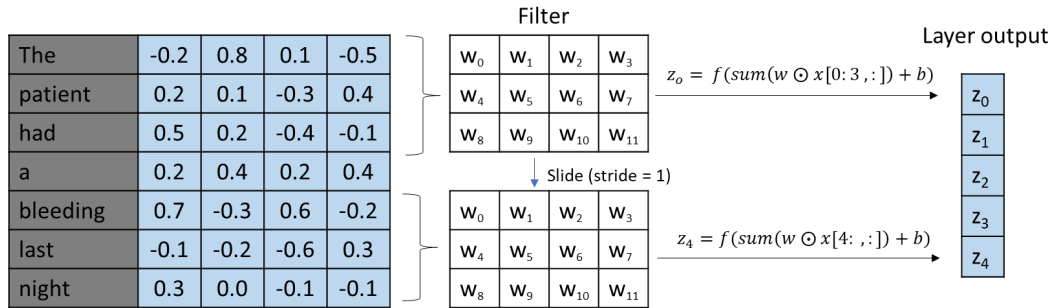


Figure 6.7: Illustration of a filter sliding across a sentence

The figure shows how the filter covers the vector representations of three words in one go. In general, this means that a filter, w , with a window size of ws is defined as $w \in \mathbb{R}^{ws \times d}$, where d is the dimension of the word embedding. It is seen that for each filter at a specific position, e.g. ‘The patient had’, an elementwise product is calculated between the words’ embeddings and the filter weights after which the elements are summed. This is added to a bias and input to an activation function, f , resulting in a single value, z_0 . Afterwards, the filter shifts by one word (stride = 1), and a new value is calculated.

For this example, the input sentence has a length of seven which, when applied to a filter $w \in \mathbb{R}^{3 \times d}$, creates an output of size 5 - this is called the feature map.

In general, the size of the feature map is $Z \in \mathbb{R}^{n-h+1}$, where n is the number of input words and h is the window size of the filter, and each element of Z is calculated by

$$z_i = f(\text{sum}(w \odot x_{i:i+h-1}) + b) \quad (6.15)$$

where f is the activation function [49].

When stacking conv-layers, the subsequent layer takes the feature maps of the preceding layer as input. Padding can be used to preserve the input dimensionality in the subsequent layers. With padding, the input is regulated by adding zero-vectors to the input sentence in a way that preserves the dimension in the subsequent layers.

Figure 6.7 only shows the output when one filter is applied to the input text, but the number of filters is in practice unlimited. Applying 3 filters to the input will create three different feature maps (see Figure 6.8).

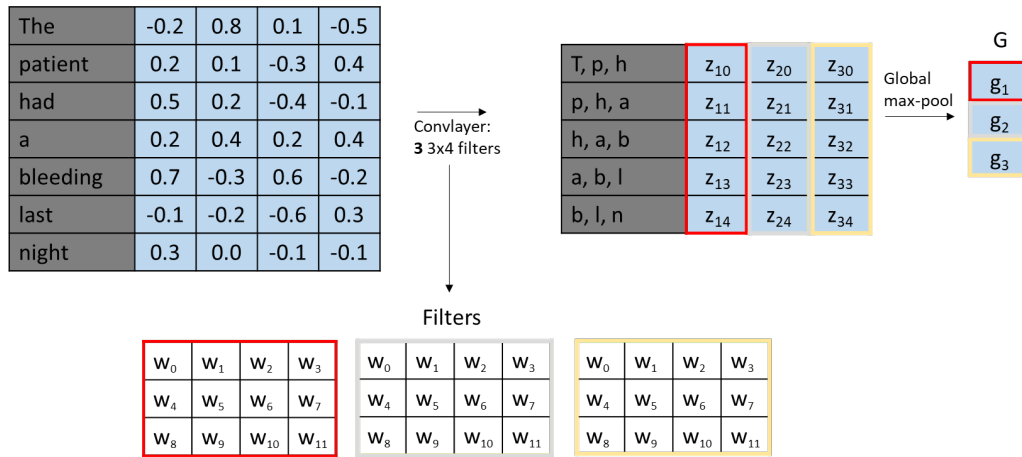


Figure 6.8: Illustration of a conv-layer with three filters

During training, each filter will learn to encode different features to look for in the input data. Each filter can be thought of as an n -gram investigator. When applying a filter of size 3, the filter will slide across the input sentence and look for patterns in 3-grams. As an example, a filter of size 3 might be a

feature detector for the word combination “*had a bleeding*”. This filter will thereby output a high value if this 3-gram is present in the sentence.

The size and number of filters is a hyperparameter that can be tuned depending on the research problem. Using different sizes of filters means that the network will have different n-gram investigators, which is a desirable feature. Moreover, stacking conv-layers increases the level of detail each filter encodes. The deeper into the network a filter is, the more abstract information it will encode.

One way to obtain the final softmax probabilities for each class is to first use a global max-pool over the produced feature maps creating a single value, g_m for each feature map. This value represents the highest filter activation in that feature map. It can be understood as how much the input activated that filter, disregarding where in the input it was found. Concatenating each value gives the resulting final feature map $G = [g_1, g_2, \dots, g_m]$, m being the number of feature maps. To obtain the final probabilities for each class, the output vector of the global max-pooling layer, G , is multiplied by a weight matrix W^S , and input to a softmax function:

$$\hat{y} = \text{softmax}(W^S G + b) \quad (6.16)$$

CNNs typically place different layers in between the conv-layers of which the pooling layer is the most common. It will be described below.

6.3.1.1 Pooling layer

A pooling layer is a building block used to reduce the dimension of a CNN’s feature maps. This is done by subsampling information from each of the feature maps, which reduces the number of parameters in the network and thereby speeds up the computation. Figure 6.9 illustrates the function of three different pooling layers.

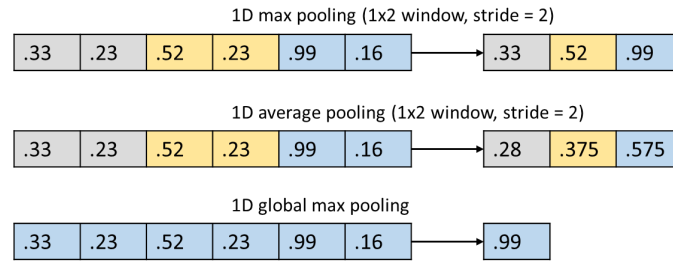


Figure 6.9: Illustration of three different pooling operations

The 1D max pooling layer takes the maximum value of some window size and outputs this number. This is done across the whole feature map with some stride to create the output of the pooling layer. Normally, the stride of the pooling layer is the same as the window size, but this is a hyperparameter that can be tuned. The average pooling layer calculates the average of the window size instead of taking the maximum value. Global max pooling takes the max value across the whole feature map. Max pooling works well because it helps the network focus on the most prominent features of some subsection of the input sentence. Average pooling preserves most of the meaning of a subsection but with less parameters which mitigates the risk of overfitting.

In general, pooling layers are useful because they process the data by looking at the relationship between words i.e. by taking an average of neighbor words, and this is a good way to understand language because the meaning of words often is defined by their neighbors.

6.3.1.2 Gradient-weighted Class Activation Map

Gradient-weighted Class Activation Map [50] (Grad-CAM) is a technique which can be used to interpret the inner workings of CNNs. Grad-CAM calculates the gradient of a class score with respect to the feature maps produced by the conv-layers. This is used to create a weighted distribution that quantifies the importance of each word in the classification of a *specific* class. Figure 6.10 visualizes the method with a simple CNN consisting of a single conv-layer with 3 filters and a filter size of 1.

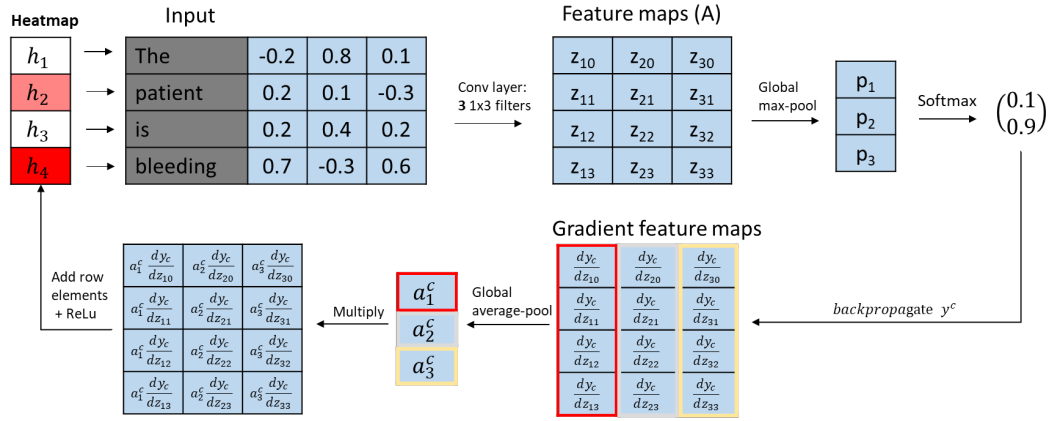


Figure 6.10: Illustration of the Grad-CAM method. The colors of the heatmap indicate the importance of a word in the classification of the sentence, dark red indicating high importance and white not important.

As seen in Figure 6.10, after the forward pass of the CNN, the gradient feature maps are calculated as the gradient of a specific class prediction with respect to the feature maps, after which a global average-pool is taken on each of the three gradient feature maps. The resulting value, a_k^c , captures the importance of each feature map. Formally, this is calculated as

$$a_k^c = \frac{1}{Z} \sum_{i=0}^Z \frac{\partial y^c}{\partial A_i^k} \quad (6.17)$$

where a_k^c is the averaged importance of a feature map k for a class c . $\frac{\partial y^c}{\partial A_i^k}$ is the gradient of the score for a class, y^c , with respect to an index, i , in the feature map activation A^k of a convolutional layer. Z is the number of indices, i , in the feature map k . To obtain the final heatmap, H , each gradient feature map is scaled with its corresponding importance, a_k^c , which is followed by a ReLU activation function:

$$H = \text{ReLU}\left(\sum_k a_k^c A^k\right) \quad (6.18)$$

The ReLU activation function is used to include only those values which has a positive influence on the class prediction, i.e. if these values increase so does the class score of y^c . The resulting heatmap is of the same size as the convolutional feature maps (4x1 on Figure 6.10). This means that if the conv-layer

in Figure 6.10 had used a filter size of more than one, the feature maps would have been downsampled, and thereby each heatmap value, h_i , would describe more than just a single word. E.g. when using a filter size of two, each heatmap value describes two words.

6.3.2 Simple recurrent neural networks

A simple RNN is a type of neural network that performs well on temporal data like e.g. words in a sentence. There are many different uses of recurrent neural networks as shown in Figure 6.11.

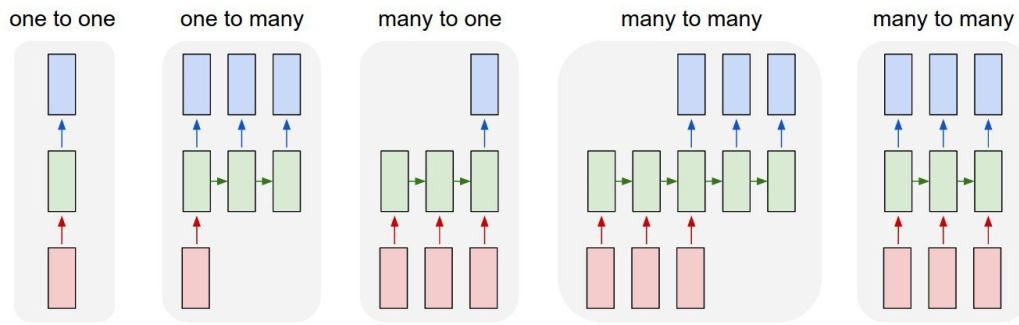


Figure 6.11: Different types of RNNs [51]

A one-to-one RNN is used for classifying one representation of e.g. a word or an image. A one-to-many RNN can be used for image captioning where the vector representation of an image is transformed into multiple words. The many-to-one RNN can be used for classifying a sequence of words into a class. This is the text classification objective of this thesis. Finally, Figure 6.11 shows two different many-to-many RNNs. The left example might be used for machine translation while the right example could be used for named entity recognition. As this thesis deals with a text classification problem, the following examples will be given based on this.

Figure 6.12 shows the simplest representation of a RNN for a text classification problem.

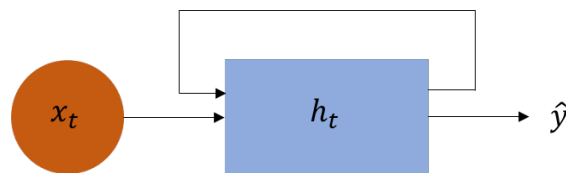


Figure 6.12: Simple RNN cell

x_t is the t 'th input word embedding, h_t is the t 'th timestep of the hidden layer, and \hat{y} is the prediction. For each timestep t , a new word embedding is sent through the hidden layer where the weights are preserved and reused in the loop together with the new input. I.e. the weights are tied between timesteps, and the inputs at different timesteps are passed through the same linear and non-linear layers. This means that the hidden layer of a RNN works as a memory that holds the information of, in theory, all the previous inputs.

Equations (6.19) through (6.21) show the input and mathematical operations performed by the RNN [52].

$$\mathbf{x}_1, \dots, \mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}, \dots, \mathbf{x}_T \quad (6.19)$$

$$\mathbf{h}_t = \tanh(W^h \mathbf{h}_{t-1} + W^x \mathbf{x}_t) \quad (6.20)$$

$$\hat{y} = \text{softmax}(W^s \mathbf{h}_T) \quad (6.21)$$

Equation (6.19) defines the T word embeddings. Equation (6.20) defines the mathematical operation of the hidden layer which is repeated T times for as many timesteps the network has, corresponding to the number of input words. W^h is the hidden layer weights, W^x is the input weights, \mathbf{h}_{t-1} is the previous hidden state, and \mathbf{x}_t is the input to the t 'th timestep. Equation (6.21) defines the output layer where W^s is the output weights and \mathbf{h}_T is final hidden state.

The operations are better visualized in Figure 6.13 where a simple RNN is unfolded across timesteps for the example sentence '*Patient is bleeding*':

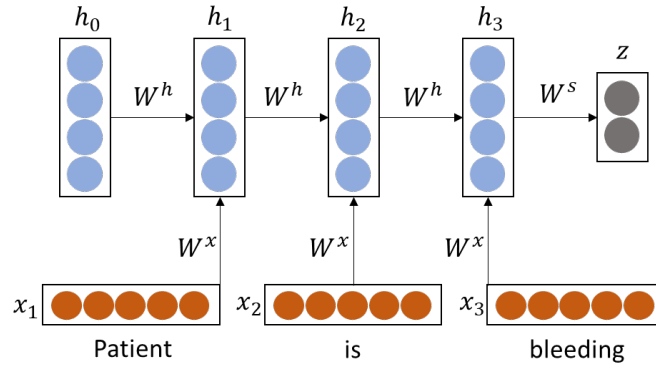


Figure 6.13: Unrolled simple RNN architecture

Figure 6.13 shows an example of a simple RNN with a hidden layer of size D_h that takes a sequence of three word embeddings, x_1, x_2 , and $x_3 \in \mathbb{R}^{D_x \times 1}$, as input. The hidden layer is initialized to zeroes at the initial timestep h_0 . It is then pre-multiplied by the hidden layer weight matrix W^h and added with the input weight matrix W^x multiplied by first word embedding x_1 :

$$q_{0(D_h \times 1)} = W^h_{(D_h \times D_h)} h_{0(D_h \times 1)} + W^x_{(D_h \times D_x)} x_{1(D_x \times 1)} \quad (6.22)$$

q_0 is passed through a non-linearity, e.g. the tanh activation function, which is applied elementwise on the vector, to give the hidden layer of the next time step, h_1 :

$$h_{1(D_h \times 1)} = \tanh(q_{0(D_h \times 1)}) \quad (6.23)$$

This process continues through all timesteps. Finally, a fully connected layer to h_T with the softmax function gives the final predicted probability distribution.

6.3.3 Extensions of the simple recurrent neural network

Simple RNNs suffer from the vanishing gradient problem. The vanish gradient problem is when the network cannot update its weights based on early timesteps because the backpropagated gradient vanishes. This limits the learning of long-term dependencies. This is further described in section 6.5.

The Gated Recurrent Unit (GRU) and Long Short-Term Memory (LSTM) network were specifically designed to deal with vanishing gradients to learn

long-term dependencies efficiently. Vanishing gradients are further explained in section 6.5. GRUs and LSTMs use special neurons called gates that control which information can flow into memory neurons. The memory neurons can hold their information over many timesteps, injecting it into the hidden layer when necessary. Because of this, they do not suffer from the vanishing gradient problem.

6.3.3.1 Gated recurrent unit networks

The GRU was proposed by Cho et al. [53] as a way of solving the vanishing gradient problem of simple RNNs and making learning long-term dependencies easier. The GRU uses an update and reset gate to decide which information is relevant to pass on between hidden layer states. Figure 6.14 shows the architecture of the GRU.

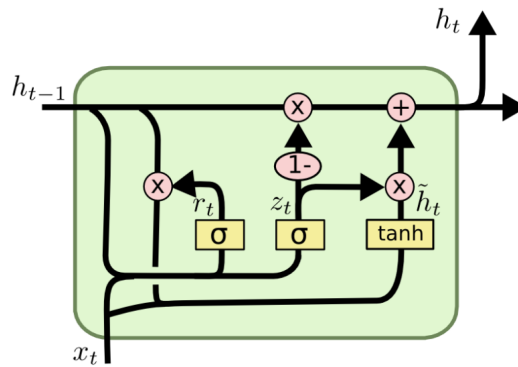


Figure 6.14: The Gated Recurrent Unit [54]. Lines meeting corresponds to addition. Weight multiplication is implicit.

The GRU introduces six weight matrices, two for each sub-architecture on Figure 6.14, instead of the two used by the simple RNN.

The reset gate, r_t , is defined as

$$r_t = \sigma(W^r \cdot x_t + U^r \cdot h_{t-1}) \quad (6.24)$$

The gate learns how much to reset, i.e. forget, of the previous hidden state h_{t-1} , the past context, when calculating the new memory, \tilde{h}_t :

$$\tilde{h}_t = \tanh(W \cdot x_t + U(r_t \odot h_{t-1})) \quad (6.25)$$

In the calculation of the new memory, r_t decides how much of the past context to keep in the summary. If r_t is close to 1, it combines x_t and h_{t-1} . This is what happens every time in a simple RNN. The GRU, on the other hand, has the capacity to drop irrelevant past context, only storing the new word input in the memory if r_t is 0. If an input sentence to the network reads

“The patient was admitted to the hospital Wednesday evening with strong pains, and it was found that he had a bleeding in his abdomen”

the network should learn that the important part of the sentence for classifying a bleeding is the last part: *bleeding in his abdomen*. This means that it should learn to set r_t close to 0 until it reaches the last part of the sentence, then setting it close to 1.

The update gate, z_t , is defined as

$$z_t = \sigma(W^z \cdot x_t + U^z \cdot h_{t-1}) \quad (6.26)$$

It learns the ratio of how much of the previous hidden state and new memory should be carried forward to the next hidden state. The sigmoid function scales the output between 0 and 1.

Finally, the hidden state, h_t , is calculated by combining the calculated new memory with the previous hidden state using the update gate:

$$h_t = z_t \odot \tilde{h}_t + (1 - z_t) \odot h_{t-1} \quad (6.27)$$

The formula uses the update gate to decide how much of the hidden state output will consist of the previous hidden state and how much it will consist of the calculated memory. Notice that if z_t is close to 0, the new hidden state will be close to a copy of the previous hidden state, almost ignoring the new word input. This mitigates the vanishing gradient problem because the network does not wash out the hidden state with the new word at every timestep. z_t should stay close to 0 after the network has iterated through the start of the example

“A bleeding was found in the abdomen of the patient who was admitted to the hospital Wednesday evening with strong pains”.

Since only the first four words are important for the classification of a bleeding, the network should learn to set the update gate, z_t , close to 0 after the first four words. This forces the network to keep the information from these four states and avoid washing out the information by ignoring the current context going forward [52], [55].

6.3.3.2 Long Short-Term Memory networks

The LSTM [56] is similar to the GRU in that it also keeps internal states, but it differs slightly as it has a *cell state* that works as an internal memory which, in addition to h_t , is carried through multiple timesteps. The LSTM is trained to learn what information to keep in the internal memory and when to exploit this knowledge in the hidden state which is the output of the cell. The LSTM cell is illustrated in Figure 6.15.

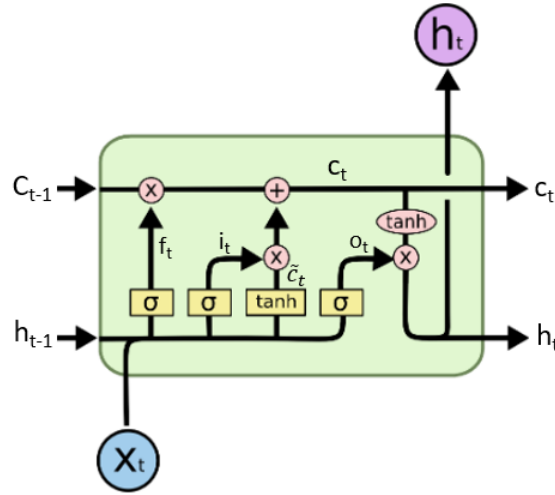


Figure 6.15: The LSTM cell [57]. Lines meeting corresponds to addition. Weight multiplication is implicit.

The hidden state, h_t , and the cell state, c_t , are carried through each timesteps, and they are calculated using four internal gates i , f , o , and \tilde{c} described below:

- The input gate, i ; determines how much information to input into the new cell state
- The forget gate, f ; erases information from the previous timestep

- The output gate, \mathbf{o} ; determines how much information to write to the hidden state
- The new cell content, $\tilde{\mathbf{c}}$; the new content to write to the cell state

The gates are calculated in the following way:

$$i_t = \sigma(W^i x_t + U^i h_{t-1}) \quad (6.28)$$

$$f_t = \sigma(W^f x_t + U^f h_{t-1}) \quad (6.29)$$

$$o_t = \sigma(W^o x_t + U^o h_{t-1}) \quad (6.30)$$

$$\tilde{c}_t = \tanh(W^c x_t + U^c h_{t-1}) \quad (6.31)$$

Each of the above gates are used to calculate the following:

- The cell state, \mathbf{c} ; erases some information from previous cell state and receives new information from the cell content
- The hidden state, \mathbf{h} ; the from the current timestep

These are calculated using:

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \quad (6.32)$$

$$h_t = o_t \odot \tanh(c_t) \quad (6.33)$$

To gain an intuition about the equations of the new cell content, the cell state and the hidden state, it is helpful to consider the i , f , o and \tilde{c} gates as binary values based on what happens at the extremes of their activation function. The values of the gates i , f and o are outputs from the sigmoid function, which means their min and max value is 0 and 1, and the g gate will have extremes at -1 and 1 as it uses the tanh function.

For the cell state in (6.32), it is seen that the previous cell state, c_{t-1} , is multiplied elementwise with the forget gate. If the forget gate is a zero vector, the information from the previous cell state is not carried through to the new

cell state. Each element of the forget gate thereby decides how much information to retain from each index of the previous cell state. The second term of the cell state is an elementwise multiplication of the input gate and the new cell content. Similar to the forget gate, the input gate thereby decides which elements of the new cell content that should be considered in the cell state.

The hidden layer is calculated using the output gate and the tanh of the cell state. Using the tanh activation function, the cell state, c_t , is transformed to values between -1 and 1, while the output gate has values between 0 and 1 because of the sigmoid function. The output gate determines how much of the cell state information that will be revealed in the hidden state and thereby in the output of the LSTM cell. In this way, the LSTM cell, at each timestep, outputs a hidden state that is relevant for the current timestep and allows information to travel through timesteps to be reinjected later using the cell state [52].

6.3.4 Modifications to recurrent architectures

This section describes two possible modifications to recurrent neural network architectures - bidirectionality and stacking of layers.

6.3.4.1 Bidirectional recurrent neural networks

A standard RNN creates meaning of a sentence by processing it going from left to right. Therefore, the hidden state of a specific time step is created using only the previous context, but it turns out that using the context of the future words as well can improve the sentence representation. This is done using a bidirectional RNN that can be described as two separate RNNs. One RNN encodes a representation of the input going from left to right, and the other RNN encodes the input from right to left. The two representations are combined, e.g. by concatenating the forward and backward hidden states, to create the final output of the bidirectional RNN, as illustrated in Figure 6.16.

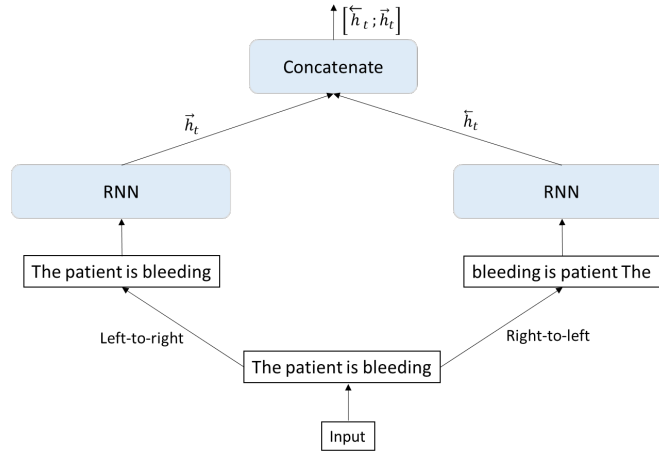


Figure 6.16: Illustration of a bilateral LSTM

In this way, the bidirectional RNN can obtain a richer representation and capture patterns that might have been missed by the one-way RNN. Mathematically this is stated as

$$\vec{h}_t = f(\vec{W}^h x_t + \vec{W}^h \vec{h}_{t-1}) \quad (6.34)$$

$$\tilde{h}_t = f(\tilde{W}^x x_t + \tilde{W}^h \tilde{h}_{t-1}) \quad (6.35)$$

$$\hat{y} = f(W^s[\tilde{h}_t; \vec{h}_t]) \quad (6.36)$$

where the arrow indicates the direction in which the input is being processed. As the network maintains two hidden states with separate weights, it consumes twice the memory and is also more prone to overfitting because of the extra parameters. This method can be used for all RNN architectures, e.g. RNN, GRU and LSTM [52].

6.3.4.2 Stacking layers

Similar to other neural network architectures, RNNs can be stacked by applying multiple RNN layers on top of each other. This allows the network to learn a more complex representation of the input. The lower level layers create simple features that increasingly become more complex in the higher layers. Figure 6.17 illustrates a three-layer stacked RNN.

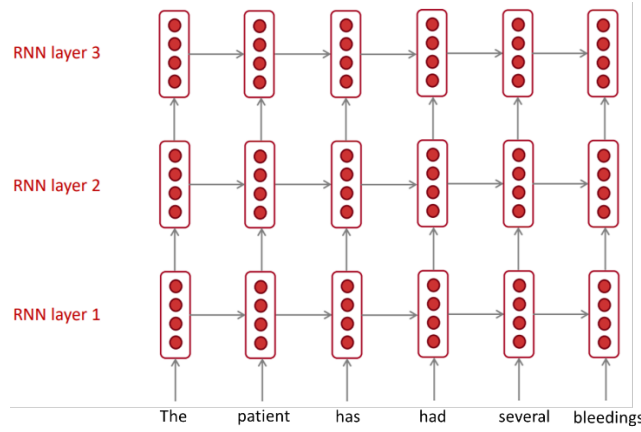


Figure 6.17: Illustration of a 3-layer stacked RNN [58]

In each timestep, the hidden state is passed on to both the next timestep of the same RNN layer and the hidden state for the RNN layer above. For the RNN layer 2 and 3 in Figure 6.17, this means that instead of receiving an input word they will take the hidden state from the timestep below as input, written as

$$h_t^i = f(W^i h_t^{(i-1)} + W^i h_{t-1}^i)$$

where i indicates the i 'th stacked RNN layer, and f is the activation function.

Stacking RNN layers has shown to increase the accuracy of RNN architectures, although there is evidence that the effect decreases when stacking above 4 layers [59].

6.4 Backpropagation

This section introduces backpropagation which is used for updating the weights in both CNNs and RNNs. First, the concept is explained through a simple example. The section then goes on to explain backpropagation through the specialized layers of a recurrent and convolutional network - a recurrent layer and convolutional layer, respectively.

6.4.1 An example of backpropagation

The term backpropagation means the backwards propagation of the loss which, for the objective of the thesis, is the cross-entropy loss coupled with the softmax activation function. After a loss has been calculated in the forward pass of the neural network, an optimizer updates the weights of the

network to minimize the loss. This is done using the chain rule to find the derivate of the loss function with respect to all weights of the network.

Figure 6.18 is used to demonstrate a toy example of backpropagation:

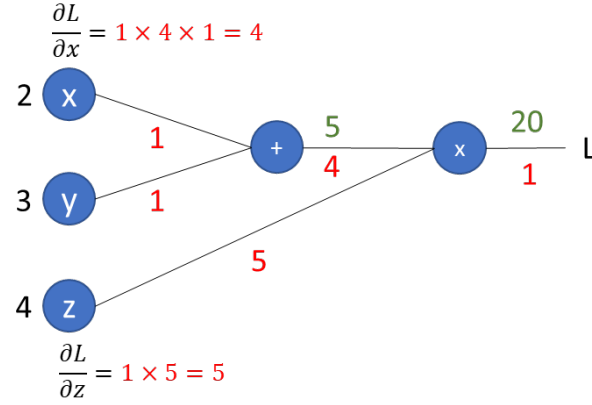


Figure 6.18: Simple example of backpropagation

The mathematical expression visualized on Figure 6.18 is $L = z \cdot (x + y)$. It can be divided in the following dependencies:

$$L = z \cdot a \rightarrow a = x + y \quad (6.37)$$

where a is a placeholder. The chain rule states that the derivative of L with respect to x can be found by multiplying the derivative of L with respect to a by that of a relative to x :

$$\frac{\partial L}{\partial x} = \frac{\partial L}{\partial a} \frac{\partial a}{\partial x} \quad (6.38)$$

Using the chain rule, the gradient travels downstream by multiplying the local gradients by the downstream gradient (DG). On Figure 6.18 the values of the forward calculations are marked in green and those of the local gradients marked in red. The first local gradient is $\frac{\partial L}{\partial L} = 1$ which then becomes the DG. The second local gradient is $\frac{\partial L}{\partial a} = z = 4$ and DG becomes $DG = 1 \cdot 4$. The third local gradient is $\frac{\partial a}{\partial x} = 1$ and DG becomes $DG = 1 \cdot 4 \cdot 1 = 4$ which is the final gradient of L relative to x . This means that when x is increased by 1, L is increased by 4, all else equal.

6.4.2 Backpropagation for a simple RNN

In this section, the math for calculating the gradient of the loss with respect to the weight parameters of a simple RNN, W^s , W^h , and W^x , is presented.

To calculate the gradient, one must look at the dependencies for each of the parameters after which the chain rule can be used to find the gradient as demonstrated in section 6.4.1.

6.4.2.1 Gradient with respect to W^s

The dependencies for W^s as defined in equation (6.3) and (6.21) are:

$$L = -\log(\hat{y}_i) \rightarrow \hat{y} = \text{softmax}(z) \rightarrow z = W^s h_T \quad (6.39)$$

The derivative of the loss with respect to W^s can be calculated using the chain rule as:

$$\frac{\partial L}{\partial W^s} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial W^s} \quad (6.40)$$

The first two gradients, $\frac{\partial L}{\partial \hat{y}}$ and $\frac{\partial \hat{y}}{\partial z}$ are evaluated together as $\frac{\partial L}{\partial z}$ and it is calculated by taking the derivative of the cross-entropy loss coupled with the softmax function. Only the solution is shown here, the calculations are attached in Appendix B.

$$\frac{\partial L}{\partial z} = \hat{y} - y \quad (6.41)$$

where \hat{y} is a vector of the calculated softmax probabilities for each class and y is the one hot encoded label, which is equal to 1 for the correct class index and otherwise zero.

The third local gradient of (6.40) is evaluated to:

$$\frac{\partial z}{\partial W^s} = h_T \quad (6.42)$$

Therefore, the loss with respect to W^s can be calculated as

$$\frac{\partial L}{\partial W^s} = (\hat{y}_t - y_t) \cdot h_t^T \quad (6.43)$$

h_T needs to be transposed because $(\hat{y}_t - y_t) \in \mathbb{R}^{c \times 1}$, where c is the number of classes, and $h_t \in \mathbb{R}^{D_h \times 1}$ and the result must yield a $c \times D_h$ matrix containing the gradient of the loss with respect to W^s . Appendix C holds the mathematical reasoning for transposing h_T .

6.4.2.2 Gradient with respect to W^x

The dependencies of W^x are:

$$\begin{aligned} L = -\log(\hat{y}_i) \rightarrow \hat{y} = \text{softmax}(z) \rightarrow z = W^s h_t \rightarrow \\ h_t = \tanh(q_{t-1}) \rightarrow q_{t-1} = W^h h_{t-1} + W^x x_t \end{aligned} \quad (6.44)$$

where $t = 1 \dots T$, T being the number of hidden states, because the previous hidden state output and the new input iteratively are sent through the hidden layer.

To calculate the derivative of the loss with respect to W^x , it must therefore be calculated for all timesteps and added. This demands an extensive use of the chain rule as the derivative of the loss with respect to h_{T-1} depends on h_T and h_{T-2} depends on h_{T-1} and so on.

The following equation sums the derivative of the loss with respect to W^x for all hidden states:

$$\begin{aligned} \frac{\partial L}{\partial W^x} &= \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_T} \frac{\partial h_T}{\partial W^x} + \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_T} \frac{\partial h_T}{\partial h_{T-1}} \frac{\partial h_{T-1}}{\partial W^x} + \dots \\ &+ \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_T} \frac{\partial h_T}{h_1} \frac{\partial h_1}{\partial W^x} = \sum_{k=1}^T \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_T} \frac{\partial h_T}{\partial h_k} \frac{\partial h_k}{\partial W^x} \end{aligned} \quad (6.45)$$

where T is the number of hidden states in the RNN.

In the following calculations, each of the derivatives of the chain in (6.45) are calculated and progressively multiplied by the downstream gradient, DG, which is marked in bold throughout the calculations.

The derivative of $\frac{\partial L}{\partial z}$ was presented in (6.41) as $\hat{y} - y$. It becomes the downstream gradient

$$\mathbf{DG}_{(\mathbf{c} \times \mathbf{1})} = (\hat{\mathbf{y}} - \mathbf{y})_{(\mathbf{c} \times \mathbf{1})} \quad (6.46)$$

The derivative of z with respect to the last timestep hidden layer, $\frac{\partial z}{\partial \mathbf{h}_T}$ is calculated as

$$\frac{\partial z}{\partial \mathbf{h}_T} = \mathbf{W}^s T \quad (6.47)$$

The local gradient is multiplied by the downstream gradient to get the new downstream gradient

$$\mathbf{DG}_{(\mathbf{D}_h \times \mathbf{1})} = \mathbf{W}^{sT}_{(\mathbf{D}_h \times \mathbf{c})} \cdot \mathbf{DG}_{(\mathbf{c} \times \mathbf{1})} \quad (6.48)$$

The calculation of the derivative $\frac{\partial h_T}{\partial h_k}$, where $h_k = h_1, h_2, \dots, h_{T-1}, h_T$, is demonstrated by calculating the derivative $\frac{\partial h_T}{\partial h_{T-1}}$. The dependency of h_T on h_{T-1} is

$$h_T = \tanh(q_{T-1}) \rightarrow q_{T-1} = \mathbf{W}^h h_{T-1} + \mathbf{W}^x x_T \quad (6.49)$$

This means that $\frac{\partial h_T}{\partial h_{T-1}}$ can be calculated using the chain rule as:

$$\frac{\partial h_T}{\partial h_{T-1}} = \frac{\partial \tanh(q_{T-1})}{\partial(q_{T-1})} \frac{\partial(q_{T-1})}{\partial h_{T-1}} \quad (6.50)$$

The first gradient is the derivative:

$$\frac{\partial \tanh(q_{T-1})}{\partial(q_{T-1})} = 1 - \tanh(q_{T-1})^2 \quad (6.51)$$

The second gradient is

$$\frac{\partial(q_{T-1})}{\partial h_{T-1}} = W^h{}^T \quad (6.52)$$

The downstream gradient is elementwise multiplied by the first local gradient and the result is pre-multiplied by W^h transposed:

$$\begin{aligned} \mathbf{DG}_{(\mathcal{D}_h \times 1)} &= \mathbf{W}^h{}^T_{(\mathcal{D}_h \times \mathcal{D}_h)} \\ &\cdot (\mathbf{DG}_{(\mathcal{D}_h \times 1)} \odot (\mathbf{1} - \tanh(q_{t-1})^2)_{(\mathcal{D}_h \times 1)}) \end{aligned} \quad (6.53)$$

where the \tanh^2 function is taken elementwise on the input vector and the result is subtracted from a vector of ones. q_{t-1} is the respective input to the tanh function of each hidden state. To evaluate $\frac{\partial h_T}{\partial h_{T-1}}$ the calculation is performed once, twice for $\frac{\partial h_T}{\partial h_{T-2}}$ because

$$\frac{\partial h_T}{\partial h_{T-2}} = \frac{\partial h_T}{\partial h_{T-1}} \cdot \frac{\partial h_{T-1}}{\partial h_{T-2}} \quad (6.54)$$

and so on as the backpropagation goes back through the timesteps.

For the last derivative, $\frac{\partial h_k}{\partial W^x}$, in (6.45) the calculation is demonstrated using

$$\frac{\partial h_T}{\partial W^x} = \frac{\tanh(q_{T-1})}{\partial(q_{T-1})} \frac{\partial(q_{T-1})}{\partial W^x} \quad (6.55)$$

The first gradient has already been evaluated to

$$\frac{\partial \tanh(q_{T-1})}{\partial(q_{T-1})} = 1 - \tanh(q_{T-1})^2 \quad (6.56)$$

The second gradient is

$$\frac{\partial(q_{T-1})}{\partial W^x} = x_T^T \quad (6.57)$$

The downstream gradient (6.53) is elementwise multiplied by the first local gradient (6.56) and the result is multiplied with the second gradient (6.57):

$$\frac{\partial L}{\partial \mathbf{W}^x}_{(\mathcal{D}_h \times \mathcal{D}_x)} = \left(\mathbf{D}\mathbf{G}_{(\mathcal{D}_h \times 1)} \odot \left(\mathbf{1} - \tanh(\mathbf{q}_{t-1})^2 \right)_{(\mathcal{D}_h \times 1)} \right) \cdot \mathbf{x}_t^T_{(1 \times \mathcal{D}_x)} \quad (6.58)$$

where \mathbf{q}_{t-1} is the input to the hidden state and \mathbf{x}_t is the word embedding input to that same hidden state.

6.4.2.3 Gradient with respect to \mathbf{W}^h

Similar calculations as those for \mathbf{W}^x can be done for \mathbf{W}^h :

$$\frac{\partial L}{\partial \mathbf{W}^h} = \sum_{k=1}^T \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_T} \frac{\partial h_T}{\partial h_k} \frac{\partial h_k}{\partial \mathbf{W}^h} \quad (6.59)$$

Only the last derivate $\frac{\partial h_k}{\partial \mathbf{W}^h}$ is different which leads to a change in the last step of calculating the downstream gradient:

$$\frac{\partial L}{\partial \mathbf{W}^h}_{(\mathcal{D}_h \times \mathcal{D}_h)} = \left(\mathbf{D}\mathbf{G}_{(\mathcal{D}_h \times 1)} \odot \left(\mathbf{1} - \tanh(\mathbf{q}_{t-1})^2 \right)_{(\mathcal{D}_h \times 1)} \right) \cdot \mathbf{h}_{t-1}^T_{(1 \times \mathcal{D}_h)} \quad (6.60)$$

where \mathbf{q}_{t-1} is the input to the hidden state and \mathbf{h}_{t-1} is the previous hidden state.

6.4.2.4 Gradient flow visualized

Figure 6.19 shows the gradient flow to the \mathbf{W}^s and \mathbf{W}^h weights in an RNN with three inputs $\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3$. The RNN has been unfolded to better visualize the backpropagation through time.

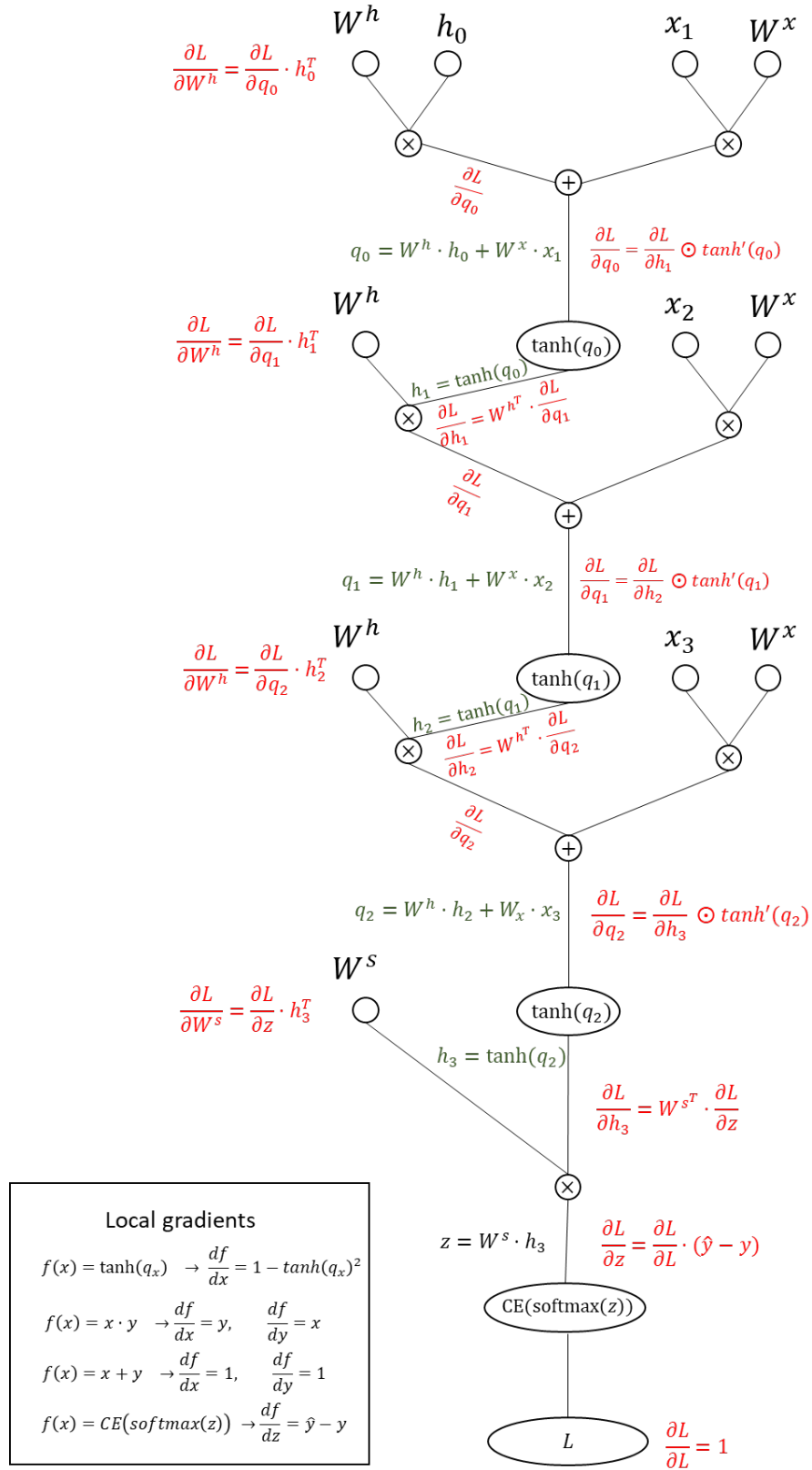


Figure 6.19: Gradient flow for a simple RNN with three inputs. Red text is the downstream gradient, green text is the flow of inputs into each node. CE = cross entropy

Only the final gradients for W^s and W^h are shown for simplicity. The red text shows the downstream gradient and the green text is the result from the forward pass which flows upstream into the nodes, e.g. q_2 flows into \tanh .

For the first timestep, the derivative of the loss with respect to W^h was calculated using the following chain:

$$\frac{\partial L}{\partial W_{(1)}^h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_3} \frac{\partial h_3}{\partial h_2} \frac{\partial h_2}{\partial h_1} \frac{\partial h_1}{\partial W^h} \quad (6.61)$$

For the second timestep the derivative of the loss is calculated as

$$\frac{\partial L}{\partial W_{(2)}^h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_3} \frac{\partial h_3}{h_2} \frac{\partial h_2}{\partial W^h} \quad (6.62)$$

and for the last time timestep

$$\frac{\partial L}{\partial W_{(3)}^h} = \frac{\partial L}{\partial \hat{y}} \frac{\partial \hat{y}}{\partial z} \frac{\partial z}{\partial h_3} \frac{\partial h_3}{\partial W^h} \quad (6.63)$$

For Figure 6.19, this means that the final loss with respect to W^h is achieved by summing the gradients of all the timesteps:

$$\frac{\partial L}{\partial W^h} = \frac{\partial L}{\partial W_{(1)}^h} + \frac{\partial L}{\partial W_{(2)}^h} + \frac{\partial L}{\partial W_{(3)}^h} \quad (6.64)$$

6.4.3 Backpropagation for a simple CNN

This section demonstrates backpropagation through a convolutional layer of a simple CNN of stride 1 and no padding. A toy example with an input sequence of 4 words, x_1 , x_2 , x_3 , and x_4 , is given. Each word is represented by a word embedding which for this example is assumed to be a single numerical value. The convolutional layer has 1 filter, w , of size 2. $*$ signifies a convolutional operation.

Gradient with respect to filter

The forward pass through the convolutional layer is

$$\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} * \begin{bmatrix} w_1 \\ w_2 \end{bmatrix} = \begin{bmatrix} x_1 \cdot w_1 + x_2 \cdot w_2 \\ x_2 \cdot w_1 + x_3 \cdot w_2 \\ x_3 \cdot w_1 + x_4 \cdot w_2 \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} \quad (6.65)$$

where \mathbf{x} is the input and \mathbf{w} is the filter.

It is assumed that the derivate of the loss with respect to the output from the convolutional layer is known and defined as:

$$dy = \frac{\partial L}{\partial y} = \begin{bmatrix} \frac{\partial L}{\partial y_1} & \frac{\partial L}{\partial y_2} & \frac{\partial L}{\partial y_3} \end{bmatrix} = [dy_1 \quad dy_2 \quad dy_3] \quad (6.66)$$

The derivative of the loss with respect to the filter, dw , is then

$$\begin{aligned} dw = \frac{\partial L}{\partial w} &= \frac{\partial L}{\partial y} \cdot \frac{\partial y}{\partial w} = dy \cdot \begin{bmatrix} \frac{\partial dy_1}{\partial w_1} & \frac{\partial dy_1}{\partial w_2} \\ \frac{\partial dy_2}{\partial w_1} & \frac{\partial dy_2}{\partial w_2} \\ \frac{\partial dy_3}{\partial w_1} & \frac{\partial dy_3}{\partial w_2} \end{bmatrix} \\ &= [dy_1 \quad dy_2 \quad dy_3] \cdot \begin{bmatrix} x_1 & x_2 \\ x_2 & x_3 \\ x_3 & x_4 \end{bmatrix} \\ &= \begin{bmatrix} dy_1 \cdot x_1 + dy_2 \cdot x_2 + dy_3 \cdot x_3 \\ dy_1 \cdot x_2 + dy_2 \cdot x_3 + dy_3 \cdot x_4 \end{bmatrix}^T \end{aligned} \quad (6.67)$$

It is seen that the gradient of the loss with respect to the filter, dw , is the convolution of the gradient of the loss with respect to the layer output, dy , on the input, \mathbf{x} (transposing dw to fit dimensions):

$$dw^T = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} * \begin{bmatrix} dy_1 \\ dy_2 \\ dy_3 \end{bmatrix} \quad (6.68)$$

Gradient with respect to input

The same calculations can be made to see that backpropagating through a convolutional layer to find the derivative of the loss with respect to the input, \mathbf{x} , is:

$$\frac{\partial L}{\partial x} = \begin{bmatrix} 0 \\ dy_1 \\ dy_2 \\ dy_3 \\ 0 \end{bmatrix} * \begin{bmatrix} w_2 \\ w_1 \end{bmatrix} \quad (6.69)$$

In this case, the gradient of the loss with respect to the output layer is padded, and a convolution is performed with the reversed filter to find the gradient of the loss with respect to x [60].

6.5 Vanishing and exploding gradients

This section first describes the problem of vanishing and exploding gradients using an RNN as example. The RNN is used as example because the depth of the network grows with the number of input tokens making it vulnerable to the problem, while the CNN developed in this thesis only has a limited depth.

Afterwards, the section explains different methods to avoid the problem.

6.5.1 The problem of vanishing and exploding gradients

In theory, the backpropagation algorithm should update the weight matrices W^h , and W^x for all inputs to the RNN so that dependencies across long text passages are stored. The importance of learning long-term dependencies is shown in the example sentence

‘The test was negative for the patient having an abdominal bleeding’

For the RNN to make the correct classification, that the patient does not have a bleeding, it must relate the word *negative* to the word *bleeding*. This means that the backpropagation algorithm must go through 8 timesteps of the hidden layer to update this dependency. In practice, though, updates based on early timesteps do not always take place. This is caused by the vanishing or exploding gradient problem.

In section 6.4.2 it was seen that the matrix multiplications that have to be made to find the gradient of the loss with respect to the weight W^h are given by

$$\frac{\partial L}{\partial W^h} = \sum_{k=1}^T \frac{\partial L}{\partial h_T} \frac{\partial h_T}{\partial h_k} \frac{\partial h_k}{\partial W^h} \quad (6.70)$$

The term $\frac{\partial h_T}{\partial h_k}$ is responsible for the backpropagation through $T - k$ timesteps of the hidden layer.

In Appendix D, it is shown that the norm of the term $\frac{\partial h_T}{\partial h_k}$ is upper bounded by

$$\left\| \frac{\partial h_T}{\partial h_k} \right\| \leq (\beta_a \cdot \beta_W)^{T-k} \quad (6.71)$$

where β_a is bounded by the derivative of the activation function used in the simple RNN. In the case of a tanh activation function the upper bound of the derivative is 1. β_W is bounded by the norm of the weight matrix W^h [52], [61].

If $\beta_a \cdot \beta_W < 1$ the gradient will gradually vanish and if $\beta_a \cdot \beta_W > 1$ the gradient can explode exponentially. If the gradient vanishes there will be small or even no updates and therefore no learning in the early layers. If the gradient explodes, the updates will become too large and there is a risk of getting overflow (NaN) which causes the training to crash.

6.5.2 Avoiding vanishing and exploding gradients

There are several ways to avoid vanishing and exploding gradients, e.g. by using specific activation functions and by initializing the weights properly. These methods work for both CNNs and RNNs. The intuition behind some of the methods is presented below.

6.5.2.1 Rectified Linear Unit activation function

Using the Rectified Linear Unit (ReLU) activation function reduces the risk of vanishing gradients over using e.g. the sigmoid or tanh activation functions. The three activation functions are written in mathematical form below and visualized on Figure 6.20. Their derivatives are visualized on Figure 6.21.

$$ReLU = \max(0, x) \quad \text{sigmoid} = \frac{1}{1 + e^{-x}} \quad \text{tanh} = \tanh(x)$$

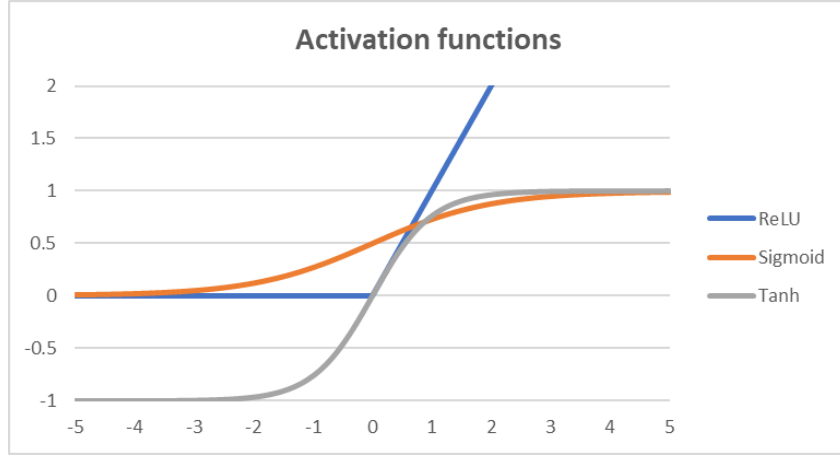


Figure 6.20: The ReLU, sigmoid, and tanh activation functions

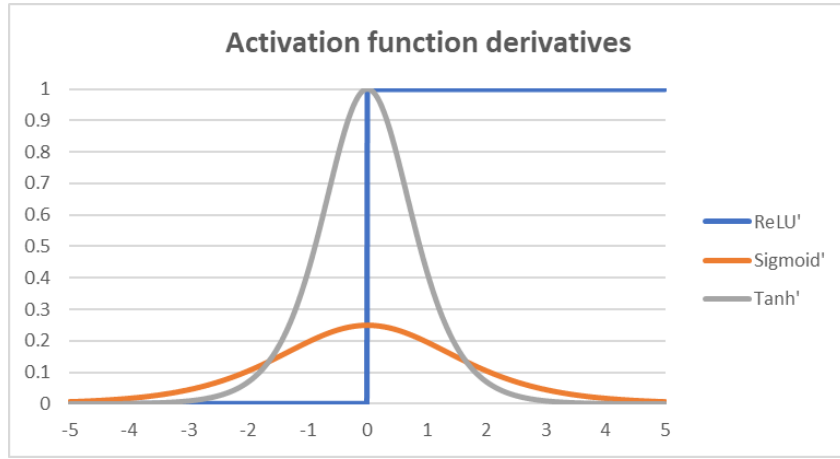


Figure 6.21: The ReLU, sigmoid, and tanh activation function derivatives

The derivative of the sigmoid function can take any value between 0 and 0.25 while it for tanh lies between 0 and 1. This means that sigmoid and tanh will reduce the downstream gradient except in the single case when $\tanh' = 1$. On the other hand, the derivative of the ReLU can only take the values 0 or 1:

$$\frac{\partial}{\partial x} \max(0, x) = \begin{cases} 0, & x < 0 \\ 1, & x > 0 \end{cases} \quad (6.72)$$

This means that the gradient will either be backpropagated fully through the activation function without vanishing or that the neuron will die, i.e. have a gradient of 0 and therefore not update the weights prior to the neuron.

6.5.2.2 Orthogonal initialization

The orthogonal initialization [62] is used by default in Tensorflow for initializing the hidden layer weights of recurrent neural networks [63]. An orthogonal matrix is a square matrix with eigenvalues with absolute values of 1 [64]. This means that the matrix can be multiplied by itself without exploding or vanishing. It can be seen from the following example:

Let a matrix F be factorized by eigendecomposition as

$$F = Q\Lambda Q^{-1} \quad (6.73)$$

where Λ is a diagonal matrix with the eigenvalues placed in the diagonal and Q is a matrix of the corresponding eigenvectors of F . The matrix F is multiplied by itself:

$$F^2 = (Q\Lambda Q^{-1})(Q\Lambda Q^{-1}) = (Q\Lambda) \cdot (Q^{-1}Q) \cdot (\Lambda Q^{-1}) = Q\Lambda^2 Q^{-1} \quad (6.74)$$

Continuing, it can similarly be seen that

$$F^n = Q\Lambda^n Q^{-1} \quad (6.75)$$

which shows that it is the eigenvalues in the diagonal matrix λ that controls the exploding or vanishing properties of the matrix F . If the absolute value of all eigenvalues is smaller than 1, F vanishes, and if the absolute value of any eigenvalue is larger than 1, F explodes. But if the absolute value of all eigenvalues is equal to 1, F is constant.

By initializing the hidden layer weights as on orthogonal matrix, it is ensured that, at initialization, the gradient does not vanish or explode as repeated matrix multiplication by the hidden layer weight matrix is performed during backpropagation through the different timesteps [64].

6.5.2.3 Glorot initialization

Initialization of the weights in CNNs using Glorot initialization is the default in Tensorflow. This is primarily motivated by the problem of saturating neurons. A neuron becomes saturated when the input takes values at the extreme of its activation function. Using the hyperbolic tangent functions

visualized in Figure 6.20 as an example, it was seen that when the input, x , is a large positive number or large negative number, the output is 1 or -1, respectively. In these regions, the gradient of the hyperbolic tangent function is close to zero, meaning that the backpropagated signal through that neuron also will be close to zero.

Saturated neurons can quickly become a problem in deep convolutional neural networks because the variance of the output distribution from a neuron grows with the number of inputs. As the output from one neuron will be the input to neurons in the next layer these values can quickly explode or vanish, making the gradients vanish as well. To avoid this, the weights of a CNN are drawn from a distribution of a certain variance that keeps the variance of neurons in different layers equal. This can be done using Glorot initialization which uses either a normal or uniform distribution [65]. First, Glorot initialization using a normal distribution is explained, then the Glorot uniform initialization.

6.5.2.3.1 Glorot normal

The variance of the output distribution from a neuron grows with the number of inputs:

$$\text{Var}(y) = \sum_i^n \text{Var}(x_i w_i) \quad (6.76)$$

where y is a neuron, x is the input from an earlier layer which is multiplied by a weight w , and n is the number of input connections to y . The term $\text{Var}(x_i w_i)$ can be rewritten as

$$\begin{aligned} \text{Var}(x_i w_i) = & E(x_i)^2 \text{Var}(w_i) + E(w_i)^2 \text{Var}(x_i) \\ & + \text{Var}(x_i) \text{Var}(w_i) \end{aligned} \quad (6.77)$$

where E is the expected value which can be thought of as the mean [66]. If zero mean is assumed, then $E(w_i) = E(x_i) = 0$, and (6.77) becomes

$$\text{Var}(x_i w_i) = \text{Var}(x_i) \text{Var}(w_i) \quad (6.78)$$

Substituting the above into (6.76) and assuming that x_i, w_i are identically distributed, i.e. $Var(x_1) = Var(x_2) = \dots$ and $Var(w_1) = Var(w_2) = \dots$:

$$Var(y) = \sum_i^n Var(x_i)Var(w_i) = nVar(w)Var(x) \quad (6.79)$$

This means that the variance of the output, $Var(y)$, is equal to the input variance, $Var(x)$, scaled by $nVar(w)$, n being the number of inputs. As the goal is to keep the same variance of all layers, $nVar(w)$ should equal 1:

$$nVar(w) = 1 \rightarrow Var(w) = \frac{1}{n} = \frac{1}{n_{in}} \quad (6.80)$$

n_{in} being the number of inputs to the neuron.

From (6.79) it can be seen that

$$Var(w) = \frac{1}{n_{in}} \rightarrow Var(y) = Var(x) \quad (6.81)$$

Initializing the values of w using (6.81), the output variance of the neurons in a layer equals the former layer.

Moreover, it is also wanted to keep the variance of the gradients similar during backpropagation. Using the same equations as above it can be seen that this is accomplished by

$$Var(w) = \frac{1}{n_{out}} \quad (6.82)$$

where n_{out} is the number of output neurons of a layer [67]. The two constraints of the variance of w stated in (6.81) and (6.82) are only satisfied if $n_{in} = n_{out}$. As a compromise, the average of the two is taken which gives the final initialization of w [61]

$$Var(w) = \frac{1}{\frac{n_{in} + n_{out}}{2}} = \frac{2}{n_{in} + n_{out}} \quad (6.83)$$

6.5.2.3.2 Glorot uniform

The above is only valid when w is drawn from a normal distribution.

In Tensorflow, the default initialization of the weights connected to the input is drawn from a uniform distribution with some *limit*. The variance of a uniform distribution, U , is found using

$$\begin{aligned} \text{Var}(U[-limit, limit]) &= \frac{1}{12} (limit - (-limit))^2 = \\ &= \frac{1 \cdot (2 \cdot limit)^2}{12} = \frac{limit^2}{3} \end{aligned} \quad (6.84)$$

where U is a uniform distribution with some limit. The limit which satisfies that $\text{Var}(y) = \text{Var}(x)$ is found using

$$\frac{limit^2}{3} = \frac{2}{n_{in} + n_{out}} \rightarrow limit = \sqrt{\frac{6}{n_{in} + n_{out}}} \quad (6.85)$$

meaning that in Glorot Uniform initialization, weights are drawn from a uniform distribution with $limit = \sqrt{\frac{6}{n_{in} + n_{out}}}$ which makes sure that the variance of the output distribution from neurons in a layer equals the former layer.

6.6 Regularization techniques

Regularization can help a model generalize better by placing constraints on the information it can store. This section introduces weight regularization and dropout which are two different methods to constrain the model.

6.6.1 Weight regularization

Weight regularization constraints the model by penalizing large weights, thereby encouraging small weights. It does this by adding a regularization term to the loss function. The L2 loss is

$$L_{L2} = \lambda \cdot \sum_{i=0}^n w_i^2 \quad (6.86)$$

where λ is the regularization factor, w_i is a weight of the layer, and n is the number of weights in the layer [68]. The regularization is summed for each layer which it is applied to and the loss is added to the loss function.

With large weights, a model can focus on very small aspects of the training data which might be noise and not present in the validation set [69]. By minimizing the regularization term together with the loss function, the model should learn smaller weights and thereby become more general and not overfit the training data. The idea is that if a model can only learn a small number of patterns, it will be forced to focus on the more prominent and general patterns [70].

6.6.2 Dropout

Dropout is a regularization method with the purpose of reducing the risk of overfitting in neural networks [71]. This is done by creating a mask that randomly drops neurons during training with some probability. At every forward pass, a dropout mask is sampled, and neurons are kept with some probability p and removed with probability $1-p$. Dropout is only applied during training because the full potential of the model must be exploited at test time. Therefore, during training, the activations are scaled by a factor $\frac{1}{p}$ to ensure that the expected output of hidden layers are equal during training and test time. Dropout can be used in between fully connected layers as illustrated in Figure 6.22.

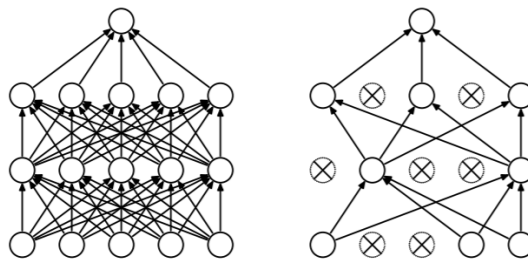


Figure 6.22: Left: Fully connected layers without dropout. Right: Fully connected layers with dropout [71].

Dropout forces the network not to rely on a series of specific neurons as their connections are randomly broken. This can be seen as a way of training an exponential number of distinct sub-networks and approximately combining their predictions at test time [71].

6.7 Optimizers

This section presents two optimizers that update the weights of the neural network using the backpropagated gradient. First, the Adam optimizer is explained, then the Stochastic Gradient Descent (SGD) optimizer with momentum.

6.7.1 Adam

The computations of the Adam optimizer [72] are:

$$m_0 = 0 \quad (6.87)$$

$$v_0 = 0 \quad (6.88)$$

$$t = 0 \quad (6.89)$$

while θ_t not converged **do**:

$$t = t + 1 \quad (6.90)$$

$$g_t = \nabla_{\theta} L_t(\theta_{t-1}) \quad (6.91)$$

$$m_t = \beta_1 \cdot m_{t-1} + (1 - \beta_1) \cdot g_t \quad (6.92)$$

$$v_t = \beta_2 \cdot v_{t-1} + (1 - \beta_2) \cdot g_t^2 \quad (6.93)$$

$$\widehat{m}_t = \frac{m_t}{1 - \beta_1^t} \quad (6.94)$$

$$\widehat{v}_t = \frac{v_t}{1 - \beta_2^t} \quad (6.95)$$

$$\theta_t = \theta_{t-1} - \alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t} + \epsilon} \quad (6.96)$$

end while

where $\beta_1 \in [0,1)$ (default 0.9) and $\beta_2 \in [0,1)$ (default 0.999) are the exponential decay rates for m_t and v_t . In equation (6.96), α is the learning rate and ϵ (default 10^{-8}) is added to avoid division by zero. Notice that a small ϵ can cause large parameter updates depending on \widehat{v}_t , while a large ϵ will cause small updates.

Equations (6.87) through (6.89) initialize the mean and uncentered variance of the gradient and the timestep counter to 0. A timestep corresponds to one batch being sent through the network. While the parameters of the network, θ , have not converged, the Adam optimizer uses the gradient of the parameters on the loss to update the parameters.

In equation (6.91), the parameters of the $(t - 1)$ 'th timestep have been used in the forward pass to calculate the loss of the t 'th timestep. The gradient of the loss with respect to the parameters is calculated through backpropagation and saved in g_t .

m_t of equation (6.92) is estimating the mean of the gradient by calculating a moving average of the gradient weighted by β_1 . With β_1 at its default value, the previous measure counts 90 % while the new gradient counts 10 % in the estimate of the mean.

v_t of equation (6.93) is estimating the uncentered variance by calculating a moving average of the squared gradient. The uncentered variance measures how spread out the gradient is from zero.

Because m_t and v_t are initialized to zero, the estimates are biased towards zero in the beginning of the optimization. Especially, if the β values are set close to 1, there will be bias towards zero for some timesteps. Equations (6.94) and (6.95) perform bias corrections of the two moment estimates. For e.g. \widehat{m}_t , if t is constant and β_1 goes towards 1, then the denominator decreases, and \widehat{m}_t will increase - counteracting the bias. Likewise, when the number of timesteps, t , goes towards 0, the value of term β_1^t increases, the denominator decreases, and \widehat{m}_t will increase - counteracting the bias.

Finally, the update of the parameters is performed in equation (6.96). Assuming that $\epsilon = 0$, the parameter update step is

$$\alpha \cdot \frac{\widehat{m}_t}{\sqrt{\widehat{v}_t}} \quad (6.97)$$

The authors Kingma and Ba call the fraction of equation (6.97) a signal-to-noise ratio. It can be understood as so because \widehat{m}_t is the bias corrected estimate of the mean of the gradient, i.e. the signal. $\sqrt{\widehat{v}_t}$ is the square root of the uncentered variance of the gradient, i.e. the noise. If there is a high signal-to-noise ratio, the update step will be large. If, on the other hand, the noise is high, it means that the uncertainty on the gradient direction is high, and therefore the update step will be small.

6.7.2 SGD with momentum

Momentum is an addition to the SGD optimization method that can help dampen oscillations in the gradient descent and avoid getting stuck in a local minimum.

The computations of the SGD with momentum optimizer is [73]:

$$v_t = \mu \cdot v_{t-1} + \alpha \cdot \nabla_{\theta} L_t(\theta_{t-1}) \quad (6.98)$$

$$\theta_t = \theta_{t-1} - v_t \quad (6.99)$$

where μ is the momentum term that scales the update vector of the past timestep, v_{t-1} , α is the learning rate, and $\nabla_{\theta} L_t(\theta_{t-1})$ is the gradient of the parameters on the loss. Equation (6.98) calculates the update vector of the current timestep, v_t , and equation (6.99) updates the parameters in the opposite direction of the gradient with momentum to minimize the loss. Figure 6.23 shows how the update step is calculated.

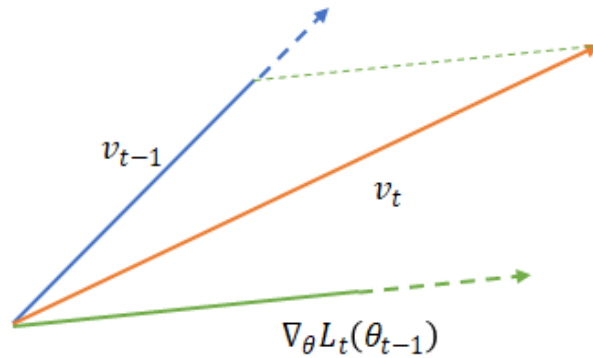


Figure 6.23: Illustration of the SGD with momentum update

Adding a fraction of the past update vector, scaled by μ , to the current gradient, scaled by α , gives the update vector a momentum from the last update. When the gradient for many timesteps has the same direction, the parameter updates become large. When the gradient changes direction, the updates become smaller.

6.8 Chapter summary

This chapter described the theory of the natural language processing methods used for this thesis.

It first introduced two different approaches to transform words into word embeddings - Word2Vec and Global Vectors (GloVe). Word2Vec encodes the meaning of words by utilizing a sliding window prediction of the center word from context words or vice versa. In this way, words that share context will be located closely in the word embedding space. The pre-trained fastText embeddings use a modified Word2Vec method that represents words as bags of character n-grams so that it can avoid out of vocabulary words.

GloVe utilizes global corpus statistics to encode the distances between words in the embeddings space to be associated to co-occurrence probability ratios. These probability ratios are connected to the meaning of words.

The chapter then presented two types of models - spatial filtering models and temporal filtering models.

A convolutional neural network is a spatial model. It uses a set of learned filters to recognize patterns in input data. The filters slide across sentences and extract features that they have learned to be important. The Grad-CAM was introduced as a way of examining which words count towards a prediction.

A recurrent neural network (RNN) is a temporal model. It has a hidden layer through which it passes a combination of each input word embedding and the previous timestep of the hidden layer. This means that the hidden layer works as a memory that holds the information of the whole sequence of inputs.

Two other temporal models, the Gated Recurrent Unit, and Long Short-Term Memory network were presented as some of the solutions to the vanishing gradient problem. They use gates that control which information can flow into memory neurons. The memory neurons can hold their information over many timesteps, injecting it into the hidden layer when necessary and thereby avoiding the vanishing gradient problem.

The backpropagation of the loss for updating the weights of the network was introduced with the RNN. It was described how the vanishing gradient problem can cause the network to not update based on the information from

the first timesteps. Then, an activation function and initialization techniques to mitigate the problem were introduced.

Next, two regularization techniques were described. Regularization can help a model generalize better by placing constraints on the information it can store. The weight regularization technique constraints the model by penalizing large weights, to avoid overfitting. The dropout technique avoid overfitting by dropping random neurons in the network during training.

Finally, the Adam and SGD optimizers for updating the weights based on the backpropagated gradient were described.

Chapter 7

7 Data acquisition and annotation

This chapter first describes the process of acquiring the electronic health record (EHR) data. It then presents the data annotation method, including extraction of the data from the raw xlsx format to Word and cleaning of the data. Finally, an inter annotator agreement is calculated. The pipeline of the data acquisition and annotation is presented on Figure 7.1.

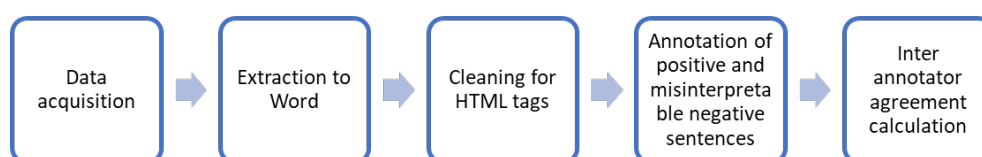


Figure 7.1: Flowchart showing the pipeline of data acquisition and annotation

7.1 Data acquisition

Data was extracted from The Region of Southern Denmark's EHR system called COSMIC. All necessary permissions to work with the data were acquired. In total, 300 EHRs were extracted. To make sure that the EHRs included different patient groups with a high chance of bleeding events, the 300 bleeding EHRs were extracted based on the following patient groups: Internal bleedings, leukemia, gastrointestinal, urinary tract disease, ear-nose-throat and respiratory disease, eyes, and 'others'. Table 7.1 shows the number of EHRs and notes per patient group.

Patient group	Number of patients	Number of notes
Internal bleedings	45	6,078
Leukemia	33	10,340
Gastrointestinal	51	6,968
Urinary tract disease	45	4,409
Ear-nose-throat and respiratory disease	23	3,702
Eyes	65	7,781
Others	38	5,597
Total	300	44,875

Table 7.1: Distribution of extracted EHRs

The ICD-10 diagnoses codes to extract EHRs from the different patient groups can be seen in Appendix E.

Extracted EHRs are contained in the xlsx format where each row contains an EHR note and each column is a descriptor of the note. The following descriptors were extracted:

- The ID of the EHR
- The ID of the note
- Unstructured text

The IDs have been changed to not directly identify a specific EHR and patient.

7.2 Data annotation

This subsection describes how the EHR notes were extracted to Word to facilitate annotation. It then presents the pre-annotation cleaning of the text after which the annotation itself is described. Finally, an inter annotator agreement is calculated.

7.2.1 Extraction to Word

As the EHR database extracts to the xlsx file format which is not easily readable, the text was next extracted to Microsoft Word using python code that copies the EHR ID, note ID, and the unstructured text between the Excel and Word document. Word was chosen because it is well-known by most people, it is easy to use, it was pre-installed on the annotators' computers, and it has a good interface for reading texts. Figure 7.2 shows an example of the unstructured text after extraction to Word.

```
<h1>Plejedata</h1><br><ul><li>Respiration og cirkula-
tion</li><ul><li>Respiration...: Udførte handlinger</li><ul><li>lidt
frisk blødning fra udbuling på h.s, lige over pleuradræn<br/>afvasket, på-
sat allyven bandage<br/>h.arm hævet efter sup.PVK<br/>h.arm eleveret
på pude, opfordret til at bevæge fingrene</li></ul></ul></ul>
```

Figure 7.2: Example of the extracted unstructured text of an EHR

7.2.2 Pre-annotation cleaning

As it can be seen from Figure 7.2, HTML tags are included in the extracted text which makes it harder to read for the annotators. Therefore, the headlines of the text surrounded by `<h1>` and `</h1>` were underlined and the tags were removed. Then, each remaining HTML tag was removed and replaced by a dot and a space. The dots were added to make visible to the annotators that some formatting of the text had been removed. Had the tags been removed without inserting dots, it would be impossible to e.g. separate words that had been bulleted in the raw EHR text. All the HTML tags were cleaned in a way that made it possible to insert them back into the annotated text when needed. The result of the cleaning is seen in Figure 7.3.

Plejedata. . . Respiration og cirkulation. . . Respiration...: Udførte handlinger. . . lidt frisk blødning fra udbuling på h.s, lige over pleuradræn. afvasket, påsat allyven bandage. h.arm hævet efter sup.PVK. h.arm eleveret på pude, opfordret til at bevæge fingrene. . . .

Figure 7.3: Example of an unstructured text cleaned for HTML tags

7.2.3 Annotation in Word

Annotation was done by 12 doctors from Odense University Hospital. Doctors were given a guide (see Appendix F) on how to do the annotation and an expert-annotated EHR as an example before starting the annotation work.

The EHRs were annotated with two different labels:

- **Positive:** Sentences that indicate any kind of bleeding.
- **Misinterpretable negative:** Sentences that mention bleeding without it being a bleeding, e.g. "The patient does not have a bleeding."

It was chosen to annotate 'misinterpretable negative' samples to be able to ensure that the training algorithm will see a sufficient amount of negative text passages that can easily be misinterpreted as a positive sample. This is important for the quality of the training because any text passage not indicating bleeding is a negative sample. Having the algorithm train on many misinterpretable negative samples is meant to ensure that e.g. a sentence with the word 'bleeding' will not automatically be classified as a positive case. It should

force the algorithm to take more aspects than single words into account, e.g. to look at negations, and this should make the model more robust.

In practice, data was annotated using macros in Word. These macros were made to ease the annotation job for the doctors as they can be used to automate repetitive tasks. As an example, if a sentence is to be labeled as positive, the doctor simply presses ALT+1 in the start of the sentence and ALT+2 in the end of the sentence. The text is then surrounded by the tags `##start_bleeding##` and `##stop_bleeding##` and colored in red as seen in Figure 7.4.

(...)appetit. Benægter mavesmerter. Ingen kvalme eller opkastninger. Afføring normal farve, normal lugt. (...)
`##start_bleeding##`Har det sidste år haft 3 episoder med frisk blod i afføring`##stop_bleeding##`.. (...)
Broedtekst . Klinisk kontakt. . . . Anamnese. . . Pt. angiver at være tæt ved sin habituelle tilstand. `##start_negative##`Han har ikke observeret yderligere blødning fra endetarmen.`##stop_negative##` . (...)

Figure 7.4: Example of an unstructured text with annotations tags

The misinterpretable negative annotations were tagged with `##start_negative##` and `##stop_negative##` and colored in blue.

7.2.4 Inter annotator agreement

To calculate an inter annotator agreement, the 12 annotators were asked to annotate the same 120 EHR notes from different EHRs.

As the doctors, per the annotation guide, were asked to annotate sentences that indicate bleeding, their annotations were examined with sentences as the unit constituting the samples in a Fleiss' Kappa [74] measure. All positively annotated sentences were treated as positives, and all other sentences as negatives. The 120 notes contained 1328 sentences.

The Fleiss' Kappa is a method to measure agreement between multiple raters that takes into consideration the agreement by chance. A score of 1 means perfect agreement between the annotators while a score of 0 means no agreement.

The 12 annotators reached a kappa score of 0.75 which according to Landis and Koch [75] is considered substantial agreement. The calculation is referenced in Appendix R.

7.3 Chapter summary

This chapter described the acquisition and annotation of the data extracted from The Region of Southern Denmark's EHR system.

In total, 300 EHRs were extracted from 7 different patient groups prone to bleeding.

The EHR database extracts to the xlsx file format and it was then extracted to Microsoft Word. The HTML tags included in the extracted free text were removed and replaced by a dot and a space in a way that made it possible to insert the HTML tags back into the annotated text when needed.

The doctors were asked to annotate positive samples (sentences that indicate any kind of bleeding) and misinterpretable samples (sentences that mention bleeding without it being a bleeding).

An inter annotator agreement was calculated on the 12 annotators' annotations of the same 120 EHR notes which gave a Fleiss' kappa score of 0.75 was reached which is considered substantial agreement.

Chapter 8

8 Generation of datasets

This chapter describes how the annotated electronic health records (EHR) were divided into two different datasets. The two datasets are constructed using the same annotated EHRs, but the individual samples of the datasets are extracted using two different approaches.

First the structure of an EHR is explained as a basis of understanding why and how the samples are created in two different ways, after which the method of creating and extracting the samples is explained. Moreover, this chapter includes a section describing how data was de-identified. Finally, each of the two datasets are split into train, validation and test sets containing positive, misinterpretable negative, and random negative samples.

The pipeline of the generation of datasets is showed on Figure 8.1.

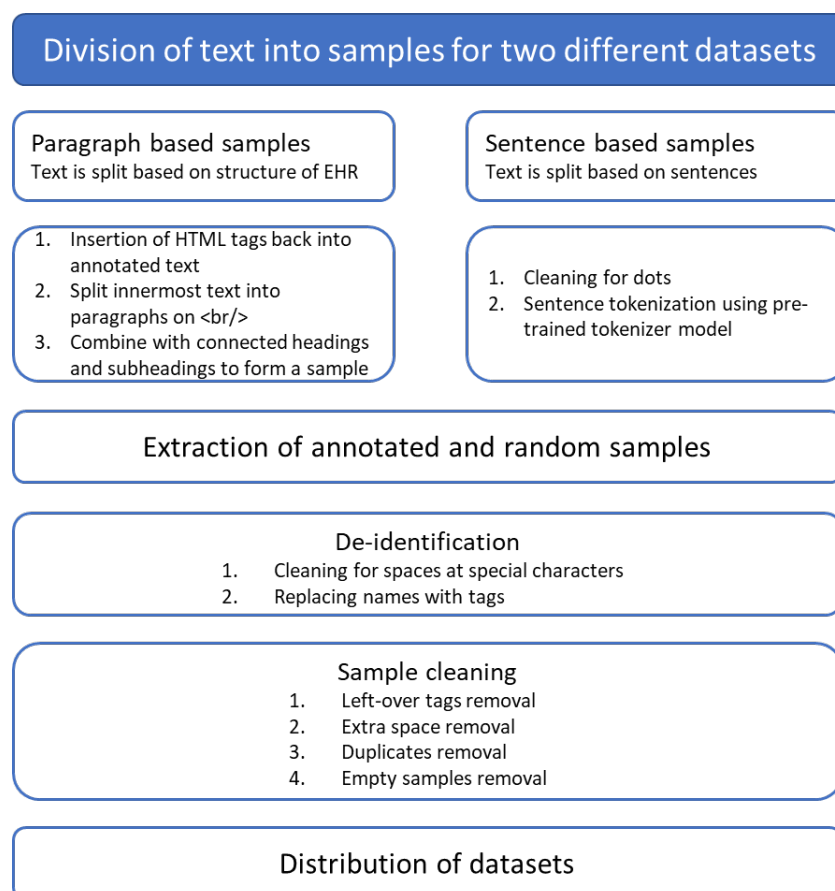


Figure 8.1: Pipeline of the generation of sentence and paragraph dataset

8.1 Structure of EHR text

In COSMIC, the text in the EHR is structured as follows:

- A heading, e.g. ‘*Klinisk kontakt*’ or ‘*Administrativt visitationsnotat*’
- Subheadings, e.g. ‘*Anamnese*’ or ‘*Henvisningsårsag*’
- The innermost text, i.e. free text related to a subheading, e.g. descriptions of bleeding events. Innermost text is defined as text which is not followed by any additional indented bullet points.

Figure 8.2 shows an example of the structure of an EHR.

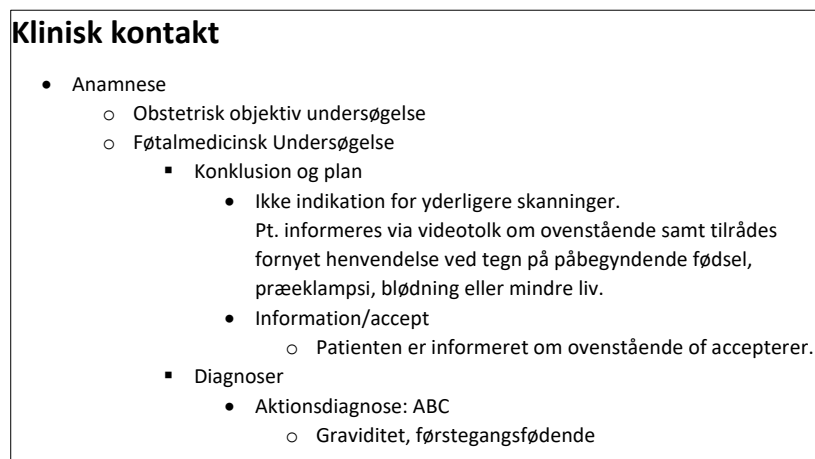


Figure 8.2: Example of the structure of an EHR

This structure creates several possibilities of how the text can be split into samples and given as input to the neural network. In this thesis, two different approaches will be tested.

8.2 Division of text into samples

This section presents two ways of constructing the samples for the dataset based on the structure of the EHRs described above. Each of the sample generation methods will be used to create a dataset.

8.2.1 Paragraph-based samples

In this method, samples are constructed using the structure of the EHR. Each sample consists of a paragraph from the innermost text and the heading and subheadings that this innermost text belongs to. One innermost text can consist of multiple paragraphs, indicated by a new line in the text. Each

paragraph can consist of multiple sentences. The heading and subheadings are appended in front of each paragraph, together constituting a sample. The sentence *‘Ikke indikation for yderligere scanninger’* in Figure 8.2 is one paragraph of the innermost text which belongs to three subheadings and the heading *‘Klinisk kontakt’*. Therefore, the sample becomes: *‘Klinisk kontakt: Anamnese: Føtalmedicinsk undersøgelse: Konklusion og plan: Ikke indikation for yderligere scanninger’*. Each heading and subheading is followed by a colon. This is used to indicate that the preceding word is associated to the following text.

8.2.1.1 Paragraph tokenization

The innermost text is split using HTML tags which was originally part of the extracted text from the EHRs. The HTML tags were inserted back into the annotated text using the code in Appendix G. The innermost text is split into paragraphs using the HTML tag `
` which indicates new line/enter. As an exception, the innermost text is not split if a colon is present before the HTML tag `
`, because it indicates that the text before and after the colon is associated.

8.2.2 Sentence-based samples

In this method, a sentence tokenizer is used to split the text into sentences which constitute the samples. For Figure 8.2 this means that subheadings and free text are split into individual samples, e.g. *‘Konklusion og plan’* is one sample and *‘Ikke indikation for yderligere scanninger’* is another.

8.2.2.1 Sentence tokenization

Two tokenization methods that work on Danish text were explored. One was the Stanza tokenizer [76] which is pre-trained on the Danish Universal Dependencies treebank containing 5,512 sentences and 100,733 tokens. The corpus contains 1,015 types of words that contain both letters and punctuation (e.g. abbreviations) [77]. This should provide a good basis for the tokenizer to have learned splitting sentences that contain abbreviations. The other tokenizer tested was the NLTK tokenizer [78] which is pre-trained on Danish content of around 550,000 tokens from Berlingske Tidende and Weekend Avisen [79].

To give the sentence tokenizers optimal conditions, the data was cleaned for the dots replacing HTML tags by removing ‘.’ from the text. This gives the text a more natural format as shown in the example:

Plejedata. Respiration og cirkulation. Respiration...: Udførte handlinger. lidt frisk blødning fra udbuling på h.s, lige over pleuradræn. afvasket, påsat ally-ven bandage. h.arm hævet efter sup.PVK. h.arm eleveret på pude,John opfordret til at bevæge fingrene.

Figure 8.3: An example of an unstructured text cleaned pre-tokenization for extra dots

After cleaning, a small test was set up to explore the two tokenizers. 5 samples from each of the following categories of text were subjectively chosen from the EHR data to test the tokenizers:

- ‘**Summary-style**’ which are text passages in a summary-style format that contain bleeding-occurrences
- ‘**Abbreviations**’ which is text passages that include abbreviations
- ‘**Sentences**’ which are long sentence text passages that include bleeding occurrences.

Summary-style texts were included to test the performance on text passages that does not contain normal sentences. *Abbreviations* were included because it was suspected that tokenizers could make the error of splitting sentences on abbreviations that include dots. *Sentences* were included because the tokenizer needs to split the sentences similar to how the annotators have done it manually with their tags.

Table 8.1 shows two examples from the *Abbreviations* category of the test because it differentiated the tokenizers the most. The test can be seen in its entirety in Appendix H. The three columns show the original text and how the Stanza and NLTK tokenizers have split the sentences. In the tokenized text, a ‘||’ tag has been placed where the sentence split was made by the tokenizer. Red marks a sentence split where the tokenizer has split the sentence wrongly on an abbreviation.

Original	Stanza	NLTK
Abbreviations (split marked in red if wrongly split on abbreviation)		
Der er faldende car-bamid til 11,1, stigende Hb til 6,9. cont. PPI-infusion. rp. biokemi inkl. Hb til aften. Der er svar på Hp-test, som er negativ. Pt. får ASA, som kan være årsag til ulcus. Vi genoptager. rp. Fragmin 12.500 til aften x 1 dagligt.	Der er faldende car-bamid til 11,1, stigende Hb til 6,9. cont. PPI-infusion. rp. biokemi inkl. Hb til aften. Der er svar på Hp-test, som er negativ. Pt. får ASA, som kan være årsag til ulcus. Vi genoptager. rp. Fragmin 12.500 til aften x 1 dagligt.	Der er faldende car-bamid til 11,1, stigende Hb til 6,9. cont. PPI-infusion. rp. biokemi inkl. Hb til aften. Der er svar på Hp-test, som er negativ. Pt. får ASA, som kan være årsag til ulcus. Vi genoptager. rp. Fragmin 12.500 til aften x 1 dagligt.
Forb. mangler på h�. coll. fem. cikatrice til morgen.. Uvist om pt. selv har f�et pillet det af.. Hud...: Udf�rte handlinger. Pga. vandladning i sengen er der gjort grundig, steril afvaskning..	Forb. mangler p� h�. coll. fem. cikatrice til morgen.. Uvist om pt. selv har f�et pillet det af.. Hud...: Udf�rte handlinger. Pga. vandladning i sengen er der gjort grundig, steril afvaskning..	Forb. mangler p� h�. coll. fem. cikatrice til morgen.. Uvist om pt. selv har f�et pillet det af.. Hud...: Udf�rte handlinger. Pga. vandladning i sengen er der gjort grundig, steril afvaskning..

Table 8.1: Sentence tokenization test between Stanza and NLTK. ||: Split. Red: Wrong split

The test showed that the Stanza tokenizer performs better than the NLTK tokenizer in this small comparison. While the Stanza tokenizer performs slightly better on summary-style text and sentences as can be seen in Appendix H, it is much better at splitting sentences at abbreviations as can be seen from the examples in Table 8.1. The NLTK tokenizer e.g. splits “Forb. mangler på hø. coll. fem. cikatrice til morgen..” into ‘*Forb. // mangler på hø. // coll. // fem. // cikatrice til morgen..*’ while Stanza keeps it as one whole sentence. For this reason, the Stanza tokenizer was chosen for the sentence tokenization.

8.2.3 Comparison of the sample methods

The advantage of the paragraph based method is that samples include the context of the free text in the EHR, e.g. a heading. This can potentially give the neural network additional information that it can use to make correct classifications. Moreover, the samples can consist of several sentences, which improves the context even further. On the other hand, the additional context increases the length of the samples. This increases the computing time and makes additional demands to the network to be able to ignore irrelevant tokens. The last, and most important advantage is that this method does not use a pre-trained sentence splitter, which potentially can perform wrong sentence splits. Instead, the text is split into samples based on where the doctors have made a new line or a new subheading when writing the EHR note.

The advantage of the sentence based samples is that the network will be presented short individual sentences one at a time. This decreases the demand on the network to be able to ignore irrelevant tokens. This is also an advantage in term of explainability as the model then will pinpoint the exact sentence indicating bleeding. However, individual sentences can also be a disadvantage for some of the text passages in the EHR if a bleeding occurrence stretches over e.g. two sentences and cannot be classified by only looking at one. In this case, the network will receive and classify the sentences independently, and this might result in wrong classifications. Moreover, this method makes high demands to the sentence splitter to split sentences correctly. The small test in section 8.2.2.1, though, shows promising results with the Stanza tokenizer.

To compare the advantages and disadvantages of the two sample generation methods in action, a test was executed. This is described in section 9.3.

8.3 Extraction of samples

This section describes how the annotated samples were extracted for the sentence and paragraph sample methods. In addition to extracting the annotated positive and misinterpretable negative samples, random negative samples are also extracted from the EHRs so that they together can represent the whole dataset. This will be further explained in 8.6.1.

8.3.1 Extraction of annotated samples

Sentence dataset

As the doctors were asked to annotate whole sentences as samples, these annotations work as a manual sentence tokenization of the positive and misinterpretable negative sentences. The samples are simply located from the annotated EHRs using the annotation tags, e.g. extracting everything in between `##start_bleeding##` and `##stop_bleeding##` for positive bleeding events.

The code can be seen in Appendix I.

Paragraph dataset

For this dataset, the annotated sentences could not be extracted using only the tags of the annotators as one paragraph can contain both positive and misinterpretable annotated sentences. Therefore, all EHRs were first split into paragraphs after which they were given either a positive, misinterpretable or no label. Paragraphs were labeled as positive if the paragraph contained a positive start and stop tag, even if the paragraph also contained a misinterpretable negative tag. Paragraphs which included the misinterpretable negative start and stop tag, and no positive tag, were labelled as misinterpretable negatives. The code can be seen in Appendix J.

8.3.2 Extraction of random samples

In addition to the labeled samples, random negative samples from all EHRs were extracted, excluding notes with already positively annotated sentences or paragraphs. This was done to ensure that the model would be presented with different kinds of sentences or paragraphs, and not just those that are related to bleeding events.

An algorithm randomly selected unstructured texts of the EHR notes that did not contain any annotated samples. Each of the unstructured texts were tokenized using the respective tokenization method for that dataset, either sentence or paragraph based, and samples were then randomly selected. The code is attached in Appendix I and Appendix J.

8.4 Data de-identification

The data collected for this thesis comprises EHRs which contain personal information. According to Danish law, any form of data processing of other persons' personal information, which is not done in private context, is protected under The General Data Protection Regulation.

The data collected for this thesis is supposed to be de-identified beforehand. De-identification is still an active field of research though [80], and therefore it was chosen to make a program that goes through the EHRs and deletes names to be absolutely sure that the EHRs were de-identified in that respect.

This section describes the cleaning of the data before the de-identification and the following de-identification of the data.

8.4.1 Pre-de-identification cleaning

The FLAIR named-entity recognition model [81], which is used to search for names in the text, considers words as a group of characters separated by whitespaces. This means that e.g. a comma after a word will be taken as a part of that word and that a missing space between words and a parenthesis would be interpreted as one word. This can be seen in the example sentence *'Patient has had a bleeding(John is informed)'* where *'bleeding(John'* is considered as one word by the named entity recognizer.

For this reason, a regular expression algorithm was run through the sentences to make spaces between some special characters and words. Dots and dashes were not part of these special characters as that would change the structure of many abbreviations.

8.4.2 De-identification method

The de-identification was done using a pre-trained named-entity recognition model called FLAIR [81]. The model is developed by Zalando research. The pre-trained model is trained on the Danish Dependency Treebank dataset

which was further annotated with named entities for name, position and organization by The Alexandra Institute [82]. This model was chosen as it has the best performance [83].

An algorithm was developed to perform named entity recognition on sentences to identify names and replace them with the tag ‘NAME’. See Appendix I and Appendix J for the algorithms which include the de-identification code.

The named-entity recognition model does not work flawlessly. Problems with recognizing e.g. drug or medical device names as persons have been observed. Two examples are shown below in Table 8.2:

Before de-identification	After de-identification
I 2014 gastropæd. amb. set med blod i afføringen, man konkluderede obstipation og blev behandlet med Movicol igennem et halvt år	I 2014 gastropæd. amb. set med blod i afføringen , man konkluderede obstipation og blev behandlet med NAME igennem et halvt år
Der kommer mørk rød (rødvinsfarvet) urin i Kath posen.	Der kommer mørk rød (rødvinsfarvet) urin i NAME posen.

Table 8.2: Example of text before and after deidentification

This is not considered a major problem as removing a single word from sentences will most likely not have an effect on the classification of a bleeding occurrence. Moreover, in a real-life setting, this would not be a problem, because the journals would not have to be de-identified.

8.5 Sample cleaning

After the extraction and de-identification, all samples of each dataset were contained in an Excel sheet with their corresponding label: Positive, misinterpretable negative, or random negative. The final sample cleaning consisted in:

- Removal of annotation tags which were left in the text due to misplacement.
- Removal of empty samples due to start and end annotation tags being placed right next to each other.
- Elimination of extra spaces and spaces starting and ending the string
- Elimination of duplicates.

Table 8.3 shows the number of samples per label after sample cleaning for the paragraph dataset.

Paragraph samples	
	Count
Positive	5,151
Misinterpretable negative	5,133
Random negative	6,976
Total	17,260

Table 8.3: Number of samples for the paragraph dataset

Table 8.4 shows the number of samples per label after sample cleaning for the sentence dataset.

Sentence samples	
	Count
Positive	5,893
Misinterpretable negative	5,630
Random negative	8,278
Total	19,801

Table 8.4: Number of samples for the sentence dataset

8.6 Distribution of datasets

This subsection first presents the training objective and then, following from the training objective, the distribution of the sentence and paragraph datasets.

8.6.1 Training objective

The overall objective of the to-be-trained model is to detect the text passages in the free text of the EHR that contain bleeding occurrences. At this point, two ways of dividing the free text into samples have been proposed - sentences and paragraphs.

There is an uneven distribution of the positive samples (bleeding occurrences) and negative samples (everything else) in EHRs, there being many more of the latter. If the model were trained on the data as it is, it would be

an imbalanced classification, which could lead to the model predicting all samples as negative. In general terms, the model will have no incentive to learn the difference between negative and positive samples if it can obtain e.g. 99 % accuracy from simply predicting all samples as negative. But as the model should be evaluated on the performance on the few positive text passages of the EHR, the training conditions must be different to those of the real-life EHR that the finished model will work on.

For the described reason, the training objective is made to be a balanced classification, where the model sees as many positive as negative samples. This makes sure that the model will be trained to perform equally well on positive and negative samples, thus being able to detect bleeding occurrences in EHRs better. It is additionally made sure that half of the negative samples that the model sees resemble positive samples. In this way, the model will be able to distinguish those cases better. Lastly, it is important that the model does not only see samples that are either bleeding occurrences or negatives that resemble bleeding occurrences as it must be able to classify all sentences in an EHR note, also those not related to bleeding occurrences. For this reason, the other half of the negative samples are randomly selected samples from the free text.

8.6.2 Distribution

The datasets are each composed of 80 % training data, 10 % validation data, and 10 % test data. The balance within each of the groups is 50 % positive samples, and 50 % negative samples. The negative samples are further divided in 50 % misinterpretable negative samples and 50 % random negative samples.

Table 8.5 shows the distribution of the training, validation, and test set for the paragraph dataset.

Paragraph dataset				
	Training	Validation	Test	Total
Positive	4,121	515	515	5,151
Negative	4,121	515	515	5,151
Total	8,242	1,030	1,030	10,302

Table 8.5: Train, validation, and test distribution of paragraph dataset

Table 8.6 shows the distribution of the training, validation, and test set for the sentence dataset.

Sentence dataset				
	Training	Validation	Test	Total
Positive	4715	589	589	5,893
Negative	4715	589	589	5,893
Total	9,430	1,178	1,178	11,786

Table 8.6: Train, validation, and test distribution of sentence dataset

The two datasets will be tested in section 9.3.

8.7 Chapter summary

This chapter described how data was annotated into three categories; random negatives, misinterpretable negatives, and positives. The annotated data was used to create two different datasets based on two different ways of creating samples; paragraphs and sentences. The paragraph-based samples were created using a paragraph-tokenizer which split the text into samples based on the structure of an EHR using HTML tags. The sentence-based samples were created using a pre-trained sentence-tokenizer which split the text into sentences. Both datasets were anonymized before being split into a train, validation and test set using a 80-10-10 distribution.

Chapter 9

9 Test of word embeddings and datasets

This chapter compares the performance of the two different pre-trained word embeddings (fastText and GloVe) and the two datasets (paragraph and sentence). The performance comparisons are used to choose which embeddings and datasets to be used for further architecture and hyperparameter tuning.

9.1 Preprocessing

Before executing the test described below, the datasets were preprocessed. The following characters were removed and replaced with a space:

£ \$ @ [] * ^ _ ~ / ; & , - () # ! ½ ' ° ” %

Moreover ‘..’ were replaced by ‘.’ and ‘:’ was removed from the sentence dataset. ‘.’ was not removed from the paragraph dataset as it is part of the sample splitting method described in section 8.2.1. The reason for removing and replacing these characters are described in more detail in section 10.3.

9.2 Test of word embeddings

This subsection presents how the two different pre-trained word embeddings, fastText and GloVe, were tested. The test was performed only using the sentence dataset because it was reasoned that the type of dataset would not affect the relative difference in performance of the two embedding types. Each of the embedding types were tested on a single layer simple recurrent neural network (RNN). The output vector of the RNN was connected to a dropout layer followed by a dense layer of size two which made the prediction using the softmax function.

Both networks were evaluated using the following hyperparameters: Learning rate [1e-3, 1e-4, 1e-5], RNN hidden layer size [32, 64, 128], dropout [0.0, 0.3] and the Adam optimizer. As written in section 6.2.1.2 fastText embeddings have 300 dimensions while GloVe has 100 dimensions. To make sure that the difference in embedding dimensions does not affect the comparison

of the two, fastText embeddings were trained with both their original 300-dimensional embeddings and with a downsized version of 100 dimensions. The downsizing was done using a built-in function of the fastText library. The 100-dimensional fastText embeddings were tested on the same architecture and with the same hyperparameters as described earlier. In this way, the only thing separating the networks are the input embeddings to the RNN. Table 9.1 summarizes the best result on the validation set for each of the models. All results can be seen in Appendix R.

	Validation loss	Validation accuracy
GloVe	0.364	85.6 %
fastText (300-d)	0.408	81.4 %
fastText (100-d)	0.485	78.0 %

Table 9.1: Result of embedding test

The reported loss is an average of the losses of the epoch with the lowest loss and the prior two epochs to smooth noise. E.g., if a model achieves the lowest loss in epoch 50, the average of epoch 48, 49 and 50 is taken. The reported accuracy is the average accuracy of the corresponding epochs used for the average loss calculation.

From the table, it is seen that the accuracy of the GloVe embeddings is 4.2 percentage points higher than the best performing fastText embedding of 300 dimensions, and the loss is 0.044 lower. Based on this test, the GloVe embeddings are therefore the best choice. Table 9.2 compares the embedding types further.

	Embed- ding size	Training corpus	Domain	Training method	OOV words
fastText	300/100	Common Crawl and Wikipedia	General	CBOW	0
GloVe	100	323.122 EHRs	Medical	GloVe	3058

Table 9.2: Comparision of fastText and GloVe word embeddings

It is noticed that the fastText embeddings are trained on a bigger corpus, Wikipedia and Common Crawl, which is an advantage over the GloVe embeddings which are trained on less data. On the other hand, the GloVe embeddings are trained on an in-domain corpus, EHR data, whereas fastText is trained on a general domain.

The biggest advantage of fastText embeddings over GloVe is that there are no OOV words whereas GloVe has 3058. To decrease the number of OOV words for GloVe, a simple modification is introduced: The GloVe embeddings are trained using both upper- and lower-case words. It is therefore possible that the GloVe embedding corpus contains the word ‘bleeding’ (lower case) but not ‘Bleeding’ (upper case). To decrease the amount of OOV words, upper-cased OOV words are lower cased and lower-case OOV words are upper cased to try to find a match. Using this technique, the number of OOV words is reduced from 3058 to 362. The disadvantage of this approach is that a lower-cased word can be perceived as an upper-cased word or vice versa by the model, but it was considered a bigger disadvantage not to have a word embedding for the word. After inspection of the remaining 362 OOV words it was noticed that almost all of them are numbers e.g. dates and times such as ‘17:10:00’.

As GloVe performs better than fastText according to the experiment in Table 9.1 and, after the modification, only contains few OOV words, it is chosen to move on with these word embeddings.

9.3 Test of datasets

As described in section 8.2, two different datasets were constructed. In this section the performance of each of them will be compared. The datasets are compared using the GloVe word embeddings on both a gated recurrent unit (GRU) neural network and a convolutional neural network (CNN). It was chosen to test the datasets on a GRU as the average length of the samples in the paragraph dataset are longer than in the sentence dataset, as seen in Table 9.3.

	Average token count	Median token count
Sentence dataset	13	11
Paragraph dataset	31	24

Table 9.3: Token count statistics of sentence and paragraph datasets

Because of the longer samples, the paragraph dataset will be more vulnerable to the vanishing gradient problem. For this reason, a GRU is used rather than a simple RNN, for the reasons described in section 6.3.3. Using the GRU, it is also more likely that the paragraph dataset will be able to utilize its extra context from the longer samples to make a classification, and the sentence dataset will not suffer from this architecture as it is also suitable for shorter sentences.

The GRU neural network consisted of a single GRU unit. The output vector of the GRU was connected to a dropout layer followed by a dense layer of size two which made the prediction using the softmax function. The following hyperparameters were tested: Learning rate [5e-4, 1e-4, 5e-5], GRU hidden layer size [32, 64, 128], dropout [0.0, 0.3] and the Adam optimizer.

The CNN consisted of a single convolution layer followed by a global max pooling layer. The output from the global max pooling layer was connected to a dropout layer followed by a dense layer of size two which made the prediction using the softmax function. The following hyperparameters were tested: Learning rate [5e-4, 1e-4, 5e-5], filter size [3, 6, 9], number of filters [64, 128, 256], dropout [0.0, 0.3] and the Adam optimizer.

The result of the test is seen in Table 9.4, with the validation accuracy and validation loss calculated in the same way as for Table 9.1.

	GRU		CNN	
	Val. accuracy	Val. loss	Val. accuracy	Val. loss
Paragraph	86.4 %	0.327	86.5 %	0.322
Sentence	88.2 %	0.290	89.5 %	0.267

Table 9.4: Result of dataset test

Table 9.4 shows that the sentence dataset performs better in terms of both loss and accuracy on both architectures. Therefore, the sentence dataset is considered superior and will be analyzed and tested further in the following

sections. Moreover, as described in Chapter 5 (Related Work) the study most similar to this by Runmeng et al. also used sentence-based classification and achieved good results.

9.4 Chapter summary

This chapter found that the Global Vectors (GloVe) embeddings perform better than fastText based on the loss and accuracy of the validation set. The GloVe embeddings achieved a loss of 0.364 and an accuracy of 85.6 % while the best performing fastText model resulted in a loss of 0.408 and an accuracy of 81.4 %. Afterwards the sentence and paragraph datasets were compared using the GloVe embeddings. The sentence dataset achieved the best loss and accuracy on the validation set using both a CNN and GRU architecture. Therefore, the following chapters will use the GloVe embeddings and the sentence dataset for evaluation.

Chapter 10

10 Exploratory data analysis and preprocessing

This chapter describes the sentence dataset that was shown to work best in section 9.3, and the challenges that follow when classifying text in the medical domain. The analysis of the data is necessary in order to preprocess the data in the best possible manner before feeding it to a neural network. The following analysis is based solely on the training set.

10.1 Quantitative analysis of data

The purpose of an exploratory data analysis is to summarize the characteristics of the dataset. Two main characteristics of text data are the sample length and word frequencies.

Figure 10.1 shows the frequency of sentence lengths of the dataset per class.

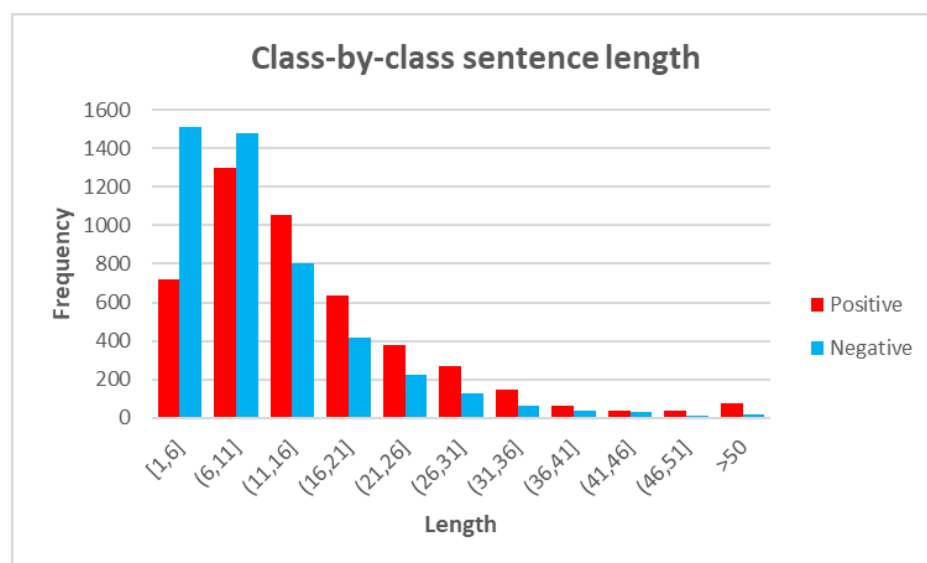


Figure 10.1: Sentence length of the dataset

Moreover, Table 10.1 shows the average and median token count for the combined and per class samples of the dataset.

	Average token count	Median token count
Positive+Negative	13.7	11
Positive	15.9	13
Negative	11.5	9

Table 10.1: Token count statistics for sentence dataset

It is seen that the majority of the samples are between 1 and 21 tokens long with an average of 13.7 and a median of 11. Table 10.1 shows that positive sentences on average have more tokens than the negative sentences. Figure 10.1 shows that this is mainly due to an overrepresentation of negative sentences in the 1-6 tokens interval. This unequal distribution is most likely caused by the method used to sample negative data. The dataset contains many random short sentences, headlines, or statements, while there will be fewer positive sentences of that length.

Table 10.2 shows the top-10 word frequencies of the whole dataset and per class when stop words have been ignored. Stop words are commonly used words such as ‘*og*’, ‘*i*’, ‘*den*’, and ‘*er*’ which occur so often that they do not add any informative meaning in the text classification task. See Appendix K for the full stop-word list. Words are not lowercased because words which start with a capital letter often are start-of-sentence words, and these can potentially be good discriminators of the two classes. Moreover, the GloVe embeddings shown to perform best in section 9.2 distinguish uppercase and lowercase letters.

	All samples	Positive samples	Negative samples
1.	<i>blødning</i> (1382)	<i>blødning</i> (964)	<i>ikke</i> (507)
2.	<i>ikke</i> (877)	<i>ses</i> (506)	<i>blødning</i> (418)
3.	<i>blod</i> (716)	<i>blod</i> (463)	<i>ingen</i> (316)
4.	<i>ses</i> - 615	<i>ikke</i> (370)	<i>Ingen</i> (302)
5.	<i>Pt.</i> (537)	<i>Pt.</i> (342)	<i>blod</i> (253)
6.	<i>ved</i> (509)	<i>hæmaturi</i> (276)	<i>ved</i> (234)
7.	<i>ingen</i> (455)	<i>ved</i> (275)	<i>uden</i> (216)
8.	<i>samt</i> (399)	<i>samt</i> (266)	<i>Pt.</i> (202)
9.	<i>pt.</i> (374)	<i>grundet</i> (262)	<i>Abdomen</i> (195)
10.	<i>Ingen</i> (348)	<i>haft</i> (249)	<i>pt.</i> (161)

Table 10.2: Frequent words in sentence training dataset. (Frequency).

The most common word is ‘*blødning*’. In the positive samples, ‘*blødning*’ is the most common token and occurs 964 times, and in the negative class ‘*blødning*’ occurs 418 times. Making a naive assumption that ‘*blødning*’ only occurs once in every sentence, ‘*blødning*’ would be able to classify 1382 of the 9430 training samples with an accuracy of $\frac{964}{1382} \cdot 100 \approx 69.8$ %. A good discriminator is the word negation ‘*Ingen*’ which occurs 302 times in the negative samples making it possible to classify 348 of the 9430 training samples with an accuracy of $\frac{302}{348} \cdot 100 \approx 86.8$ %. This relatively high percentage though only applies to $\frac{348}{9430} \cdot 100 \approx 3.7$ % of the data in the training set.

The observations described above will be further investigated in section 11.1 where it will be used to make a rule-based classifier.

10.2 Qualitative analysis

The quantitative analysis provides an overall understanding of the dataset but does not analyze the data in depth. This is best done by looking at individual samples of the data to get a sense of e.g. special patterns, special characters, syntax, and analyzing if it would be beneficial to e.g. remove special characters.

For this analysis, samples from the training set were manually inspected. Table 10.3 highlights some examples of patterns and challenges that were found in the samples and explains why the examples are interesting in regard to the classification problem.

Sentence	Explanation
Negative samples	
Gastrointestinal blødning? : Nej	Although the text says ‘ <i>gastrointestinal blødning</i> ’ it does not mean that the sentence is positive because the sentence is followed by a question mark and a negation.
Abdomen. Blødt og uømt	The word <i>Blødt</i> is ambiguous
Pt. er informeret om risici i form af dura- og nerverodslæsion , påvirkning af de sakrale nerverødder , blødning og infektion	The word <i>informeret</i> is an atypical negation for bleeding. Moreover, <i>bleeding</i> is written several tokens after the actual negation
Positive samples	
Aspireret 50ml kaffegrumslygn ved ankomst	<i>kaffegrumslygn</i> indicates bleeding.
Der laves behandlingsforsøg med : . rp. Cyclocapron 500 mg x 2 i 1 uge.	The drug <i>Cyclocapron</i> , which is used to stop bleedings, indicates that there is a bleeding.
Urinstix : . Glukose : neg. Ketoner : neg. Blod : 3+. PH : 8.5. Protein : 2+. Nitrit : neg. Leucocyttter : neg.	‘ <i>Blod 3+</i> ’ indicates blood in the urine
(...) det kan meget vel være denne , der er blødningsårsagen.	<i>blødning</i> mistakenly spelled <i>blødnign</i> . Moreover, the bleeding indicating word <i>blødningnsårsagen</i> is a compound word consisting of two nouns <i>blødnign</i> and <i>årsagen</i> .

Table 10.3: Examples of sentence structures in the sentence dataset

Regarding the importance of special characters, the analysis revealed that question marks are important, e.g. in the phrase ‘*Gastrointestinal blødning? : Nej*’, the question mark clarify that the two first tokens ‘*Gastrointestinal*

blødning’ is not a statement but a question which is answered afterwards. Moreover, numbers are also important as they can indicate a bleeding e.g. ‘Urinstix: (...) blod: 3+’, where ‘3+’ indicates blood in the urine.

During the manual inspection of the samples, several single words that indicate a bleeding occurrence were found. Table 10.4 shows examples of these, but the list is not complete.

Words indicating bleeding		
<i>Hæmoragi</i>	<i>hæmaturi</i>	<i>hævelse</i>
<i>vaginalblødning</i>	<i>kaffegrums</i>	<i>sivblødning</i>
<i>BLØDERKALD</i>	<i>corpushæm.</i>	<i>petekkie</i>
<i>hæmothorax</i>	<i>Blodtab</i>	<i>Hæmoptyse</i>
<i>epistaxis-patient</i>	<i>ulcerationer</i>	<i>Koagel</i>
<i>corpusblødning</i>	<i>gastropati</i>	<i>SAG-M</i>
<i>blod</i>	<i>hæmatom</i>	<i>bløderpt.</i>
<i>blødning</i>	<i>Epistaxis</i>	<i>blødt</i>

Table 10.4: Examples of bleeding indicating words

It is worth noticing that the words can be written in all its forms, e.g. present, past tense, past perfect, etc., and words can be misspelled. Moreover, the words can be compound with another noun, e.g. *corpusblødning*. A word like *blødt* is ambiguous as it both has the meaning of a bleeding, but it can also mean the opposite of hard. This is some of the many challenges that is met when classifying medical text.

10.3 Data preprocessing

Based on the inspection of the data, the following characters are removed from the samples before feeding it to the neural networks:

£ \$ @ [] * ^ _ ~ / : ; & , - () # ! ½ ’ ° ” % ..

The reason for removing these characters is that the text is simplified. E.g. removing the exclamation mark from the samples modifies a sentence like ‘*he*

has a bleeding!’ to *he has a bleeding*’. This change does not influence the classification of the sentence because the preprocessed sentence still indicates a bleeding. On the other hand, removing special characters reduces the number of words in the vocabulary and it reduces the number of input tokens to e.g. a recurrent neural network (RNN) model. This potentially helps avoiding the vanishing gradient problem, and it helps the network focus on the important parts of the samples.

The special character ‘?’ has been kept as it can influence a sample like *blødning ? nej*’. Here, the question mark indicates a question and that the word following *blødning* is the answer to the question.

As the GloVe embeddings only have 362 out of vocabulary (OOV) words on the training data and that they were all deemed insignificant by inspection, it is argued that further preprocessing like lower casing, stemming, lemmatization, or compounding is not necessary. The pre-trained GloVe embeddings also include abbreviations of many forms.

10.4 Chapter summary

This chapter presented the exploratory data analysis and the pre-processing following from that analysis.

The quantitative analysis revealed that the majority of sentences are between 1-21 tokens at an average of 14 and median of 11. From the top-10 word frequencies it could be seen that *blødning* is in top-2 for both classes.

The qualitative analysis highlighted some of the interesting text patterns distinguishing the two classes, including different indicator words for bleeding.

Based on the data analysis, the pre-processing was then decided on. It mainly consists in removing special characters.

Chapter 11

11 Training

This chapter first presents a rule-based classification baseline for the deep learning models. It then describes the training strategy after which it presents the iterations of choosing and training a convolutional neural network (CNN) architecture on the classification problem. A recurrent neural network architecture was then trained, and the description of the iterations of that training process can be seen in Appendix L. Then, a combination of the two architectures is tested, after which the validation results of all three architectures are presented, and the best model is chosen for further exploration with two different augmentation methods. Lastly, a model ensemble is created for the final model.

11.1 Rule-based classification baseline

A simple rule-based classification was used to create a baseline for the deep learning models to be examined in the further sections of the thesis. The rule-based approach was inspired by the ones used by Taggart et al. [26] to detect bleeding and Tian et al. [28] to classify DVT and PE.

The rule-based approach was based on the findings of the data exploration in Chapter 10 and consisted in finding bleeding-positive words such as *blødning* or *hæmaturi* and their possible negative modifiers such as *ikke* and *muligvis*.

As a rule-based classification is not the main goal of the thesis, but just a baseline measure, the approach was based mainly on unigrams. The rule-based classification algorithm searches each sample for a positive word. If no positive words are found, the sample is classified as negative. If a positive word is found, the algorithm searches its context words in a window of size 4, which was found to work best, to look for negative modifiers. If a word from the positive list is not accompanied by a negative modifier, the sample is classified as positive. If all positive words are accompanied by negative modifiers, the sample is classified as negative.

The lists of positive words and negative modifiers were based on the findings of the data exploration which was based solely on the training set. The lists were then modified stepwise during training in order to improve the accuracy on the validation set.

This approach yielded an accuracy of 80.65 %. The algorithm with the positive indicators and negative modifiers can be seen in Appendix M.

11.2 Training strategy for neural networks

As almost all hyperparameters of a neural network interact, i.e. changes their optimal setting when other hyperparameters are changed, it is difficult to train a neural network. To aid training in a structured fashion and in affordable time, it is divided in three major steps. For the purpose of these steps, the **architecture hyperparameters** are defined as the ones defining e.g. the layer types, number of layers, layer sizes, and filter sizes. The **setting hyperparameters** are defined as the ones that are applied to the layers and adjust the training, e.g. dropout, regularization, batch size, and if embeddings are trainable. The training steps are:

1. Find the best architecture hyperparameters for the model. The training is done across intervals of several setting hyperparameters that might interact with the architecture hyperparameters, e.g. dropout might interact with layer size.
2. Find the best optimizer and learning rate for the architecture hyperparameters.
3. Tune the setting hyperparameters to obtain the best model.

The validation loss will be used as metric and reported when making the decisions on the best hyperparameters in the following subsections. The validation accuracy will also be reported. The loss is used as metric because the model is trained on loss and it therefore expresses the raw performance of the model. The loss reflects the decision margin on the predictions, i.e. it is a measure of the certainty of all predictions as it is a function of the probability of the true class. On the other hand, accuracy only considers the percentage of correct predictions. For this thesis, improving the decision margin is

preferred over increasing the accuracy (although they typically improve together). This is because, when used in a clinical setting, the doctor can check the decision margin to know which predictions can be trusted and which should be double checked. With a lower overall loss, more samples will have a high decision margin, resulting in fewer uncertain predictions.

When training, diagrams are used to support that an optimal hyperparameter has been reached. The relevant diagrams are shown in the following sections.

11.3 Convolutional model

This subsection presents the choosing and training of a CNN model. Figure 11.1 visualizes the structure of a general convolutional architecture and the hyperparameters that can be tuned to improve the model. Code for training the CNN can be seen in Appendix N.

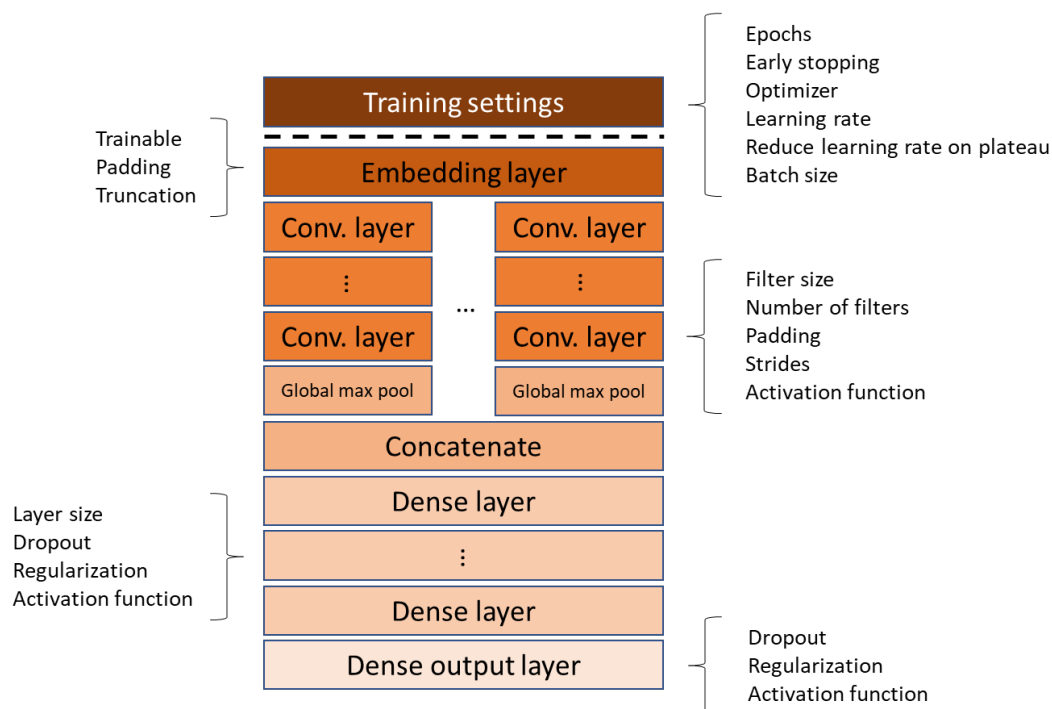


Figure 11.1: General structure of a CNN and the hyperparameters associated

The training settings are not technically part of the network architecture but is visualized to show the hyperparameters related to training. This includes how many epochs the model is trained, which optimizer is used, the

learning rate, and the batch size. During training, the maximum number of epochs was set to 200 and early stopping was used to stop the training after 15 epochs with no improvement. The learning rate was reduced with a factor 0.2 after 5 epochs with no improvement. In the further sections, the reported losses and accuracies are the average of the 2 epochs prior to and the 1 epoch that got the lowest loss during training. This is done to smooth noise.

The embedding layer is the input to the model and the parameters are initialized with the pre-trained GloVe embeddings. These can either be constant during training or jointly be trained with the rest of the parameters of the model. To handle variable input lengths, the input is padded with zeroes to the right of the sentence if smaller than 100 words and truncated to the right if larger.

Figure 11.1 shows that one or more convolutional layers (conv-layer) can work on the output of the embedding layer. This creates ‘parallel’ processes which is depicted as two stacks, or series, of convolutional layers. In the following training, a single filter size means that the model has a single series of conv-layer(s). If a filter size is e.g. [2, 3], it means that the network has one series of filter size 2 and one series of filter size 3.

The number of conv-layers refers to how many convolutional layers are stacked in each series. Each conv-layer has a filter size and a number of filters. These numbers are the same throughout the stacked layers of the series. The first conv-layer of each series operates on the embedding layer. If multiple conv-layers are stacked, each operate on the output of the previous layer. For the training of the model, the conv-layers have a stride of 1, no padding of the input, and a ReLU activation function is applied to the output.

After the convolutional layers, each series is sent through a global max pool function and concatenated. This concatenation is sent through zero or more dense layers before the final output dense layer.

The output dense layer must have the same number of neurons as the number of classes that the model classifies when using the softmax activation function. If there are dense layers prior to the output layer, they can be of variable size and with different activation functions applied. The ReLU was used for all dense layers but the output layer and the weights were initialized

using the Glorot uniform method. Dropout and regularization can be applied to all dense layers.

The further sections first test how many dense layers should be added before the output dense layer, then the filter sizes, and how many conv-layers should be stacked and their number of filters. Next, the best optimizer and learning rate for that architecture is found. Finally, the last hyperparameters are tuned and the best recurrent architecture is chosen.

11.3.1 Architecture hyperparameters

The Adam optimizer with default settings [84] and learning rates 5.0E-4, 1.0E-4, and 5.0E-5 were used throughout the tuning of the architecture hyperparameters because it was found that it is very robust to changes in the model. Additionally, the batch size was set to 128 and embeddings to non-trainable for all training loops as they were not expected to interact with the architecture hyperparameters.

11.3.1.1 Number of dense layers

The optimal number of dense layers between the concatenation and the final dense output layer was tested in a training loop with 0, 1, and 2 dense layers of sizes 64, 128, and 256. The dense layers were trained with dropout 0 and 0.3 as it might interact with the number of dense layers. The convolutional part of the network consisted in filters of sizes 3, 5, 7, [3, 5], [3, 7], [5, 7], [3, 5, 7]. There were 32, 64, 128, 256, 512, or 1024 filters for each of the filter sizes. This was applied in 1, 2, or 3 convolutional layers.

Figure 11.2 shows the minimum loss across all hyperparameters for 0, 1, and 2 dense layers.

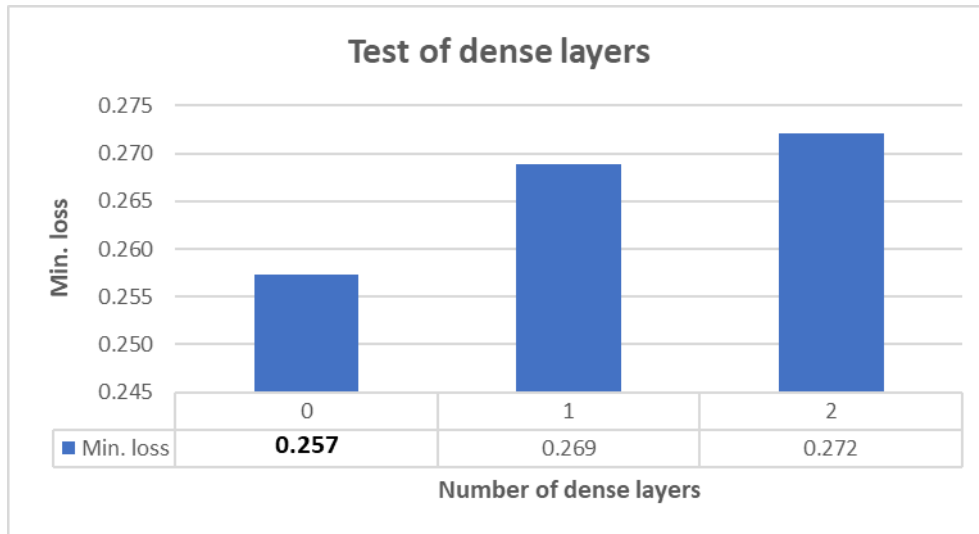


Figure 11.2: Test of number of dense layers for CNN model

As no dense layers before the final output dense layer gave the best loss, it was decided to select this architecture.

11.3.1.2 Filter sizes

To determine the optimal filter size(s) to work on the input, the following filter sizes were tested: 1, 2, 3, 4, 5, 7, [1, 2], [1, 3], [2, 3], [3, 4], [3, 5], [3, 7], [5, 7], [1, 2, 3], and [3, 5, 7]. There were variations in the number of convolutional layers (1, 2, 3), number of filters (32, 64, 128, 256, 512, 1024, 1280, 1536, 1792, 2048, 2560), and dropout (0, 0.3).

Figure 11.3 shows the minimum losses produced across all hyperparameters for the different filter sizes.

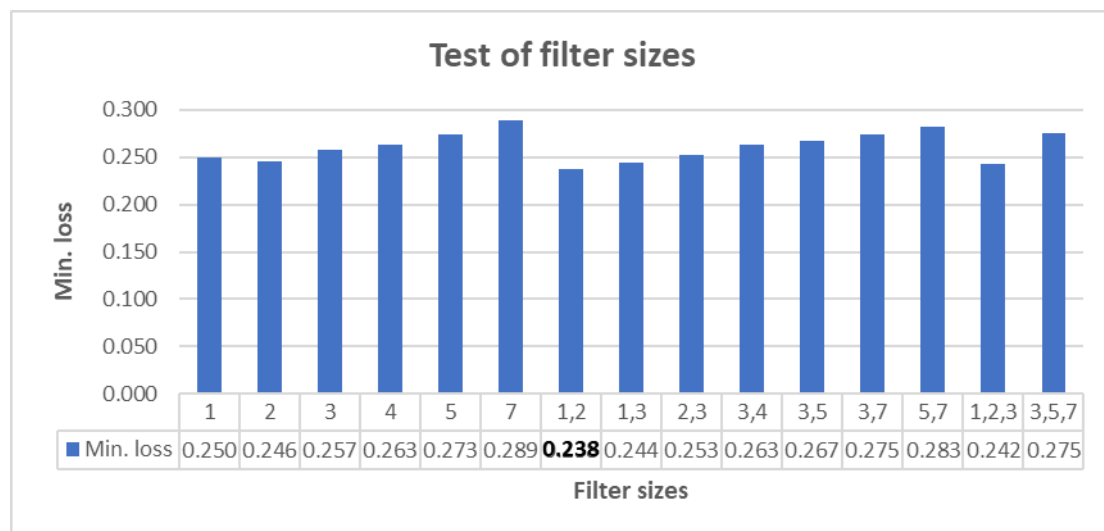


Figure 11.3: Test of filter sizes for CNN model

Using the filter sizes $[1, 2]$ on the input gave the best loss of 0.238 and is therefore used going forward. Looking at Figure 11.3, it seems that using a filter size of 1 or 2 is essential in producing a low loss. These sizes produce good results independent of their combinations with other filter sizes. When a filter size of 1 or 2 is not included, the loss seems to increase as the filter sizes become larger.

11.3.1.3 Number of convolutional layers and filters

To examine the effect of the number of conv-layers and filters, the training loop to test the filter sizes from section 11.3.1.2 was filtered to only contain the models with filter sizes $[1, 2]$.

Figure 11.4 shows the combinations of number of convolutional layers with number of filters for filter sizes $[1, 2]$ across all remaining hyperparameters.

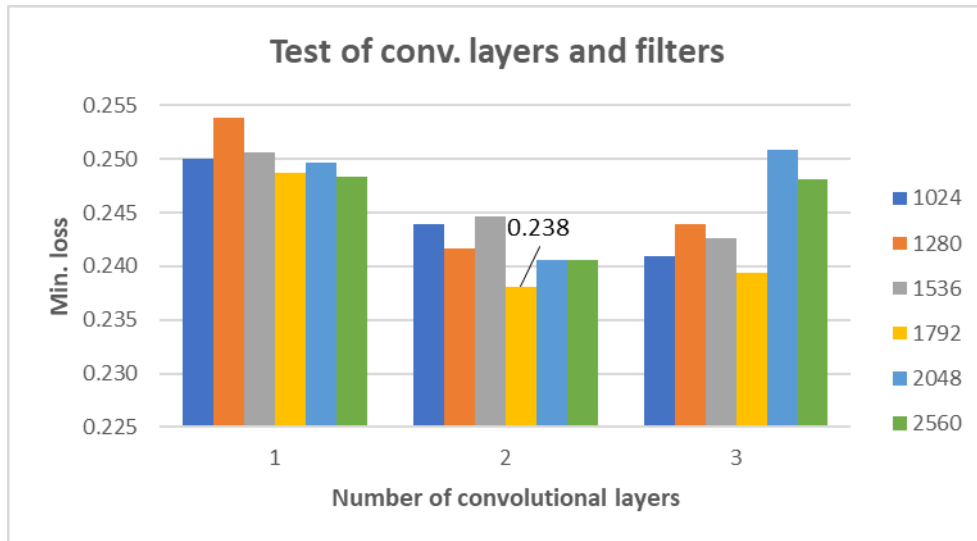


Figure 11.4: Test of convolutional layers and filters for CNN model

Figure 11.4 shows that the best result of the filter sizes $[1, 2]$ was produced with 2 convolutional layers per filter size series and 1792 filters per layer. The trend on Figure 11.4 seems to indicate that 1792 filters is a good ‘optimum’ with regards to the capacity of the layer.

11.3.2 Optimizer and learning rate

To find the best optimizer and learning rate to use with the chosen architecture, Adam with default settings and SGD with momentum of 0.9 were

tested in a range of learning rates until an ‘optimum’ was found. The architecture hyperparameters were used with dropout 0, 0.25, and 0.5 in the training loop.

Figure 11.5 shows the minimum loss produced across all hyperparameters for the Adam and SGD optimizers.

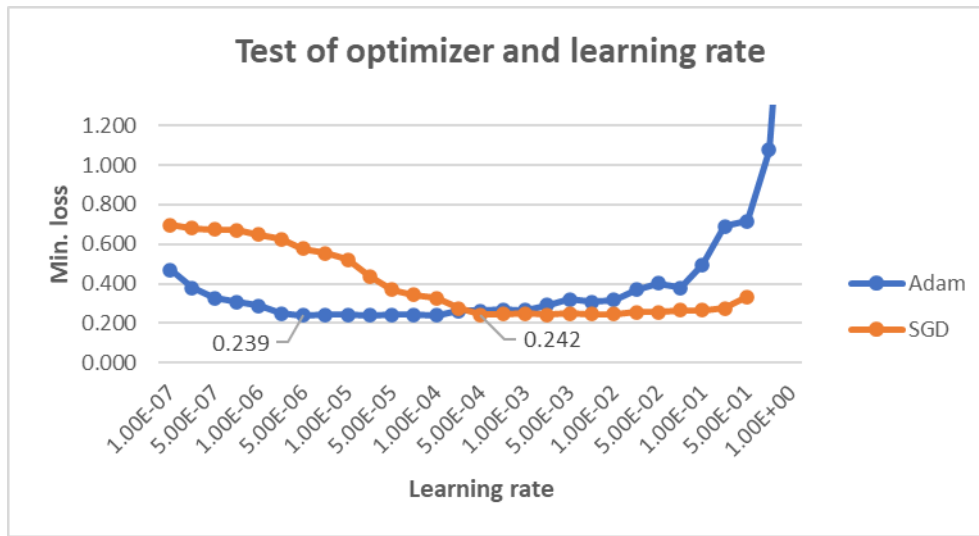


Figure 11.5: Test of optimizer and learning rate for CNN model

It was found that Adam performed best, producing a loss of 0.239, with a learning rate of 5.00E-06.

11.3.3 Setting hyperparameters

The setting hyperparameters were tested on the chosen architecture hyperparameters: 2 series with each 2 stacked convolutional layers. One series using 1792 filters of size 1 for each stacked layer, the other series using 1792 filters of size 2. Dropout between the concatenation of the two series and the output dense layer at 0.1, 0.2, 0.3, 0.4, and 0.5 was tested. Regularization of 0, 0.0001, 0.001, and 0.01 was applied to the dense layer. The embedding layer was tested as trainable and non-trainable. The model was trained with the Adam optimizer with a learning rate of 5.00E-06 and at batch sizes 8, 16, 32, 64, 128, 256, and 512.

Table 11.1 shows the final model with the chosen best architecture and setting hyperparameters.

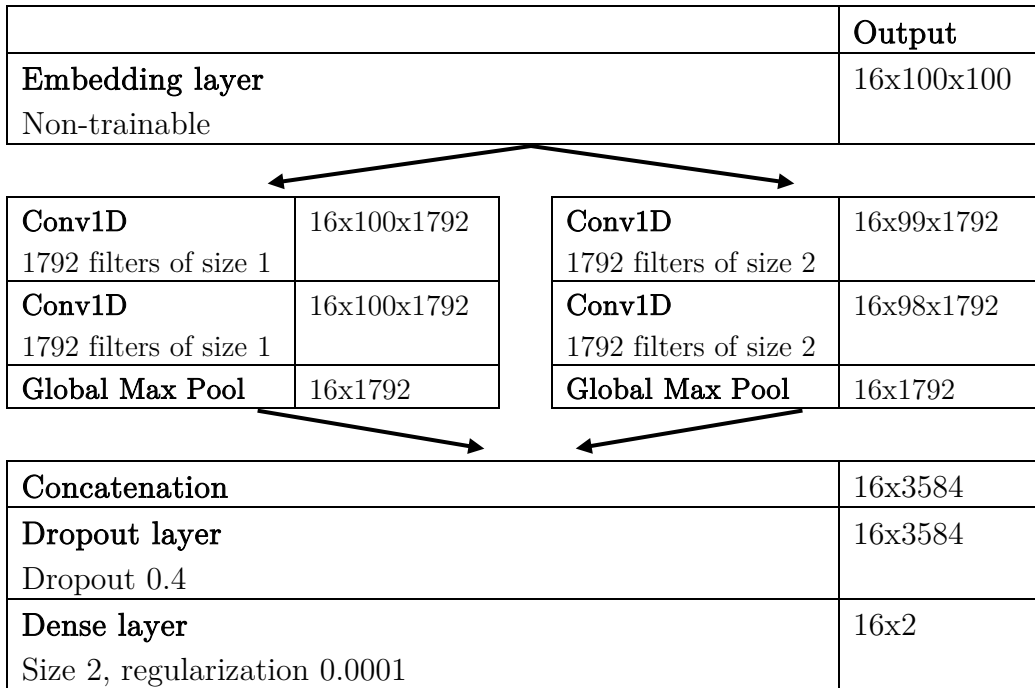


Table 11.1: Final CNN structure

The output column describes the output size of the specific layer of the model. On Table 11.1 the first dimension of all layer outputs is 16 because that was the best performing batch size. The embedding layer therefore outputs a batch of 16 sequences of 100 words, each word represented by an embedding of size 100. It is noted that the size of the second dimension drops from 100 to 99 to 98 in the convolutional series of filter size 2. This is because a filter of size 2 sliding over an input with stride 1 will output the input size minus 1.

The best model achieved a minimum loss averaged over the best and 2 prior epochs of 0.237 and a validation accuracy of 91.9 %. The training process of the best achieving model is shown in Figure 11.6.

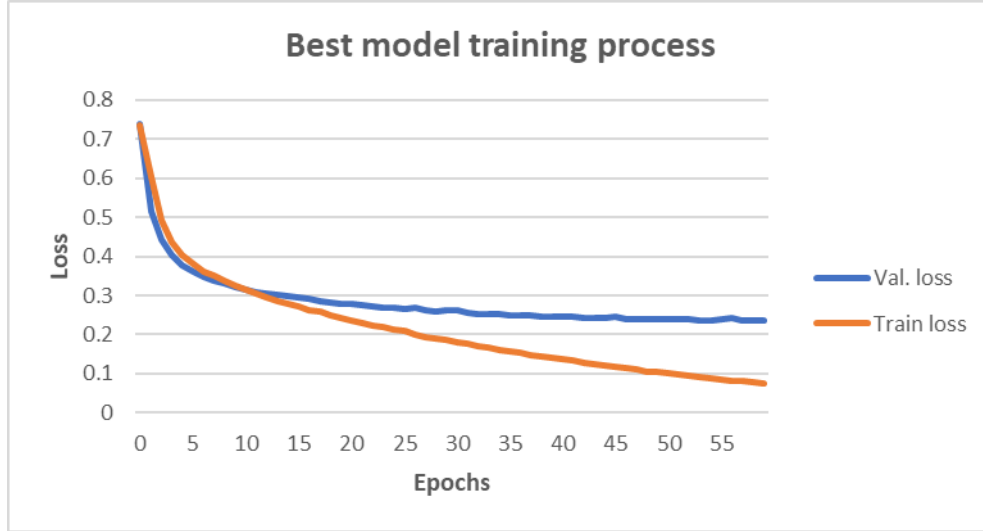


Figure 11.6: Visualization of the training process for the best CNN model

The model experiences lower validation loss than training loss the first 9 epochs which might seem counterintuitive. This is likely caused by the regularization mechanisms, dropout and L2 weight regularization, being turned on during training loss evaluation but turned off during validation loss evaluation. Additionally, the training loss reported is an average of all batch losses during one epoch while the validation loss is a one-time evaluation. In many cases, the early batches of an epoch will have higher losses than the later which will result in a higher reported training loss [85]. The training eventually reaches overfitting on the train set before the learning is stopped.

11.4 Recurrent model

The description of the different iterations of the training process of the recurrent model can be seen in Appendix L. The best model was found to be a single bidirectional Gated Recurrent Unit (GRU) layer of size 16 with trainable embeddings. The output of the bidirectional GRU is connected to the final softmax dense layer with a dropout layer of rate 0.3 in between. The model was trained with a learning rate of $7.89\text{e-}4$, a batch size of 32, and the Adam optimizer. The best model achieved a minimum loss averaged over the best and 2 prior epochs of 0.273 and a validation accuracy of 90.2 %.

11.5 Combination model

A model combining the best recurrent architecture and the best convolutional architecture was tested to see if it would benefit from both types of

features. The combination model concatenates the two architectures at the output from the BiGRU layer of the recurrent architecture and the concatenation layer of the convolutional architecture.

The model is shown on Figure 11.7.

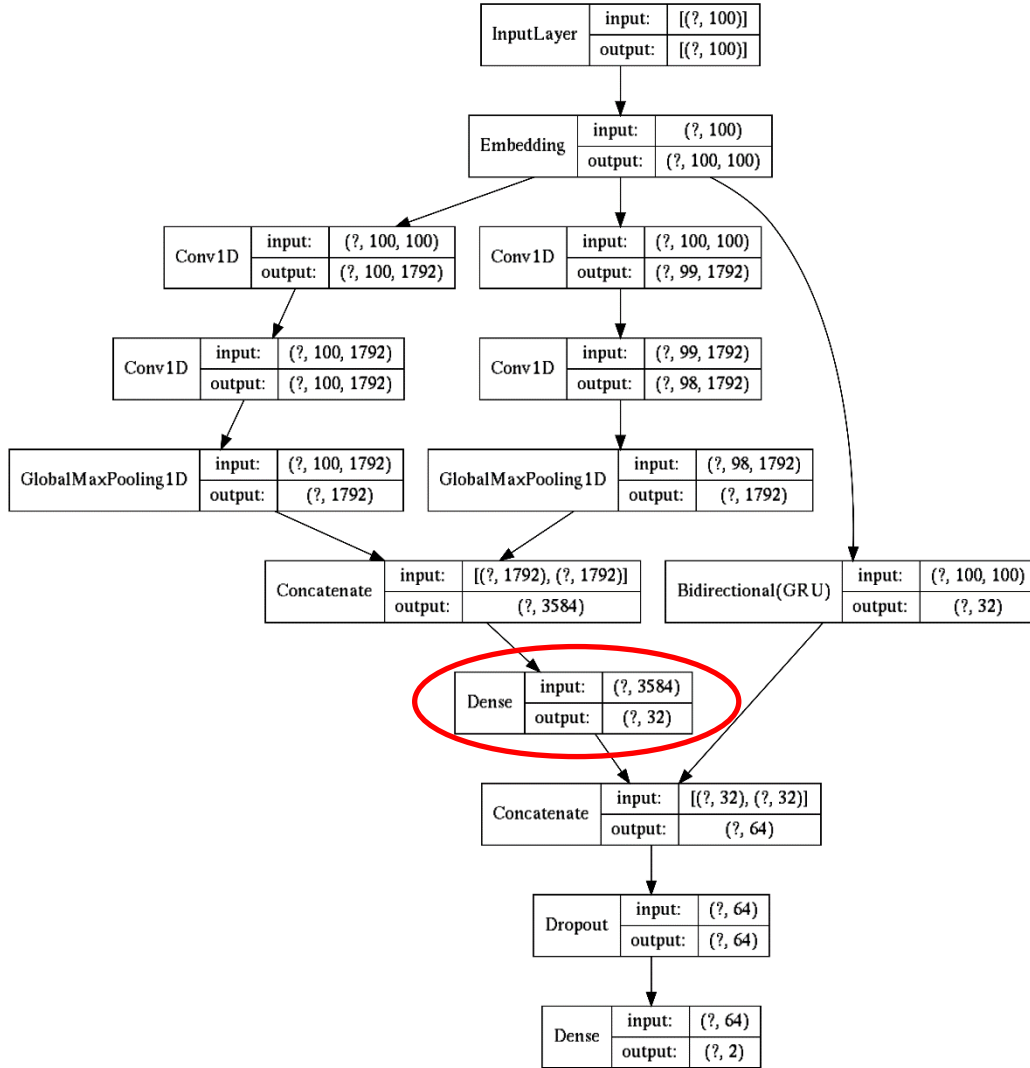


Figure 11.7: Architecture of the combination model. Red circle marks downsizing layer.

The red circle on Figure 11.7 marks a layer which was inserted to downsize the output of the convolutional architecture to have the same size as that of the recurrent architecture. The reasoning behind downsizing to equal sizes is that the network can weigh the output from the two architectures equally. The model was tested with and without the downsizing layer.

The training loop consisted of training on the setting hyperparameters of the final recurrent and convolutional models. The best combination model

was achieved with trainable embeddings, dropout of 0.3, and with no downsizing layer. It achieved a minimum loss averaged over the best and 2 prior epochs of 0.240 and a validation accuracy of 91.8 %.

11.6 Selection of best model

Table 11.2 provides an overview of the minimum loss and corresponding accuracy of each tested model.

Model	Minimum loss	Avg. true class prob.	Accuracy
Recurrent	0.273	0.761	90.2 %
Convolutional	0.237	0.789	91.9 %
Combination	0.240	0.787	91.8 %

Table 11.2: Overview of the results of the recurrent, convolutional, and combination model

Table 11.2 also shows the corresponding average probability with which the model classifies the true class in the validation set. It is calculated from the loss function

$$L = -\log(\hat{y}_i) \rightarrow \hat{y}_i = e^{-L} \quad (11.1)$$

where L is the average loss of the validation set samples and \hat{y}_i is the average softmax score of the correct class.

It is seen that the convolutional model has the lowest loss of 0.2366 and a corresponding average certainty on the correct class of 78.93 %. Therefore, the convolutional model with the selected hyperparameters is chosen as the best model.

It is noted that all models did better than the 80.65 % accuracy of the rule-based approach used for baseline measure described in section 11.1.

11.7 Augmentation

Data augmentation was tested on the convolutional model to see if additional data would improve model performance. The input dropout and back-translation augmentation methods were tested. Each augmentation method was used to increase the number of positive training samples by 50 % and 100 %. The negative samples were increased the same amount as the positive

samples to keep a balanced dataset by including left-over samples from the extracted negative samples. In this way, data augmentation increases the dataset size and allows for use of more of the random negative and misinterpretable negative samples which the data contains in abundance.

11.7.1 Input dropout

The input dropout method drops random words from the positive input samples to generate new samples. It forces the model to learn based on the whole structure of the sentence and to not rely only on single words.

An algorithm was used to drop 1 word from the sentence if it was less than 8 words, 2 if less than 15, and 3 if more than 14. The code can be seen in Appendix O.

The result of the input dropout method is shown in Table 11.3.

Input dropout augmentation	
Increase in data	Minimum loss
0 %	0.237
50 %	0.246
100 %	0.242

Table 11.3: Results of increasing the dataset with input dropout augmentation

Input dropout augmentation did not improve the model performance. This suggests that although the algorithm was exposed to more data, this data did not add any additional information which could be used to classify samples of the validation set.

11.7.2 Backtranslation

Augmentation using backtranslation has been used with success in both neural machine translation and text classification tasks [86], [87]. Backtranslation works by translating the training data to another language and then translating it back to the source language as illustrated in Figure 11.8.

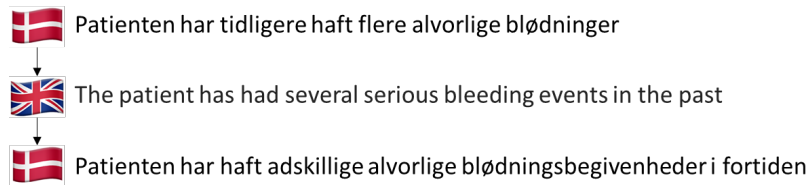


Figure 11.8: An example of backtranslation

The example of Figure 11.8 shows that the original sentence keeps its meaning, but some words change to a synonym, e.g. ‘*tidligere*’ is replaced by ‘*fortiden*’. Moreover, words also change position as seen with ‘*tidligere*’ and ‘*fortiden*’. The idea behind backtranslation for augmentation is that the generated samples will represent other versions of the text that will make the algorithm generalize better to unseen data.

Data was translated using the Google Translate API for Python (see Appendix O). The results are shown in Table 11.4.

Backtranslation augmentation	
Increase in data	Minimum loss
0 %	0.237
50 %	0.239
100 %	0.245

Table 11.4: Results of increasing the dataset with backtranslation augmentation

Backtranslation augmentation did not improve the model performance. Like for input dropout augmentation, this suggest that although the algorithm was exposed to more data, this data did not add any additional information which could be used to classify samples.

11.8 Final model ensemble

The convolutional model from section 11.3.3 was chosen as the best. To obtain the best performance with this model, it was trained multiple times with the same settings. The stochastic nature of mini-batch gradient descent and different initialization of the weights at the start of each training means that they might give different results.

The best models of each training were saved to be used in an ensemble classification. Ensemble classification averages the predictions of multiple models at test time. It is a reliable way to increase performance because

averaging decreases noise, and different models might have high certainty on different samples [88].

Ensembles of the top-1 to the top-21 best models were tested on the validation set to decide the number of models to include in the ensemble. Figure 11.9 shows the result of the ensembles of different sizes.

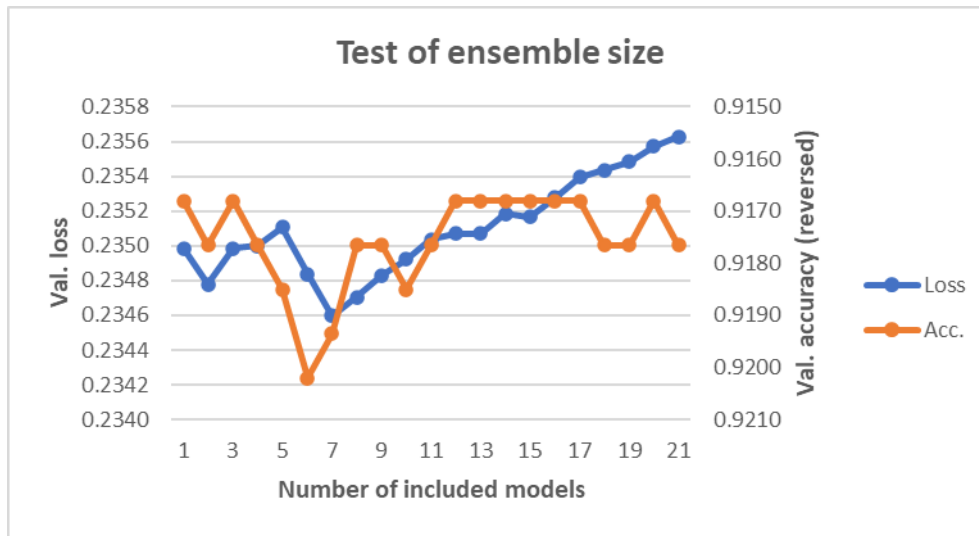


Figure 11.9: Visualization of the performance of different ensemble sizes

It is seen that including 7 models in the ensemble gives the best loss on the validation set of 0.235⁵. This corresponds to an average certainty on the correct class of 79.1 %. The achieved validation accuracy is 91.9 %. This model ensemble will be used on the test set in section 12.1.

11.9 Chapter summary

This chapter presented the training of a convolutional, recurrent and combination model. The training was done in three steps: First, the architecture hyperparameters, e.g. layer types and sizes, were tuned. Secondly, the best optimizer and learning rate for that architecture was found. Thirdly, the setting hyperparameters such as dropout and regularization were tuned. The validation loss was used as metric for deciding on the hyperparameters.

⁵ The ensemble loss is not directly comparable to the single loss output of the keras evaluate function on individual models reported so far. Rounding errors occur because the loss is calculated from an average of 7 predictions which are returned with fewer decimals than in the evaluation of a single model in Keras.

The convolutional model taking non-trainable embeddings as input produced the best loss of the three models. The model has 2 convolutional series with each 2 stacked convolutional layers. Each layer has 1792 filters with filter sizes 1 and 2 respectively for each series. The two series are global max pooled, concatenated, and sent through a dropout layer of 0.4 and an output dense layer with regularization 0.0001 to produce the prediction.

The convolutional model was tested with two different augmentation methods, but it did not improve the performance.

Finally, an ensemble of the 7 best iterations of the selected convolutional model was made. The ensemble produced a validation loss of 0.235 and a validation accuracy of 91.9 %.

Chapter 12

12 Results

This chapter presents the results of the best model in two different tests. First, the model is run on the test set to measure its performance on the training objective. It is then tested in a real-life setting on an EHR.

12.1 Test set performance

The test set measures the model performance on the training objective of classifying samples in a balanced dataset of 50 % positive samples and 50 % negative samples. The negative samples further consisted of 50 % misinterpretable negative samples and 50 % random negative samples. The balanced objective was set up to perform equally well on positive and negative samples.

The results on the test set is presented in Table 12.1. Validation results are included for comparison.

Validation and test set results		
	Loss	Accuracy
Validation	0.235	91.9 %
Test	0.233	89.2 %

Table 12.1: Validation and test set results

It can be seen from the losses, that the model performs almost equally on the validation and test set, though slightly better on the test set. As the model is trained using the loss, it means that there has been no overfitting on the validation set from the hyperparameter tuning. Average certainty on the correct class, derived from the loss using equation (11.1), is 79.1 % for the validation set and 79.3 % for the test set.

Looking at the accuracy, the model performance is worse on the test set than the validation set. This means that, while on average the model has a better certainty on the correct class, it is mistaken in 10.8 % of cases on the test set and 8.1 % of cases on the validation set.

Table 12.2 shows the confusion matrix on the test set and Table 12.3 shows the measures.

		Predicted	
		0	1
True	0	521	68
	1	59	530

Table 12.2: Confusion matrix on the test set. 0: Negative sentence. 1: Positive sentence.

Test set measures	
Sensitivity	0.900
Specificity	0.885
Precision	0.886
Negative predictive value	0.898
F1	0.893
Accuracy	0.892

Table 12.3: Measures on the test set

As the test set is balanced, accuracy is a meaningful way of analyzing the result. The model classifies 89.2 % of the test set correctly. It is further seen that the model correctly classifies 90.0 % of all positive samples and 88.5 % of all negative samples. Furthermore, the model is correct 88.6 % of the times that it classifies a sample as positive and 89.8 % of the times it classifies a sample as negative.

Focusing on the 589 negative samples, it is further noted that 277 out of 294 random negative samples were classified correctly (94.2 %) while 244 out of 295 misinterpretable negative samples were classified correctly (82.7 %). This means that the classification accuracy is above the model specificity for random negative samples and below for misinterpretable negative samples.

12.2 Real-life EHR performance

The model was tested in a real-life setting on an EHR containing 208 notes (166 negative and 42 positive) that had been annotated by the expert annotator. The test worked on the note-level, meaning that the objective of the test was to classify a note as either positive or negative. A positive note is one

that contains one or more sentences that have been classified as positive by the model.

Table 12.4 shows the confusion matrix of the real-life EHR test and Table 12.5 shows the measures.

		Predicted	
		0	1
True	0	103	63
	1	1	41

Table 12.4: Confusion matrix on the real-life EHR test. 0: Negative note. 1: Positive note.

Real-life EHR test measures	
Sensitivity	0.976
Specificity	0.620
Precision	0.394
Negative predictive value	0.990
F1	0.562

Table 12.5: Measures on the real-life EHR test

As the real-life EHR test is highly unbalanced with respect to the number of negative (166) and positive (42) notes, the accuracy is not reported. The model correctly classifies 97.6 % of all positive notes and 62.0 % of all negative notes. The model is correct 39.4 % of the times that it classifies a note as positive and 99.0 % of the times it classifies a note as negative.

Chapter 13

13 Discussion

This chapter will discuss the methods and results obtained throughout this thesis. First, the results of and the convolutional neural network (CNN) model will be analyzed. Afterwards, different procedures to improve the results are presented, and finally important aspects of how the model can be implemented in a real-life setting are discussed.

Per the objective of the thesis, the goal of the model was restrained to detecting any bleeding. The final goal of the research, though, is to be able to detect relevant bleeding occurrences within 3 months before admission for the IMPROVE score as described in section 2.2.3. Therefore, the aspect of including the doctor in the decision process of the relevancy of the bleeding is included in the discussion of the real-life applicability of the model.

13.1 Analysis of results

This section first discusses the results of the model on the balanced test set, and then the results on classifying notes of a single electronic health record (EHR) to examine how the model would perform in a real-life setting.

Notice that the samples of the balanced test set are sentences, as this is how the model was trained. The samples when classifying a single EHR in the real-life test is the notes of the EHR, a note being defined as positive if it contains any sentence classified as positive by the model.

The model performance in both tests is analyzed in the context of predicting positive EHR notes because this is a way that the model prediction can be reported to doctors. Reading the full note instead of an individual sentence should give the doctor the context to evaluate if the model's suggestion is a relevant bleeding.

13.1.1 Analysis of test set performance

It is seen from the balanced test set results that the model has an 89.2 % accuracy and that it classifies positive and negative sentences almost equally well which was the purpose of the balancing.

This distribution of samples does not reflect the real-life distribution though. While the test set consists of 50 % positive, 25 % misinterpretable negative, and 25 % random negative sentences, it is most likely that the larger part of a random EHR will be random negative sentences not having anything to do with bleeding. The model has an accuracy of 94.2 % on this type, and it can be assumed that the real-life accuracy on EHR sentences is close to that.

The number of sentences in an EHR will pose a problem to the model performance though. The final goal of the model is to find if a patient has had a relevant bleeding within 3 months before the admission. A patient can have many EHR notes attached to 3 months and between them hundredths of sentences. Even with an accuracy as high as the assumed 94.2 %, the model will statistically misclassify 1 in 17 random negative sentences as positive.

13.1.2 Analysis of EHR test performance

This section examines the performance of the model on classifying notes of a single EHR. For the following discussion, it is noted that a test on one single EHR is not enough to generalize the results to other EHRs, but it can give some insight into the strengths and shortcomings of the model.

Figure 13.1 shows an example of a correctly classified note from the test set that could be returned to the doctor for verification of relevancy. For the figures in this section, a red square marks a positive annotated sentence, a light red highlight marks a classification of low probability, while a dark red highlight signifies a high probability. The black squares are redacted data.



Figure 13.1: A correctly classified EHR note. Red square: Positive annotation. Light red highlight: Low prob. positive pred. Dark red highlight: High prob. positive pred. Black: Redacted.

Looking at the performance on the real-life EHR test, two results stand out:

- On classifying the EHR note, the model has a high sensitivity of 97.6 % but the precision is only 38.3 %.
- The specificity is low at 62.0 % but the negative predictive value is very high at 99.0 %.

These results indicate, per the high sensitivity, that almost all true positives will be among the predicted positives. This is usually very important in a clinical setting. But it comes at a price of misclassifying many negatives as positives per the low precision.

Figure 13.2 shows a note that exemplifies why the sensitivity is high.

Broedtekst Klinisk kontakt . Anamnese . Velbehandlede varicer ved skopien i går
 og der skal ikke foretages ny endoskopi før tidligst 2 uger fra
 Hun er således stabil blødningsmæssigt . Hun afvandes der er
 mangler stadig en del . Hun indtager flydende kost uden problemer .
 Fortsat i antibiotisk behandling . Får Beta blokade som profylakse mod recidiv
 blødning . Hun er indforstået med at hun overflyttes til
 til medicinsk færdigbehandling og dette aftalt med bagvagt på Inden hun
 kan medicineres peroralt . Behandlingsplan . Ordination af medicin . sep. inj.
 Furix hun kan genoptage vanlig insulin behandling NovoMix rp . inj. NovoMix
 30 + 0' + 38 hun holder fortsat pause med ace hæmmer
 og calcium antagonist

Figure 13.2: Correctly classified EHR note with some sentence misclassifications. Red square: Positive annotation. Light red highlight: Low prob. positive pred. Dark red highlight: High prob. positive pred. Black: Redacted.

Firstly, the correct positive classification of a note can be ‘saved’ by classifying a wrong sentence as positive. Secondly, it also shows how the presence of two positive sentences in one note increases the chance of a correct classification of the note.

The poor precision on the real-life EHR test follows directly from the model performance on the balanced test set. One note contains many sentences and if one of those sentences are classified as positive by the model, then the whole note is considered positive. If it like in section 13.1.1 is assumed that a note only contains random negative sentences, then for each 17 sentences in a note, the model will statistically misclassify one sentence as positive and thereby the whole note.

On the other hand, the high negative predictive value indicates that almost all notes that are predicted negative, are correct. This means that the model has a good potential for filtering notes that are negatives.

Since the model is very accurate on the negative samples that it filters, the remaining notes will contain almost all positive cases. The positive predicted notes can then be returned to a doctor for final evaluation. For the specific EHR tested in this subsection, a doctor would have to read through 104 notes out of 208 to locate the 41 notes that contain a bleeding occurrence. This is a reduction of 50 %.

Improving especially the specificity and precision of the model would decrease the workload of the doctor reviewing the notes.

13.2 Analysis of model

This section analyses the trained model by discussing why the convolutional model performed better than the recurrent model. It discusses the function of the stacked layers and filter sizes in the convolutional model and analyzes them using the Grad-CAM method. Then, the effect of spelling mistakes and gender and age bias of the model is examined.

13.2.1 Convolutional vs. recurrent model

It is an interesting result that a convolutional architecture did better than a recurrent architecture on a text classification objective.

In theory, a recurrent model should outperform a convolutional architecture on data where the temporal features are important because it has the possibility of understanding each word in relation to prior context (and future context in the case of bidirectional architectures). The convolutional model performing better might indicate that the temporal aspect of the data is not important enough to outweigh the features of a convolutional model.

The convolutional model looks for patterns in the input information using filters. Each filter can decode information in small units, but the model has no sense of the order of information outside of the units covered by the filter size, because of the global max-pool operation. The better performance of the convolutional model indicates that the order of information is only significant for the classification within small units of the text. The size of these units is covered in section 13.2.2. The full sentence structure captured by the recurrent model does not seem as important as being able to decode the information from the smaller units - and this is where the convolutional model has its apparent advantage: Each filter in the model can learn to decode different kinds of information from the units of words. The model of this thesis used 3,584 different filters to decode information from the input embeddings and 3,584 further filters to decode information from the output of the first filters - this amounts to 10,178,560 parameters. This is to be compared to the 11,328 learnable parameters of the bidirectional Gated Recurrent Unit layer. Both

models were tuned to their optimal number of parameters and this indicates that the convolutional structure can utilize the added capacity better.

13.2.2 Filter sizes and stacked layers

It was established in section 13.2.1 that the convolutional architecture most likely outperformed the recurrent because of its ability to decode information from small units of the text using filters, and because the classification of a sentence does not rely heavily on the structure of the full sentence. This section discusses the reasons why two stacked layers of filter size 1, and two of filter size 2 works best with this model.

It is surprising that two stacked convolutional layers of filter size 1 worked well because the subarchitecture can only extract information from a single word at a time. The success might be comprehended better when thinking of a single word as an embedding of 100 dimensions instead of just a word. As described in section 6.2.2, different dimensions of meaning are encoded in the word embeddings - one part might hold information on the tense of the word, one part if it is singular or plural, including more abstract dimensions of meaning. In this way, the filters of size 1 working on the input might decode information about different dimensions of meaning. The next stacked layer might then work to decode information from the features discovered by the first layer.

The fact that a filter of size 1 improves the model reveals that a lot of the information needed to classify a sentence can be extracted from a single word like *'blødning'*, *'ikke'*, or *'hæmatom'*.

It makes more intuitive sense that two stacked convolutional layers of filter size 2 works well. The first layer decodes two words at a time with stride 1. The words decoded in pairs for a sentence *'Der er opstået en blødning'* would be *'Der er'*, *'er opstået'*, *'opstået en'*, and *'en blødning'*. When the 1792 first layer filters have decoded information about the pairs, the second layer filters, again 1792 filters of size 2, will decode information from the features of two pairs at a time. This means it will consider information about the units *'Der er opstået'*, *'er opstået en'*, and *'opstået en blødning'*. The model stopped improving for filter sizes that decode information from more than 3 words at a time. This indicates that 3 is the maximum size of the significant patterns of words where it is necessary to store information about the context in order

to classify the sentences well. This is interesting when, in theory, a filter size covering the entire sentence could have been applied, giving the model information about the whole sentence structure - imitating some of the functionality of the recurrent model.

Finally, it is the inclusion of both filter sizes and them complementing each other that gives the model the best performance.

To further examine the functions of the convolutional layers (conv-layers) and filters, they are analyzed with the Grad-CAM.

13.2.3 Inner workings of the convolutional neural network

To achieve a better understanding of why the developed CNN based model classifies as it does, the following section will use the Grad-CAM method described in section 6.3.1.2 to analyze examples of true positives, true negatives, false negatives and false positives from the test set of the sentence dataset. The Grad-CAM visualizes which words the model considers important for its classification in each of the four conv-layers. The importance of specific words will be visualized using a heatmap where words which are considered most important are dark red, less important words are light red and non-important words are white. The code is attached in Appendix P. Figure 13.3 visualizes this approach where a positive sample from the test set,

'Pt. føler, at blødningen fra højre øje er i bedring',

is sent through the network.

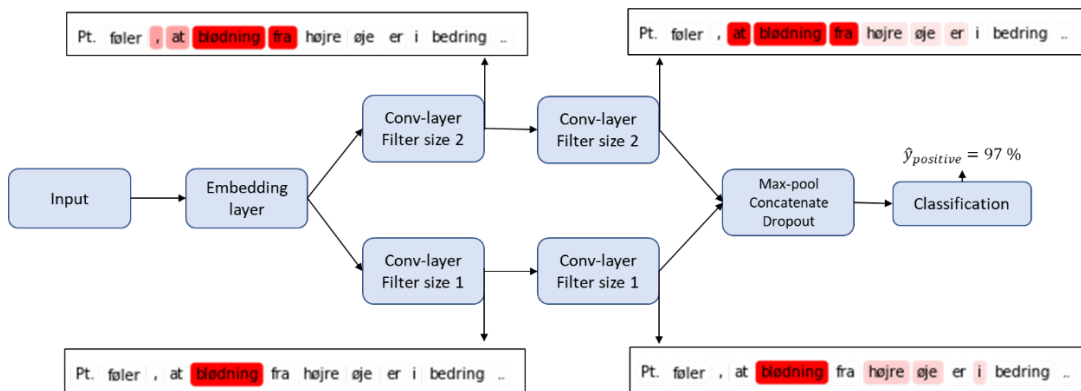


Figure 13.3: Grad-CAM visualization of each conv-layer

The network correctly classifies the sample as positive with a softmax score of 0.97. For the first conv-layer of filter size 2, it is seen that the network pays attention to the two bigrams ‘, at’ and ‘blødning fra’. In the subsequent conv-layer with filter size 2, the network removes its attention from the comma and focuses significantly on the 3-gram ‘at blødning fra’, which makes sense because ‘blødning’ is an important word. Moreover, the conv-layer extends its attention to also focus on the words ‘højre øje er’. For the first conv-layer of filter size 1 it is seen that ‘blødning’ is marked as important. In the subsequent conv-layer of size 1 the word ‘blødning’ is still considered most important, and moreover the network pays attention to the words ‘højre’, ‘øje’ and ‘i’. This pattern where the first conv-layer of size 1 only pays attention to obvious bleeding-indicating words after which the subsequent conv-layer extends its attention to also look for other words, seems to be a general pattern for the positive class. This is also seen in the upcoming examples. To ease the interpretation of samples, the following heatmaps are instead shown in tables.

13.2.3.1 Example of true positive

Table 13.1 shows an additional example of a true positive sample from the test set.

Layer	Sample
Input	KL 02.20 er der kommet 375 ml blodtilblandet urin i kassetten.
Conv1 filter size 1	KL 02.20 er der kommet 375 ml blodtilblandet urin i kassetten .
Conv2 filter size 1	KL 02.20 er der kommet 375 ml blodtilblandet urin i kassetten .
Conv1 filter size 2	KL 02.20 er der kommet 375 ml blodtilblandet urin i kassetten .
Conv2 filter size 2	KL 02.20 er der kommet 375 ml blodtilblandet urin i kassetten .

Table 13.1: Grad-CAM visualization of a true positive

For both conv-layers of filter size 1 it is seen that ‘blodtilblandet’ and ‘urin’ are considered important. The second conv-layer also pays a little attention to the words ‘der’ and ‘kommet’, supporting the argument that this layer

extends its attention to more words than the first conv-layer. Both conv-layers of filter size 2 also pay attention to ‘*blodtilblandet*’ and ‘*urin*’, and moreover the words ‘*ml*’ and ‘*i*’ are also highlighted.

13.2.3.2 Example of true negative

Table 13.2 shows the heatmaps for a true negative sample from the test set.

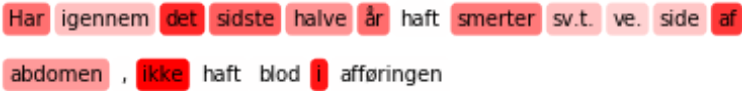
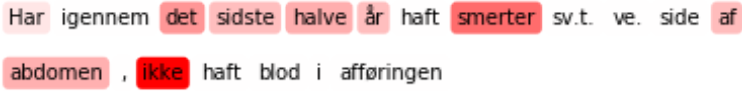
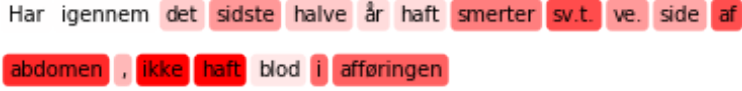
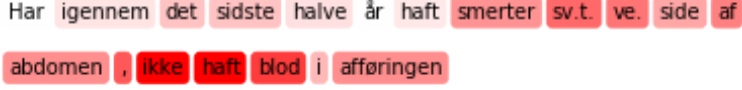
Layer	Sample
Input	Har igennem det sidste halve år haft smerter sv.t. ve. side af abdomen, ikke haft blod i afføringen
Conv1 filter size 1	
Conv2 filter size 1	
Conv1 filter size 2	
Conv2 filter size 2	

Table 13.2: Grad-CAM visualization of a true negative

In order to make a correct prediction, the model needs to know that the word ‘*ikke*’ refers to ‘*blod*’. In the first conv-layer with filter size 1 it is seen that the model looks at many different words, but ‘*ikke*’ is the most highlighted. In the subsequent conv-layer with filter size 1 the number of words which is highlighted is narrowed down while ‘*ikke*’ is still the word with most attention. This suggest that for the *negative* class the first conv-layer is looking broadly after important words, while the subsequent conv-layer tries to reduce the number of relevant words. For the conv-layers with filter size 2 it is seen that the first layer pays most attention to ‘*ikke haft*’ and the subsequent layer extends this attention to ‘*ikke haft blod*’, suggesting that the model infers that ‘*ikke*’ refers to ‘*blod*’.

13.2.3.3 Example of false negative

Table 13.3 shows a false negative sample from the test set.

Layer	Sample
Input	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes
Conv1 filter size 1	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes
Conv2 filter size 1	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes
Conv1 filter size 2	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes
Conv2 filter size 2	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes

Table 13.3: Grad-CAM visualization of a false negative

It is seen that the network almost looks at everything else than the word ‘corpushæmoragien’ which is the bleeding-relevant word. This means that all words except from ‘corpushæmoragien’ leads the classification towards a negative prediction. If instead, the Grad-CAM is used to visualize which words that leads the classification towards a positive prediction, ‘corpushæmoragien’ seems important as seen in Table 13.4.

Layer	Sample
Conv1 filter size 1	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes
Conv2 filter size 1	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes
Conv1 filter size 2	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes
Conv2 filter size 2	Patienten har kontroltid om 2 3 uger i ambulatoriet hvor corpushæmoragien o.dx. skal vurderes

Table 13.4: Grad-CAM visualization of words leading towards negative prediction of a true positive

Even though the model correctly identifies the bleeding-relevant word it has considered the negative class triggering words as more important for the final prediction.

13.2.3.4 Example of false positive

Table 13.5 shows a false positive sample from the test set.

Layer	Sample
Input	Har hostet lidt på det sidste af og til med ekspektoration aldrig blodtilblandet.
Conv1 filter size 1	Har hostet lidt på det sidste af og til med ekspektoration aldrig blodtilblandet .
Conv2 filter size 1	Har hostet lidt på det sidste af og til med ekspektoration aldrig blodtilblandet .
Conv1 filter size 2	Har hostet lidt på det sidste af og til med ekspektoration aldrig blodtilblandet .
Conv2 filter size 2	Har hostet lidt på det sidste af og til med ekspektoration aldrig blodtilblandet .

Table 13.5: Grad-CAM visualization of a false positive

It is seen that the two conv-layers with a filter size of 1 mainly focus on ‘*blodtilblandet*’. The first conv-layer with filter size 2 both focuses on ‘*hostet lidt*’ and ‘*blodtilblandet .*’. The subsequent conv-layer attends to ‘*har hostet lidt*’ which does not make intuitively sense but it might be because many positive samples have included the word ‘*hostet*’.

If the Grad-CAM is used to see which words are leading the classification towards a negative prediction it would make intuitive sense that the first conv-layer with a filter size of 2 focuses significantly on the bigram ‘*aldrig blodtilblandet*’. As seen below on Figure 13.4, this is not the case though.

Har hostet lidt på det sidste af og til med ekspektoration aldrig blodtilblandet .

Figure 13.4: Grad-CAM visualization of words leading towards a negative prediction for a false positive

Although there is some attention to the bigram, it considers ‘*af og til med*’ as more important. This might be one of the reasons that the sample is classified as positive.

13.2.3.5 Example of false positive

Table 13.6 shows a false positive from the test set. Unlike the above example, this sample is drawn from the *random* samples, meaning that it has nothing to do with a bleeding.

Layer	Sample
Input	Ukompliceret ned til duodenum 2. stykke hvor man ser det uregelmæssige papilkompleks.
Conv1 filter size 1	Ukompliceret ned til duodenum 2. stykke hvor man ser det uregelmæssige papilkompleks .
Conv2 filter size 1	Ukompliceret ned til duodenum 2. stykke hvor man ser det uregelmæssige papilkompleks .
Conv1 filter size 2	Ukompliceret ned til duodenum 2. stykke hvor man ser det uregelmæssige papilkompleks .
Conv2 filter size 2	Ukompliceret ned til duodenum 2. stykke hvor man ser det uregelmæssige papilkompleks .

Table 13.6: Grad-CAM visualization of a false positive

It is seen that the model mainly focuses on ‘*uregelmæssig*’ which might be because it has often seen ‘*uregelmæssig*’ in positive samples. The model’s inability to infer that ‘*uregelmæssig*’ is not always a bleeding triggering word is one of the things that should be improved in order to achieve a better model.

The above visualizations of the Grad-CAM method shows that the inner workings of the CNN are often easily understandable, especially on the positive predictions where bleeding relevant words are clearly marked for the tested sentences. This is important because it increases the transparency of the model and makes it possible to explain the decision process of the model to e.g. a doctor.

13.2.4 Effect of spelling mistakes

The following will test what happens when a bleeding relevant word is misspelled. The sentences tested are seen in Table 13.7 together with the model's prediction and softmax score for that prediction.

Sentence	Prediction	Softmax score
'Det bløder kraftigt fra patientens ben'	Positive	0.83
'Det blødder kraftigt fra patientens ben'	Negative	0.73

Table 13.7: Test of the model's reaction to spelling mistakes

It is seen that the only difference between the two sentences is the spelling mistake in the second sentence where 'blødning' is misspelled as 'bløddning'. This makes the network misclassify the sentence which is a big problem as spelling mistakes are common in medical texts. One way to mitigate this problem is to develop high-quality word embeddings that can handle misspellings such as subword models, e.g. fastText.

13.2.5 Gender and age bias of the model

Bias is an important factor to recognize when considering implementing machine learning algorithms as part of a decision support process in the health care sector. It is known that neural networks are vulnerable to e.g. gender and age bias [89], [90]. Bias might arise from the EHR data used to train the model, but it can also be hidden inside the pre-trained word embeddings. A small test is therefore executed to explore if bias is present in the developed model. This is done by classifying different sentences where the only difference between them is the gender or age as seen in Table 13.8.

Sentence	Prediction	Softmax score
Gender weighted tokens		
Personen har en blødning	Positive	0.93
Kvinden har en blødning	Positive	0.93
Manden har en blødning	Positive	0.93
Pigen har en blødning	Positive	0.90
Drengen har en blødning	Positive	0.93
Age and gender weighted tokens		
Ældre mand har en blødning	Positive	0.92
Ung mand har en blødning	Positive	0.92
Ældre kvinde har en blødning	Positive	0.94
Ung kvinde har en blødning	Positive	0.94

Table 13.8: Softmax scores of sentences with gender and age weighted tokens

The table shows that the model predicts all sentences as positives, but the softmax scores differ slightly. For the gender neutral token ‘*Personen*’ and the gender weighted tokens ‘*Kvinden*’ and ‘*Manden*’ the softmax scores are equal. The softmax scores differ the most in the sentence with the gender weighted token ‘*Pigen*’ where it drops to 0.90. Although this is not a huge difference, it is notable. When assigning the age weighted tokens ‘*Ældre*’/’*Ung*’ in front of the gender weighted tokens ‘*mand*’/’*kvinde*’, the softmax scores are the same for each of the gender weighted tokens, but it changes from 0.92 to 0.94 when the gender weighted tokens change.

This analysis is not complete, but it does suggest that gender bias could be a problem in the model although only small changes are seen in the softmax scores. These small changes though, can potentially assign different predictions to samples where the softmax score is ~ 0.50 solely based on the gender.

13.3 Improvement of performance

There are several ways to possibly improve the performance of the model. This section discusses improvement of data quality, increasing the amount of training data, and the possibility of other types of models and word embeddings increasing the performance.

13.3.1 Improvement of data quality

One way to improve the performance of the model is to train it on data of higher quality. Higher quality data can be obtained by improving the annotation.

During training, it is very important to feed unambiguous data to the model as to not confuse it. It is also important for the validation and test sets to reflect the real world to get an accurate evaluation of the model. All this depends on an accurate annotation of the data. An inter annotator agreement of kappa score 0.75 was reached between the 12 annotators for a subset of the training data, corresponding to substantial agreement. Improving this would most likely reflect in better model performance.

In retrospect, several changes to the annotation process could most likely have improved the inter annotator agreement:

Emphasis on sentence annotation

More emphasis should be on the need for sentence-wise annotation. In some cases, text passages of several sentences indicating bleeding were annotated as a single sample instead of sentence-wise. Sentence-wise annotation is important because the positive annotations and the samples generated by the sentence splitter must be of the same format. Only in cases where e.g. two sentences in a row is necessary for understanding a single sentence as indicating bleeding can they be annotated as a single sample.

Intro meeting

In retrospect, an intro meeting with the annotators would have mitigated many errors in the annotation process. Examples should be worked through and questions should be answered to standardize the annotation style. Some of the noted uncertainties during exploration of the data were:

- Suspicion of a bleeding occurrence
- *‘No further bleeding observed’*
- *‘Has not had any problems with bleeding since’*

The first example is a question of deciding if suspicions are valid bleeding occurrences or not. The last two examples are sentences from EHRs that

clearly indicate that the patient has previously had a bleeding occurrence but not necessarily at the time where the EHR note entry was made.

Follow up

Follow up on the annotations after the annotators have reviewed a couple of EHRs to make sure that all annotators are adhering to the decided annotation style.

Annotation software

The expert annotator commented that the Word annotation macros worked slowly when the document became large. Several start-tags without end-tags and end-tags without start-tags were found during data exploration. A dedicated annotation software could help avoid these types of problems and should be sought to develop or buy.

13.3.2 Increase amount of data

The training set used to achieve the final result consists of 9430 samples. In order to explore if more data potentially can improve the result, the training dataset was reduced to 25 %, 50 % and 75 % of its original size. The reduced training sets were tested on the original sized validation set (1178 samples) with the same architecture and hyperparameters that produced the best result with 100 % of the data. The result is seen in Figure 13.5.

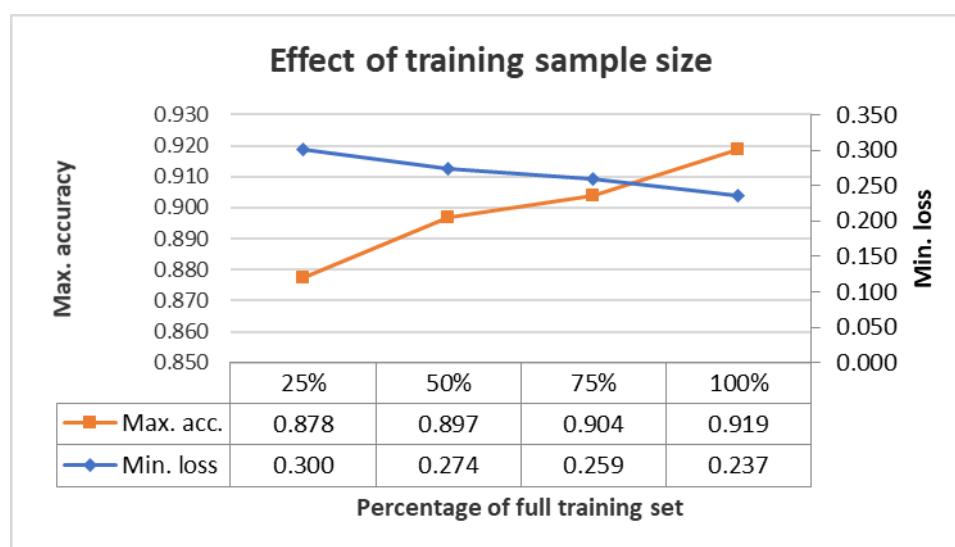


Figure 13.5: Graph of the loss and accuracy for different sizes of the training set

As seen, the loss decreases and the accuracy increases as the number of training samples increases. The trends of the curves do not seem to stop at 100 % of the data. It is therefore hypothesized that more data will be beneficial for improving the model. More data is important because it increases the variability of the training data which is needed for the model to generalize better to unseen data.

In section 11.7, it was found that augmentation did not improve the model performance which emphasizes the necessity of acquiring and annotating more EHR data.

13.3.3 Other models

To improve the final result, other models could also have been tested. It has e.g. been shown that stacking multiple conv-layers (up to 29) can be used to achieve state-of-the-art results on text classification tasks [91]. Moreover, as seen in Chapter 5 (Related work), other papers have used attention layers and the BERT [34] model which could also be explored.

13.3.4 Word embeddings

This thesis used GloVe pre-trained word embeddings trained on a medical corpus. As described in section 9.2, the GloVe embeddings were found to work better than the fastText embeddings as they performed best on the validation set.

Word embeddings can also be tested by measuring how close related words are in the embedding space. GloVe is trained with the purpose of similar words having similar embeddings. The cosine distance measures the similarity of the embeddings of two words:

$$similarity = \cos(\theta) = \frac{A \cdot B}{\|A\| \|B\|} \quad (13.1)$$

where A and B are the embedding vectors. A similarity of 1 corresponds to $\theta = 0^\circ$, i.e. the two embeddings have the same orientation. A similarity of 0 corresponds to $\theta = 90^\circ$, i.e. the embeddings are orthogonal. Using this similarity measure, it is expected that the most similar words to a specific word are also semantically similar. E.g. *‘blødning’* would be expected to have most

similar words such as ‘*blødninger*’, ‘*blødingerne*’ and ‘*hæma*’. This is not the case for the top-3 most similar words of the examples shown in Table 13.9.

Word	1 st	2 nd	3 rd
blødning	/postpartum	Symbolsk	perforation
Epistaxis	næseblødning	næseblod	urinretention
blod	afføring	afføringen	frisk
hæmoragi	subkonjunktival	intracerebral	cerebral
hæmothrox	pneumo-	pneumothorax	luxation

Table 13.9: Selected words and their 3 most similar words

For this small test, it is seen that the nearest neighbors seldomly are semantically similar words. This means that if a filter of a conv-layer outputs a high value when it sees ‘*blod*’, it will output a similar high value if it sees ‘*afføring*’ because ‘*blod*’ and ‘*afføring*’ have similar embeddings. This is a weakness of the embeddings and it can therefore be hypothesized that developing high-quality word embeddings will have the potential to improve the result.

13.4 Implementation of a Clinical Decision Support System

While the model, per the objective, was restrained to detecting all bleeding occurrences, the implementation of the model is discussed in the context of detecting relevant bleeding occurrences for the IMPROVE score as described in section 2.2.3. Because of the model’s restraint, this must be done in collaboration with a human doctor. One way to facilitate the cooperation between the model and a doctor is implementing a Clinical Decision Support System (CDSS).

13.4.1 Clinical Decision Support System for relevant bleeding

There are several indicators that a CDSS would be a good way of implementing the bleeding detection model in the health care sector:

- The model cannot distinguish relevant bleedings from irrelevant bleedings that are not considered for the IMPROVE score. At this stage, a doctor is needed for the final decision on relevancy of the bleeding.

- The results have indicated that the model has a high sensitivity and high negative predictive value which means that it makes few mistakes when detecting bleeding-positive notes to be evaluated for relevancy by the doctor.
- Based on the limited test on a single real-life EHR, the model could filter away 50 % of notes not needing to be evaluated by the doctor. This is an obvious decrease in workload for the doctor.
- It was discussed in section 2.3 that studies show that thromboprophylaxis guidelines are not always used in practice. Clinical practice has shown to improve when implementing a computer based CDSS that provides actionable recommendations automatically at the location of decision-making [92].

The bleeding detection model presented in this thesis would be the knowledge base of the CDSS. It should be integrated in the EHR system of the hospital with access to the free text of the EHRs. Using the CDSS, the doctor can detect relevant bleeding occurrences in EHRs more easily. If a relevant bleeding has occurred within 3 months prior to admission, it will be counted in the IMPROVE scoring system. The IMPROVE score can finally be used to decide on which type, if any, thromboprophylaxis should be given.

13.4.2 Communication mechanism

As a CDSS is supposed to support the decision of doctors, the communication aspect is very important [93]. It is suggested that the communication of the CDSS is integrated in the EHR. It should prompt the doctor inside the EHR system of the hospital to evaluate the thromboprophylaxis treatment of a patient when he or she is newly admitted or when the model detects possible bleeding occurrences in newly attached EHR notes. Bleeding occurrences described in new notes might change the status of the patient and indicate a de-escalation or escalation of the prophylaxis treatment.

The CDSS should present the EHR notes that it has predicted containing bleeding occurrences. They could be presented on a 3-month timeline that the doctor could scroll through to evaluate the notes suggested as bleeding occurrence candidates for relevancy. Sentences that have triggered a positive

prediction by the model is marked in red so the doctor can pay extra attention to those.

13.4.3 Keeping the model up to date

It is important to keep the CDSS as up to date on e.g. new bleeding disorders as the doctors who use the system. Continuous access to new EHR notes would help keeping the knowledge base up to date because the model can be improved by e.g. adding new words to the vocabulary and re-training the weights. Continuous access to data is also important to allow the model to evaluate new EHR notes in the background so that the CDSS is up to date on new data when the doctor needs it.

It was shown in section 13.3.2 that the model will most likely benefit from more data. A CDSS presents a good way of continuously providing new expert-annotated data for improving the model. If the doctor is asked to mark the sentences that he or she found to be relevant bleeding occurrences, this data could be included in the dataset. In this way, the model can continuously be retrained to be better at detecting bleedings and start to suggest only relevant bleedings to the doctor.

13.5 Chapter summary

This chapter discussed the methods and results of the thesis followed by how the model can be improved and implemented. The analysis of results was split in the evaluation of the model on the balanced test set and a real-life EHR.

It was reasoned that with a 89.2 % accuracy on the balanced test set, and a 94.2 % accuracy specifically on random negative samples which represent most of the sentences in an EHR, the large number of sentences in an EHR would pose a problem for the model.

Analyzing the performance on classifying notes of a real-life EHR, it was seen that per the sensitivity of 97.6 % and negative predictive value of 99.0 % that the model finds almost all positive cases and discards very few of them as negative. The precision of 38.3 % means, though, that the model misclassifies many negatives as positives.

In the analysis of the model, the inner workings of the convolutional neural network were examined using the Grad-CAM method that visualizes which

words the model considers important for its classification. Some of the patterns found was that when making a positive prediction, the first conv-layer of size 1 only pays attention to obvious bleeding-indicating words after which the subsequent conv-layer extends its attention to also look for other words. When making a negative prediction, the first conv-layer is looking broadly after important words, while the subsequent conv-layer tries to reduce the number of relevant words.

Furthermore, it was found that the model is not robust to spelling mistakes. This could be mitigated by using e.g. subword embedding models as fastText. A small analysis did not show the model to have any gender or age bias.

Next, it was discussed how improving the annotations, increasing the amount of data, and using other models and word embeddings could possibly improve the model performance.

Finally, it was discussed to implement the model in a clinical decision support system where a doctor can evaluate the returned positive predictions.

Chapter 14

14 Future work

This chapter presents the suggested future work on first the thesis objective of extracting bleeding occurrences from the free text of electronic health records (EHR), and then of the overall research objective of supporting the decision of thromboprophylaxis treatment by automatically detecting the risk of venous thromboembolism (VTE) and bleeding.

14.1 Extraction of bleeding occurrences

Section 13.3 presented four ways of improving the model performance on bleeding extraction.

It was shown that increasing the amount of data most likely will lead to better performance of the model. It would also be beneficial to increase the quality of new data through annotations. Before starting further annotation work, a dedicated annotation software could be either developed or acquired. It will ease the work of the annotators and prevent mistakes like unclosed tags. Furthermore, a more standardized annotation style of only annotating sentences and not full text passages indicating bleeding should be emphasized. It should also be discussed how to deal with ambiguous cases such as sentences indicating suspicion of a bleeding occurrence.

Because of time constraints, it was not possible to try adding an attention layer to the recurrent neural networks. It would be interesting to compare the performance of the models after an addition like that. Other models like BERT should be tried to see if it can outperform the current best convolutional neural network model.

It was noted in section 13.2.4 that the model is not robust to spelling mistakes. Implementing a subword based model like fastText would likely decrease the effect of spelling mistakes and should be explored in the future. Furthermore, regarding word embeddings, it was shown that it should be strived to increase the quality of the word embeddings in general, e.g. by training them on more EHR data.

Lastly, it would be interesting to further examine the bias of the finished model as it is important when implementing artificial intelligence. Especially within the health care system it is important not to introduce any bias into decisions.

14.2 Supporting decisions on thromboprophylaxis

This section outlines the further work needed to be done to reach the overall research objective of supporting decisions on thromboprophylaxis.

The current model detects bleeding occurrences and leaves the question of relevancy to the doctor via a Clinical Decision Support System (CDSS). While this setup can decrease the workload of the doctor, it would be beneficial to decrease the workload further by having the model filter out bleeding occurrences that are not relevant. This added feature of the model demands a new dataset that differentiates the relevant bleedings from irrelevant bleedings. If the current model is implemented as a CDSS, it would be possible to gather new data from the sentences that the doctor marks as relevant.

When a high-accuracy and minimum-doctor-workload process has been achieved, the other biomarkers needed for the IMPROVE bleeding risk score must be extracted.

A new model must then be trained to extract e.g. the occurrence of previous VTE, along with several other biomarkers from the EHR, for the ImPACT-ILL and Khorana VTE risk scores.

Finally, the scores of the three bleeding and VTE risk scoring systems must be input to the RADS decision support table to decide the type of thromboprophylaxis that should be given.

All steps mentioned above should be integrated in the EHR system as a CDSS, where the doctor is mainly used to evaluate the proposed relevant text passages that might include a relevant bleeding within 3 months prior of admission or previous VTE. With time and continuously gathered and improved data, the model should be trained to filter the results better to minimize the workload of the doctor.

Chapter 15

15 Conclusion

This thesis explored if bleeding occurrences from unstructured text of electronic health records (EHR) can be extracted using deep learning. The collected dataset consisted of 11,786 samples, annotated by 12 doctors with an inter annotator agreement of 0.75.

In a balanced test dataset of 1178 sentences from EHRs, the developed ensemble convolutional neural network (CNN) achieved an accuracy of 89.2%. Moreover, the ensemble CNN was tested on a note-level using an EHR consisting of 208 notes, achieving a sensitivity of 0.976 and a specificity of 0.620. These results suggest that it is possible to identify bleeding occurrences in Danish unstructured text from EHRs, and that is a possibility to implement the model as a clinical decision support system (CDSS). Further improvements should be conducted before implementation though, e.g. by improving the size and quality of the dataset.

The CNN was found to be superior to a recurrent neural network for the classification problem after a thorough hyperparameter and architecture search which also explored a combination of the two.

Moreover, the thesis showed examples of how the model can visualize its classifications decision of EHR notes using either its sentence level classification or the Grad-CAM method to create heatmaps. In a future prospect, this is promising for the model if it should be used in a CDSS as the doctor can obtain an understanding of the inner workings of the CNN, reinforcing the cooperation between the doctor and the system.

The above findings are part of a bigger research objective with the purpose of supporting the decision of thromboprophylaxis treatment by automatically detecting the risk of venous thromboembolism (VTE) and bleeding for hospitalized patients. This is currently a time consuming and error prone task, e.g. because EHRs should be inspected manually. Future work includes differentiating between relevant and none-relevant bleedings, as well as extraction of occurrences of VTE and other biomarkers from EHRs.

Appendix A: Explanation of the derivation of GloVe

The following explanation of the formulas which GloVe builds upon describes how Pennington et al. incorporated probability ratios and dimensions of meaning in the loss function. They define that

$$\exp\left((u_i - v_j)^T c_k\right) = \frac{P_{i,k}}{P_{j,k}} \quad (\text{A.1})$$

where u_i is the i 'th word embedding of the target matrix u , v_j is the j 'th word embedding of the target matrix v , and c_k is the k 'th word embedding of the context matrix c used to discriminate the target words. $u_i - v_j$ is performed to enforce the vector arithmetic of the analogies and encode dimensions of meaning. The result is transposed and multiplied by c_k to give a scalar that can be compared to the probability ratio of the right-hand-side of the equation.

Rearranging equation (A.1), the following equation is reached

$$\exp\left((u_i - v_j)^T c_k\right) = \exp(u_i^T c_k - v_j^T c_k) = \frac{\exp(u_i^T c_k)}{\exp(v_j^T c_k)} = \frac{P_{i,k}}{P_{j,k}} \quad (\text{A.2})$$

Pennington et al. then assume that [94]

$$\frac{\exp(u_i^T c_k)}{\exp(v_j^T c_k)} = \frac{P_{i,k}}{P_{j,k}} \rightarrow \exp(u_i^T c_k) = P_{i,k} \quad (\text{A.3})$$

and further state that

$$\begin{aligned} \exp(u_i^T c_k) = P_{i,k} &= \frac{X_{i,k}}{\sum_m X_{i,m}} \rightarrow u_i^T c_k \\ &= \log(X_{i,k}) - \log\left(\sum_m X_{i,m}\right) \end{aligned} \quad (\text{A.4})$$

Because the co-occurrence matrix is symmetric so that target and context word can be interchanged, this must also be the case in the final loss function. Notice that the free interchangeability of target and context words is not

satisfied for equation (A.4) because of the term $\log(\sum_m X_{i,m})$ which would yield another result if the sum over the k'th row was taken instead of the i'th. As the term is independent of k, it is made a bias, b_i , for u_i , and another bias, b_k , is added for c_k to keep the exchange symmetry:

$$u_i^T c_k + b_i + b_k - \log(X_{i,k}) = 0 \quad (\text{A.5})$$

A least squares loss function that sums over all pairs of words in the vocabulary is then defined:

$$L = \sum_{i,k=1}^V f(X_{i,k}) (u_i^T c_k + b_i + b_k - \log(X_{i,k}))^2 \quad (\text{A.6})$$

where V is the size of the vocabulary, u_i is the i'th word embedding of the target matrix u , c_k is the k'th word embedding of the context matrix c , b_i and b_k are biases, and $X_{i,k}$ is the number of times that the k'th word co-occurs with the i'th word. The function f is a weighting function that makes sure that when $X_{i,k} = 0$ then $f(X_{i,k}) = 0$ which means that the loss for that pair will not be calculated:

$$f(x) = \begin{cases} (x/x_{max})^{\frac{3}{4}}, & x < x_{max} \\ 1, & x \geq x_{max} \end{cases} \quad (\text{A.7})$$

Here, x is the evaluation of $X_{i,k}$ and x_{max} is the cutoff, which is set to 100 in the original paper. The weighting function eliminates the risk of getting an undefined result from $\log(X_{i,k}) = \log(0) = NaN$ and improves the training speed because training is now only on non-zero values.

Lastly, the weighting function also weights less the co-occurrences that are very infrequent and therefore noisy, as well as the very frequent ones like *it is* that should not dominate the loss over less occurring pairs [46]. The function is plotted on Figure A.1.

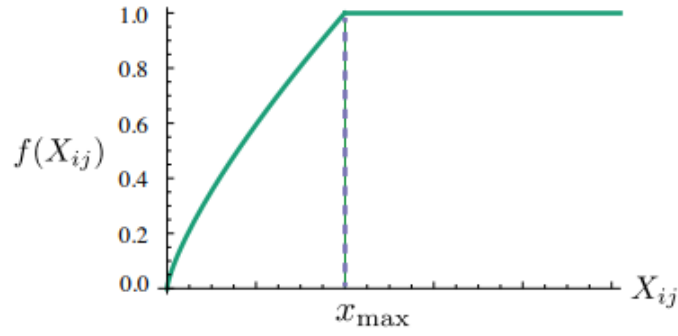


Figure A.1: Weighting function for GloVe loss function [38]

When the weight matrices, \mathbf{u} and \mathbf{v} , have been optimized to give the lowest loss, they are added to give the final word embeddings.

Summarizing, the training objective of GloVe is to go through all pairs of co-occurring words, minimizing the distance between the dot product of the two word embeddings and the log of their co-occurrence count. Because the difference of logarithms equals the logarithm of a ratio, word embedding differences are associated with co-occurrence probability ratios [45]. These probability ratios are connected to the meaning of words as exemplified earlier.

Appendix B: Differentiation of cross entropy loss with softmax

The cross entropy loss coupled with the softmax function is

$$L = -\log\left(\frac{e^{z_i}}{\sum_{j=1}^c e^{z_j}}\right) \quad (B.1)$$

where \mathbf{z} is the score vector and z_i is the score of the true class. The denominator sums the scores of all classes.

The derivate with respect to z_j is calculated using the chain rule:

$$\frac{\partial L}{\partial z_j} = \frac{-1}{\frac{e^{z_i}}{\sum_{j=1}^c e^{z_j}}} \cdot \frac{\partial}{\partial z_j} \frac{e^{z_i}}{\sum_{j=1}^c e^{z_j}} \quad (B.2)$$

Using the quotient rule the expression becomes:

$$\frac{\partial L}{\partial z_j} = \frac{-\sum_{j=1}^c e^{z_j}}{e^{z_i}} \cdot \frac{\frac{\partial}{\partial z_j} e^{z_i} \cdot \sum_{j=1}^c e^{z_j} - \frac{\partial}{\partial z_j} \sum_{j=1}^c e^{z_j} \cdot e^{z_i}}{\left(\sum_{j=1}^c e^{z_j}\right)^2} \quad (B.3)$$

There exists two different cases of the function. One where $i = j$ and one where $i \neq j$.

Case $i = j$

The derivative is calculated for when $i = j$:

$$\frac{\partial L}{\partial z_j} = \frac{-\sum_{j=1}^c e^{z_j}}{e^{z_i}} \cdot \frac{e^{z_i} \cdot \sum_{j=1}^c e^{z_j} - e^{z_j} \cdot e^{z_i}}{\left(\sum_{j=1}^c e^{z_j}\right)^2} \quad (B.4)$$

Notice that $\frac{\partial}{\partial z_j} \sum_{j=1}^c e^{z_j}$ evaluates to e^{z_j} because

$$\nabla(e^{z_1} + e^{z_2} + \dots) = \begin{bmatrix} e^{z_1} \\ e^{z_2} \\ \dots \end{bmatrix} \quad (B.5)$$

Rearranging gives

$$\begin{aligned} \frac{\partial L}{\partial z_j} &= \frac{-\sum_{j=1}^c e^{z_j}}{e^{z_i}} \cdot \frac{e^{z_i} \cdot \sum_{j=1}^c e^{z_j} - e^{z_j} \cdot e^{z_i}}{\left(\sum_{j=1}^c e^{z_j}\right)^2} \\ &= \frac{-\sum_{j=1}^c e^{z_j} \cdot e^{z_i} \cdot \sum_{j=1}^c e^{z_j} + \sum_{j=1}^c e^{z_j} \cdot e^{z_j} \cdot e^{z_i}}{e^{z_i} \cdot \left(\sum_{j=1}^c e^{z_j}\right)^2} \\ &= \frac{-\left(\sum_{j=1}^c e^{z_j}\right)^2 + \sum_{j=1}^c e^{z_j} \cdot e^{z_j}}{\left(\sum_{j=1}^c e^{z_j}\right)^2} \\ &= \frac{-\sum_{j=1}^c e^{z_j} + e^{z_j}}{\sum_{j=1}^c e^{z_j}} = \frac{e^{z_j}}{\sum_{j=1}^c e^{z_j}} - 1 \end{aligned} \quad (B.6)$$

Notice that the result when $i = j$ is the original softmax layer output minus 1.

Case $i \neq j$

Now, the derivative is calculated for when $i \neq j$:

$$\frac{\partial L}{\partial z_j} = \frac{-\sum_{j=1}^c e^{z_j}}{e^{z_i}} \cdot \frac{\frac{\partial}{\partial z_j} e^{z_i} \cdot \sum_{j=1}^c e^{z_j} - \frac{\partial}{\partial z_j} \sum_{j=1}^c e^{z_j} \cdot e^{z_i}}{\left(\sum_{j=1}^c e^{z_j}\right)^2} \quad (B.7)$$

$$\frac{\partial L}{\partial z_j} = \frac{-\sum_{j=1}^c e^{z_j}}{e^{z_i}} \cdot \frac{-e^{z_j} \cdot e^{z_i}}{\left(\sum_{j=1}^c e^{z_j}\right)^2} \quad (B.8)$$

Notice that $\frac{\partial}{\partial z_j} e^{z_i}$ evaluates to 0 because $i \neq j$.

Rearranging gives

$$\frac{\partial L}{\partial z_j} = \frac{-\sum_{j=1}^c e^{z_j}}{e^{z_i}} \cdot \frac{-e^{z_j} \cdot e^{z_i}}{\left(\sum_{j=1}^c e^{z_j}\right)^2} = \frac{\sum_{j=1}^c e^{z_j} \cdot e^{z_j}}{\left(\sum_{j=1}^c e^{z_j}\right)^2} = \frac{e^{z_j}}{\sum_{j=1}^c e^{z_j}} \quad (B.9)$$

Notice that the result when $i \neq j$ is the original softmax layer output.

Finally, from the two cases, it can be seen that the derivate of the cross entropy loss written in vector form is

$$\frac{\partial L}{\partial \mathbf{z}} = \hat{\mathbf{y}} - \mathbf{y} \tag{B.10}$$

where $\hat{\mathbf{y}}$ is the output from the softmax layer, i.e. the predicted probability distribution, and \mathbf{y} is the true probability distribution of the classes in the form of a one-hot vector. This corresponds to setting the derivate of the loss to the softmax layer output and subtracting 1 from the index where $i = j$, i.e. the index of the true class.

Appendix C: Transposing h_t

It can be seen from the following calculations why h_T needs to be transposed. The example is with $W^s \in \mathbb{R}^{2 \times 3}$ and $h_T \in \mathbb{R}^{3 \times 1}$.

The derivative of the loss with respect to W^s is

$$\begin{aligned} \frac{\partial L}{\partial W^s} &= \begin{bmatrix} \frac{\partial L}{\partial W_{1,1}^s} & \frac{\partial L}{\partial W_{1,2}^s} & \frac{\partial L}{\partial W_{1,3}^s} \\ \frac{\partial L}{\partial W_{2,1}^s} & \frac{\partial L}{\partial W_{2,2}^s} & \frac{\partial L}{\partial W_{2,3}^s} \end{bmatrix} \\ &= \begin{bmatrix} \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{1,1}^s} & \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{1,2}^s} & \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{1,3}^s} \\ \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{2,1}^s} & \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{2,2}^s} & \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{2,3}^s} \end{bmatrix} \end{aligned} \quad (C.1)$$

The derivative of the loss with respect to z is already known and defined as dz :

$$\frac{\partial L}{\partial z} = \hat{y} - y = \begin{bmatrix} dz_1 \\ dz_2 \end{bmatrix} = dz \quad (C.2)$$

Remembering that

$$\begin{aligned} z = \begin{bmatrix} z_1 \\ z_2 \end{bmatrix} &= W^s \cdot h_T = \begin{bmatrix} W_{1,1}^s & W_{1,2}^s & W_{1,3}^s \\ W_{2,1}^s & W_{2,2}^s & W_{2,3}^s \end{bmatrix} \begin{bmatrix} h_{T1} \\ h_{T2} \\ h_{T3} \end{bmatrix} \\ &= \begin{bmatrix} W_{1,1}^s \cdot h_{T1} + W_{1,2}^s \cdot h_{T2} + W_{1,3}^s \cdot h_{T3} \\ W_{2,1}^s \cdot h_{T1} + W_{2,2}^s \cdot h_{T2} + W_{2,3}^s \cdot h_{T3} \end{bmatrix} \end{aligned} \quad (C.3)$$

The second local gradient of the six applications of the chain rule in (C.1) are evaluated:

$$\frac{\partial z}{\partial W_{1,1}^s} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{1,1}^s} \\ \frac{\partial z_2}{\partial W_{1,1}^s} \end{bmatrix} = \begin{bmatrix} h_{T1} \\ 0 \end{bmatrix} \quad (C.4)$$

$$\frac{\partial z}{\partial W_{1,2}^s} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{1,2}^s} \\ \frac{\partial z_2}{\partial W_{1,2}^s} \end{bmatrix} = \begin{bmatrix} h_{T_2} \\ 0 \end{bmatrix} \quad (C.5)$$

$$\frac{\partial z}{\partial W_{1,3}^s} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{1,3}^s} \\ \frac{\partial z_2}{\partial W_{1,3}^s} \end{bmatrix} = \begin{bmatrix} h_{T_3} \\ 0 \end{bmatrix} \quad (C.6)$$

$$\frac{\partial z}{\partial W_{2,1}^s} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{2,1}^s} \\ \frac{\partial z_2}{\partial W_{2,1}^s} \end{bmatrix} = \begin{bmatrix} 0 \\ h_{T_1} \end{bmatrix} \quad (C.7)$$

$$\frac{\partial z}{\partial W_{2,2}^s} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{2,2}^s} \\ \frac{\partial z_2}{\partial W_{2,2}^s} \end{bmatrix} = \begin{bmatrix} 0 \\ h_{T_2} \end{bmatrix} \quad (C.8)$$

$$\frac{\partial z}{\partial W_{2,3}^s} = \begin{bmatrix} \frac{\partial z_1}{\partial W_{2,3}^s} \\ \frac{\partial z_2}{\partial W_{2,3}^s} \end{bmatrix} = \begin{bmatrix} 0 \\ h_{T_3} \end{bmatrix} \quad (C.9)$$

These are substituted into the expression for the derivative of the loss with respect to W^s :

$$\begin{aligned}
\frac{\partial L}{\partial W^s} &= \begin{bmatrix} \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{1,1}^s} & \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{1,2}^s} & \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{1,3}^s} \\ \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{2,1}^s} & \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{2,2}^s} & \frac{\partial L}{\partial z} \frac{\partial z}{\partial W_{2,3}^s} \end{bmatrix} \\
&= \begin{bmatrix} \begin{bmatrix} dz_1 \\ dz_2 \end{bmatrix} \cdot \begin{bmatrix} h_{T_1} \\ 0 \end{bmatrix} & \begin{bmatrix} dz_1 \\ dz_2 \end{bmatrix} \cdot \begin{bmatrix} h_{T_2} \\ 0 \end{bmatrix} & \begin{bmatrix} dz_1 \\ dz_2 \end{bmatrix} \cdot \begin{bmatrix} h_{T_3} \\ 0 \end{bmatrix} \\ \begin{bmatrix} dz_1 \\ dz_2 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ h_{T_1} \end{bmatrix} & \begin{bmatrix} dz_1 \\ dz_2 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ h_{T_2} \end{bmatrix} & \begin{bmatrix} dz_1 \\ dz_2 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ h_{T_3} \end{bmatrix} \end{bmatrix} \quad (C.10) \\
&= \begin{bmatrix} dz_1 \cdot h_{T_1} & dz_1 \cdot h_{T_2} & dz_1 \cdot h_{T_3} \\ dz_2 \cdot h_{T_1} & dz_2 \cdot h_{T_2} & dz_2 \cdot h_{T_3} \end{bmatrix} \\
&= \begin{bmatrix} dz_1 \\ dz_2 \end{bmatrix} \cdot \begin{bmatrix} h_{T_1} \\ h_{T_2} \\ h_{T_3} \end{bmatrix}^T = dz \cdot h_T^T = (\hat{y} - y) \cdot h_T^T
\end{aligned}$$

It can finally be seen that the loss with respect to W^s is $(\hat{y} - y)$ multiplied by the transposed hidden state output. While this was an example for specific dimensions of the network parameters, the result holds in general and supports the transposes and order of multiplication used in the further text [95].

Appendix D: Math of upper bounded terms of RNN

In theory, the backpropagation algorithm should update the weight matrices W^h , and W^x for all inputs to the RNN so that dependencies across long text passages are stored. The importance of learning long-term dependencies is shown in the example sentence

‘The test was negative for the patient having an abdominal bleeding’

For the RNN to make the correct classification that the patient does not have a bleeding, it must relate the word *negative* to the word *bleeding*. This means that the backpropagation algorithm must go through 8 timesteps of the hidden layer to update this dependency. In practice, updates based on early timesteps do not always take place. This is caused by the vanishing or exploding gradient problem.

In section 6.4.2 it was seen that the matrix multiplications that have to be made to find the gradient of the loss with respect to the weight W^h is given by.

$$\frac{\partial L}{\partial W^h} = \sum_{k=1}^T \frac{\partial L}{\partial h_T} \frac{\partial h_T}{\partial h_k} \frac{\partial h_k}{\partial W^h} \quad (\text{D.1})$$

The term $\frac{\partial h_T}{\partial h_k}$ is responsible for the backpropagation through $T - k$ timesteps of the hidden layer. Therefore, the derivative of the loss can be written as

$$\frac{\partial L}{\partial W^h} = \sum_{k=1}^T \frac{\partial L}{\partial h_T} \cdot \prod_{j=k+1}^T \frac{\partial h_j}{\partial h_{j-1}} \cdot \frac{\partial h_k}{\partial W^h} \quad (\text{D.2})$$

Expanding the term $\frac{\partial h_j}{\partial h_{j-1}}$ gives

$$\frac{\partial h_j}{\partial h_{j-1}} = \frac{\partial h_j}{\partial q_{j-1}} \cdot \frac{\partial q_{j-1}}{\partial h_{j-1}} \quad (\text{D.0.1})$$

where h_j is the output from the activation function to which q_{j-1} is the input:

$$h_j = \tanh(q_{j-1}) \rightarrow q_{j-1} = W^h h_{j-1} + W^x x_j \quad (\text{D.0.2})$$

Focusing on the input and output vectors it can be seen that

$$q_{j-1} \in \mathbb{R}^{D_h \times 1} \text{ is a vector } [q_{j-1_1}, q_{j-1_2}, \dots, q_{j-1_{D_h}}] \quad (\text{D.0.3})$$

$$h_j \in \mathbb{R}^{D_h \times 1} \text{ is a vector } [\tanh(q_{j-1_1}), \tanh(q_{j-1_2}), \dots, \tanh(q_{j-1_{D_h}})] \quad (\text{D.0.4})$$

Looking at the first term of (D.0.1), the derivative of h_j with respect to q_{j-1} is

$$\frac{\partial h_j}{\partial q_{j-1}} = \begin{bmatrix} \frac{\partial h_{j_1}}{\partial q_{j-1_1}} & \dots & \frac{\partial h_{j_1}}{\partial q_{j-1_{D_h}}} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_{j_{D_h}}}{\partial q_{j-1_1}} & \dots & \frac{\partial h_{j_{D_h}}}{\partial q_{j-1_{D_h}}} \end{bmatrix} \quad (\text{D.0.5})$$

It can be seen from (D.0.4) that the derivative of h_{jx} with respect to anything else than q_{j-1_x} is 0. For this reason, the Jacobian matrix $\frac{\partial h_j}{\partial q_{j-1}}$ is a matrix of zeroes with the derivative $\tanh'(q_{j-1})$ in the diagonal. It can be written as

$$\frac{\partial h_j}{\partial q_{j-1}} = \text{diag}(\tanh'(q_{j-1})) \quad (\text{D.0.6})$$

Now for the second term of (D.0.1), $\frac{\partial q_{j-1}}{\partial h_{j-1}}$, it can be seen from (D.0.2) that the derivative is W^h .

The magnitude of the $\frac{\partial h_j}{\partial h_{j-1}}$ term, which is multiplied iteratively, is interesting because it can either make the backpropagated gradient vanish or explode. The norm of the term is taken to find the magnitude [96]:

$$\begin{aligned}
\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| &= \left\| \text{diag} \left(\tanh'(q_{j-1}) \right) \cdot W^h \right\| \rightarrow \\
\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| &\leq \left\| \text{diag} \left(\tanh'(q_{j-1}) \right) \right\| \cdot \|W^h\|
\end{aligned} \tag{D.0.7}$$

Defining the upper bounds of the norms of the right side β_h and β_W , respectively, the following is reached:

$$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\| \leq \beta_h \cdot \beta_W \tag{D.0.8}$$

$\left\| \frac{\partial h_j}{\partial h_{j-1}} \right\|$ is thereby upper bounded by $\beta_h \cdot \beta_W$. β_h is bounded by the derivative of the activation function. In the case of a tanh activation function the upper bound of the derivate is 1. β_W is bounded by the weight matrix.

Finally, it is shown that the term $\frac{\partial h_T}{\partial h_k}$, which is responsible for the back-propagation through $T - k$ timesteps of the hidden layer, is upper bounded:

$$\begin{aligned}
\left\| \frac{\partial h_T}{\partial h_k} \right\| &= \left\| \prod_{j=k+1}^T \frac{\partial h_j}{\partial h_{j-1}} \right\| \rightarrow \\
\left\| \frac{\partial h_T}{\partial h_k} \right\| &\leq \prod_{j=k+1}^T \beta_h \cdot \beta_W \rightarrow \\
\left\| \frac{\partial h_T}{\partial h_k} \right\| &\leq (\beta_h \cdot \beta_W)^{T-k}
\end{aligned} \tag{D.0.9}$$

It can be seen from the expression that if $\beta_h \cdot \beta_W < 1$ the gradient will gradually vanish and that if $\beta_h \cdot \beta_W > 1$ the gradient can explode exponentially. If the gradient vanishes there will be small or even no updates and therefore no learning in the early layers. If the gradient explodes, the updates will become too large and there is a risk of getting overflow (NaN) which causes the training to crash [52], [61].

Appendix E: ICD-10 codes of patient groups

Trombose 100 cases Group 1	DI81	portal trombose
	DI82	Andre venøse tromboser og embolier
	DI80	phlebitis og thrombophlebitis
	DI26	lungeemboli
	DG08	sinus trombose
Interne blødninger 100 cases Group 2	DK661	Blødning i peritoneum
	DM250	Hæmartrose
	DK838F	Blødning i galdegang
	DK762	Central hæmorrhagisk levernekrose
	DK858B	Akut hæmorrhagisk pankreatitis
	DK868G	Pankreasblødning UNS
	DS260	Læsion af hjertet med hæmoperikardium
	DT144A	Hæmatomyeli UNS
	DE078B	Blødning i skjoldbruskkirtlen
	DE274B	Blødning i binyre
	DG361	Akut eller subakut hæmorrhagisk leukoencefalitis
	DI230	Hæmoperikardium efter akut myokardieinfarkt
	DI288A	Ruptur af lungekar
	DI312	Hæmoperikardium ikke klassificeret andetsteds
	DN830A	Blødning i follikulær ovariecyste
	DN831B	Blødning i corpus luteum-cyste
	DN836	Hæmatosalpinx
	DN837	Hæmatom i parametriet
	DN838D	Retrouterint hæmatocele
	DN838E	Ovariehæmatom
	DN838F	Ovarieblødning
	DN857	Hæmatometra
	DN897	Hæmatokolpos
	DJ942	Hæmothorax
	DJ942A	Hæmopneumothorax
	DJ950C	Blødning fra trakeostomistoma
	DK089A	Blødning fra processus alveolaris UNS
	DG951A	Blødning i rygmarven
Andet 100 cases Group 3	DS601A	Subungvalt hæmatom på finger
	DS902A	Subungvalt hæmatom på tå
	DT140D	Hæmatom UNS
	DT140K	Subungvalt hæmatom UNS
	DR233	Spontane ekkymoser
	DR233A	Spontane petekkier
	DN908D	Hæmatom i vulva
	DD500	Kronisk blødningsanæmi
	DD62	Akut anæmi efter blødning
	DN488D	Haematoma corporis cavernosi penis
	DN488E	Haematoma penis

	DN501A	Haematocele testis
	DN501B	Haematoma funiculi spermatici
	DN501C	Haematoma scroti
	DN501D	Haematoma testis
	DN501E	Haematoma tunicae vaginalis
	DN421	Prostatablødning
Cerebralt 100 cases Group 4	DI60	Subaraknoidalblødning
	DI61	Hjerneblødning
	DI62	Andre ikke-traumatiske intrakranielle blødninger
	DI620	Akut ikke-traumatisk subdural blødning
	DI621	Ikke-traumatisk epidural blødning
	DI629	Ikke-traumatisk intrakraniel blødning UNS
Mave-tarm 100 cases Group 5	DK250	Akut mavesår med blødning
	DK251	Akut mavesår med perforation
	DK252	Akut mavesår med blødning og perforation
	DK254	Kronisk eller ikke specificeret mavesår med blødning
	DK256	Kronisk eller ikke specificeret mavesår med blødning og perforation
	DK260	Akut duodenalulcus med blødning
	DK262	Akut duodenalulcus med blødning og perforation
	DK270	Akut gastroduodenalt ulcus med blødning
	DK272	Akut gastroduodenalt ulcus med blødning og perforation
	DK274	Kronisk eller ikke specificeret gastroduodenalt ulcus med blødning
	DK276	Kronisk eller ikke specificeret gastroduodenalt ulcus med blødning og perforation
	DK280	Akut gastrointestinalt sår med blødning
	DK282	Akut gastrointestinalt sår med blødning og perforation
	DK284	Kronisk eller ikke specificeret gastrointestinalt sår med blødning
	DK286	Kronisk eller ikke specificeret gastrointestinalt sår med blødning og perforation
	DK290	Akut hæmoragisk gastritis
	DK625	Blødning fra anus eller rectum
	DK638B	Tarmblødning UNS
	DK638C	Blødning fra mavetarmkanalen UNS
	DK766B	Portal hypertensiv gastropati med blødning
	DK920	Hæmatemese
	DK921	Melæna
	DI841C	Interne hæmorroider med blødning
	DI844C	Eksterne hæmorroider med blødning
	DI848C	Hæmorroider UNS med blødning
	DI850	Øsofagusvaricer med blødning
	DI859	Øsofagusvaricer uden blødning
	DI864A	Varicer i mavesækken med blødning

	DK228F	Blødning i spiserøret UNS
	DK922	Gastrointestinal blødning UNS
Urinveje mfl 100 cases Group 6	DN02	Tilbagevendende og vedvarende blod i urinen
	DN508X	Hæmospermi
	DN308B	Hæmorrhagisk cystitis
	DR31	Hæmaturi UNS
Fødsel/genitalia 100 cases Group 7	DN93	Anden abnorm blødning fra livmoderen og vagina
	DN930	Livmoderblødning efter coitus eller kontaktblødning
	DN938	Anden form for abnorm blødning fra livmoderen eller vagina
	DN939	Abnorm blødning fra livmoderen eller vagina UNS
	DO031	Inkomplet spontan abort kompliceret med sen eller excessiv blødning
	DO036	Komplet eller ikke specificeret spontan abort kompliceret med sen eller excessiv blødning
	DO081	Sen eller excessiv blødning efter abort, ektopisk graviditet eller mola
	DO081G	Blødning efter abort
	DO20	Blødning tidligt i svangerskabet
	DO208	Anden blødning før 22 svangerskabsuger
	DO208A	Accidentel blødning tidligt i svangerskabet
	DO208B	Blødning ved fibrinolyse tidligt i svangerskabet
	DO208C	Blødning ved koagulationsdefekt tidligt i svangerskabet
	DO209	Blødning tidligt i svangerskabet UNS
	DO441	Forliggende moderkage med blødning
	DO441A	Totalt forliggende moderkage med blødning
	DO441B	Partielt forliggende moderkage med blødning
	DO46	Vaginalblødning før fødsel IKA
	DO460	Blødning før fødsel med koagulationsdefekt
	DO468	Anden form for blødning før fødsel
	DO468A	Blødning i graviditet efter 22 svangerskabsuger
	DO469	Blødning før fødsel UNS
	DO67	Fødsel kompliceret af blødning IKA
	DO678	Fødsel kompliceret af anden form for blødning
	DO679	Fødsel kompliceret af blødning UNS
	DO72	Blødning i efterbyrdsperioden
	DO720	Blødning efter fødsel
	DO722	Sen blødning efter fødsel
	DP021	Anden blødning med følger for nyfødt
Øre-næse-hals + luftveje 100 cases Group 8	DR04	Blødning fra luftveje
	DR040	Næseblod
	DR041	Blødning fra svælg
	DR042	Hæmoptyse
	DR048	Blødning fra anden lokalisation i luftveje
	DR049	Blødning fra luftvejene UNS
	DS003A	Hæmatom i næseskillevæggen

	DH922	Blødning fra øre
Øjne 100 cases Group 9	DH052	Eksoftalmi forårsaget af blødning i øjenhule
	DH113	Blødning i conjunctiva
	DH210	Hyphaema
	DH180A	Haematocornea
	DH313	Blødning eller ruptur i årehinden
	DH356	Retinal blødning
	DH431	Blødning i øjets glaslegeme
	DH448	Hæmophthalmos
	DH450	Blødning i øjets glaslegeme ved sygdom klassificeret andetsteds
	DB303C	Akut hæmoragisk konjunktivitis UNS
Leukæmi 100 cases Group 10		

Table E.1: ICD-10 codes of patient groups

Appendix F: Annotation guide

Formål

- Positive samples: Annotering af tekstpassagerer der indikerer en hver form for blødning, værende det medicinsk blødning eller andre typer
- Interessante negative samples: Annotering af tekstpassagerer der nævner blødning uden at der er en, fx "Patienten har ikke en blødning" el. "Patienten er blevet undersøgt pga. mistanke om blødning"

Regler

- Markér hele sætningen eller sammenhængende sætninger der indikerer en blødning.
- I tilfælde af at sætninger ved siden af hinanden indikerer den samme blødning, skal de markeres som en samlet passage.

Appendix G: Code for inserting HTML tags in annotated EHRs

Functions to clean and find tag insert positions

```

1. #Takes a text, removes ._, <...>, and characters that might can be misinter-
   #preted as a regex command
2. #Returns cleaned text and a list of removed text, index 0: position, in-
   #dex 1: text
3. def clean_save_index(text):
4.     InsertBack=[] #In relation to raw HTML
5.     cleanout=[[0,0]]
6.     cleanout_sub=[[m.start(),m.end()] for m in re.finditer('\.|\s|\\(|\\)|\\?|\\
   +|\\,|<.|.|.|.|.', text)]
7.     for sub in cleanout_sub:
8.         cleanout.append(sub)
9.         InsertBack.append([sub[0],text[sub[0]:sub[1]]])
10.    clean=''
11.    for s in range(1,len(cleanout)):
12.        clean+=text[cleanout[s-1][1]:cleanout[s][0]]
13.    clean+=text[cleanout[-1][1]:]
14.
15.    #Convert indexes to insert back from raw HTML to clean HTML
16.    shift=0
17.    for i in range(len(InsertBack)):
18.        InsertBack[i][0]-=shift
19.        shift+=len(InsertBack[i][1])
20.
21.    return clean,InsertBack #InsertBack in relation to clean HTML
22.
23. #Cleans everything out, including annotation tags
24. def clean_all(text):
25.     cleanout=[[0,0]]
26.     cleanout_sub=[[m.start(),m.end()] for m in re.finditer('\.|\s|\\(|\\)|\\?|\\
   +|\\,|<.|.|.|.|.|##TVIVL##|##start_bleeding##|##stop_bleeding##|##start_nega-
   tive##|##stop_negative##', text)]
27.     for sub in cleanout_sub:
28.         cleanout.append(sub)
29.     clean_text=''
30.     for s in range(1,len(cleanout)):
31.         clean_text+=text[cleanout[s-1][1]:cleanout[s][0]]
32.     clean_text+=text[cleanout[-1][1]:]
33.     return clean_text
34.
35. #Takes clean annotated and clean HTML text, tag_type: True=bleed-
   #ing, False=negative
36. #Returns a list of tags to be inserted, index 0: position, in-
   #dex 1: text, and 1 if the tag order doesn't align
37. def tagsPosition(Annotated_clean, HTML_clean, tag_type):
38.     if tag_type==True:
39.         tag='bleeding'
40.     if tag_type==False:
41.         tag='negative'
42.     #Find position of start and end of negative tags
43.     start_regex='##start_'+tag+'##'
44.     stop_regex='##stop_'+tag+'##'
45.     start=[[m.end(), 'start'] for m in re.finditer(start_regex, Anno-
   #tated_clean)] #Find start of annotated sentences
46.     stop=[[m.start(), 'stop'] for m in re.finditer(stop_regex, Anno-
   #tated_clean)] #Find end of annotated sentences

```

```

47.
48.     tag_placement=[]
49.
50.     if len(start)==0 and len(stop)==0: #If there are no annota-
        tion tags. AND check because maybe one of the tags has a miss-
        ing # and we would like to print it
51.         return tag_placement, 0
52.
53.     #Check the order of the start and stop tags
54.     order=sorted(start+stop, key=lambda x: x[0])
55.     last_was_start=0 #Start as if last was false be-
        cause first must be a start
56.     for tag in order:
57.         if tag[1]=='start':
58.             if last_was_start==1:
59.                 print('2 ##start_# in a row:\n')
60.                 return tag_placement, -1
61.             last_was_start=1
62.         elif tag[1]=='stop':
63.             if last_was_start==0:
64.                 print('2 ##stop_## in a row:\n')
65.                 return tag_placement, -1
66.             last_was_start=0
67.
68.
69.     #Find the position in HTML where to insert tags from Annotated
70.     not_all_found=0
71.     for start_index,stop_index in zip(start,stop):
72.         Annotated=Annotated_clean[start_index[0]:stop_index[0]] #Se-
        lect the annotated sentence from the clean annotated text
73.         subsample=[[m.start(),m.end()] for m in re.finditer(Anno-
        tated, HTML_clean)] #Search for sentence in be-
        tween start and stop tag in the clean HTML text
74.         if len(subsample)==0:
75.             print('NOT FOUND: '+Annotated)
76.             not_all_found=1
77.         else:
78.             if len(subsample)>1:
79.                 print('FOUND > 1: '+Annotated)
80.                 for sub in subsample:
81.                     tag_placement.append([sub[0],start_regex])
82.                     tag_placement.append([sub[1],stop_regex])
83.
84.     return tag_placement, not_all_found #tag_placement in rela-
        tion to clean HTML

```

Loading and Preprocessing

```

1. import re
2. import docx
3. from docx import Document
4.
5. # Open documents
6. doc_annotated = docx.Document(r"C:\Users\marti_000\Desktop\Speciale_for-
    budt online\EXCEL MED HTML\Klar\VTE2_2 - del nr. 4.docx")
7. doc_HTML = docx.Document(r"C:\Users\marti_000\Desktop\Speciale_forbudt on-
    line\EXCEL MED HTML\Klar\VTE2_2 - del nr. 4_raw_excel.xlsx.docx")
8. doc_HTML_with_tags=Document()
9.
10. # Read each paragraph of files
11. paragraphs_annotated = [p.text for p in doc_annotated.paragraphs]
12. paragraphs_HTML = [p.text for p in doc_HTML.paragraphs]

```

```

13. print('Initial # of paragraphs_annotated: ',len(paragraphs_annotated))
14. print('Initial # of paragraphs_HTML: ',len(paragraphs_HTML))
15.
16. #Remove annotated para-
    graphs which are not one of the four: newid, noteid, broedtekst, count
17. print('\nRemoving annotated paragraphs which are not one of the follow-
    ing: newid, noteid, broedtekst, count')
18. p=0
19. while p < len(paragraphs_annotated):
20.     if not(paragraphs_annotated[p][:5]=='newid' or paragraphs_anno-
        tated[p][:14]=='JournalNote_ID' or paragraphs_anno-
        tated[p][:10]=='Broedtekst' or paragraphs_annotated[p][:7]=='count_x'):
21.         print('Deleted:',paragraphs_annotated[p])
22.         paragraphs_annotated.pop(p)
23.     else:
24.         p+=1
25.
26. #Remove HTML para-
    graphs which are not one of the four: newid, noteid, broedtekst, count
27. print('Removing HTML paragraphs which are not one of the follow-
    ing: newid, noteid, broedtekst, count')
28. p=0
29. while p < len(paragraphs_HTML):
30.     if not(paragraphs_HTML[p][:5]=='newid' or para-
        graphs_HTML[p][:14]=='JournalNote_ID' or para-
        graphs_HTML[p][:10]=='Broedtekst' or paragraphs_HTML[p][:7]=='count_x'):
31.         print('Deleted: ',paragraphs_HTML[p])
32.         paragraphs_HTML.pop(p)
33.     else:
34.         p+=1
35.
36. print('\n# of paragraphs_annotated after removal:',len(paragraphs_anno-
    tated))
37. print('# of paragraphs_HTML after removal: ',len(paragraphs_HTML))
38.
39. #Remove annotated empty broedtekst
40. print('\nRemoving annotated empty broedtekt notes')
41. p=0
42. while p < len(paragraphs_annotated):
43.     if paragraphs_annotated[p][:10]=='Broedtekst' and len(paragraphs_anno-
        tated[p])<12:
44.         print('Deleted:',paragraphs_annotated[p-2], paragraphs_annotated[p-
            1], paragraphs_annotated[p], paragraphs_annotated[p+1])
45.         paragraphs_annotated.pop(p+1)
46.         paragraphs_annotated.pop(p)
47.         paragraphs_annotated.pop(p-1)
48.         paragraphs_annotated.pop(p-2)
49.         p-=2
50.     else:
51.         p+=1
52.
53. #Remove empty broedtekst
54. print('\nRemoving HTML empty broedtekt notes')
55. p=0
56. while p < len(paragraphs_HTML):
57.     if paragraphs_HTML[p][:10]=='Broedtekst' and len(para-
        graphs_HTML[p])<12:
58.         print('Deleted: ',paragraphs_HTML[p-2], paragraphs_HTML[p-1], para-
            graphs_HTML[p], paragraphs_HTML[p+1])
59.         paragraphs_HTML.pop(p+1)
60.         paragraphs_HTML.pop(p)
61.         paragraphs_HTML.pop(p-1)
62.         paragraphs_HTML.pop(p-2)

```

```

63.         p-=2
64.     else:
65.         p+=1
66.
67.
68. #Preprocess the paragraphs to align
69. print('\nPreprocessing the paragraphs to align and pushing anno-
    tated not fitting any HTML to the back')
70. i=0
71. while i<len(paragraphs_HTML):
72.     #Check if paragraph has open tag
73.     open_tag=False
74.     for j in range(1,len(paragraphs_HTML[i+2])):
75.         #print(paragraphs_HTML[i+2][-10:])
76.         if paragraphs_HTML[i+2][-j]==>':
77.             break
78.         if paragraphs_HTML[i+2][-j]==<':
79.             open_tag=True
80.             break
81.     if open_tag: #insert what was eaten
82.         paragraphs_annotated.insert(i+3,paragraphs_annotated[i+2])
83.         paragraphs_annotated.insert(i+3,paragraphs_HTML[i+5])
84.         paragraphs_annotated.insert(i+3,paragraphs_HTML[i+4])
85.         paragraphs_annotated.insert(i+3,paragraphs_HTML[i+3])
86.         print('Open tag correction at',paragraphs_HTML[i+1])
87.         clean_HTML=clean_all(paragraphs_HTML[i+2])
88.         clean_annotated=clean_all(paragraphs_annotated[i+2])
89.         if not(paragraphs_HTML[i][:7]==paragraphs_annotated[i][:7] and para-
    graphs_HTML[i+1][:12]==paragraphs_anno-
    tated[i+1][:12] and (clean_HTML[:30]==clean_annotated[:30] or clean_HTML[-
    30:]==clean_annotated[-30:])):
90.             print('Matching:',paragraphs_HTML[i+1])
91.             print(paragraphs_HTML[i],paragraphs_annotated[i])
92.             print(paragraphs_HTML[i+1],paragraphs_annotated[i+1])
93.             print(clean_HTML[:30],clean_annotated[:30])
94.             print(clean_HTML[-30:],clean_annotated[-30:])
95.
96.             paragraphs_annotated.append(paragraphs_annotated[i])
97.             paragraphs_annotated.append(paragraphs_annotated[i+1])
98.             paragraphs_annotated.append(paragraphs_annotated[i+2])
99.             paragraphs_annotated.append(paragraphs_annotated[i+3])
100.             paragraphs_annotated.pop(i)
101.             paragraphs_annotated.pop(i)
102.             paragraphs_annotated.pop(i)
103.             paragraphs_annotated.pop(i)
104.         else:
105.             print('Matched',paragraphs_HTML[i],paragraphs_annotated[i],para-
    graphs_HTML[i+1],paragraphs_annotated[i+1])
106.             i+=4
107.
108. #Check if all HTML paragraphs has a match
109. print('\nChecking if all HTML paragraphs has a match')
110. print(len(paragraphs_HTML))
111. print(len(paragraphs_annotated))
112. for r in range(0,len(paragraphs_HTML),4):
113.     if paragraphs_HTML[r+1][:12]!=paragraphs_annotated[r+1][:12]:
114.         print('No match for',paragraphs_HTML[r+1][:12],paragraphs_anno-
    tated[r+1][:12])
115.     print('Done aligning paragraphs\n')
116.
117.     print('\nDeleting excess annotation paragraphs')
118.     for i in reversed(range(len(paragraphs_HTML),len(paragraphs_anno-
    tated),4)):

```

```

119.     print('Deleted:', paragraphs_annotated[i+1], '\t', paragraphs_anno-
120.         tated[i+2][:50])
121.     paragraphs_annotated.pop(i)
122.     paragraphs_annotated.pop(i)
123.     paragraphs_annotated.pop(i)
124.     print('\n# of annotated paragraphs after preprocessing:', len(para-
125.         graphs_annotated))
126.     print('# of HTML paragraphs after preprocessing:', len(para-
127.         graphs_HTML))
128.     for i in range(len(paragraphs_HTML)):
129.         if paragraphs_HTML[i][0:10]=='Broedtekst':
130.
131.             #Clean the annotated and HTML text and save list of the positi-
132.             ons and text deleted
133.             Annotated_clean, Annotated_getBack = clean_save_index(para-
134.                 graphs_annotated[i])
135.             HTML_clean, HTML_getBack = clean_save_index(para-
136.                 graphs_HTML[i])
137.
138.             #Save list of the position and tags to be inserted
139.             bleeding_tags, error_bleeding=tagsPosition(Anno-
140.                 tated_clean, HTML_clean, True) #True indicates check for bleeding tag
141.             negative_tags, error_negative=tagsPosition(Anno-
142.                 tated_clean, HTML_clean, False) #False indicates check for negative tag
143.
144.             if error_bleeding==1 or error_negative==1: #if ei-
145.                 ther start/stop doesn't align or one annotated sentence could-
146.                 n't be found in HTML
147.                 print('Error! Two start/stop in a row - inserted HTML with-
148.                     out annotations')
149.                 print('newid: ', paragraphs_HTML[i-2], ' noteid: ', para-
150.                     graphs_HTML[i-1])
151.                 print('Annotated:\n', paragraphs_annotated[i])
152.                 print('HTML:\n', paragraphs_HTML[i], '\n')
153.                 doc_HTML_with_tags.add_paragraph(paragraphs_HTML[i])
154.             else:
155.                 if error_bleeding==1 or error_negative==1:
156.                     print('Warning! One annota-
157.                         tion was not found in the HTML text. Can be because of copied para-
158.                         graphs.')
159.                 print('newid: ', paragraphs_HTML[i-2], ' noteid: ', para-
160.                     graphs_HTML[i-1])
161.                 print('Annotated:\n', paragraphs_annotated[i])
162.                 print('HTML:\n', paragraphs_HTML[i], '\n')
163.                 #Sort tuples using first element of sublist
164.                 insert_to_HTML = sorted(HTML_getBack+bleeding_tags+nega-
165.                     tive_tags, key=lambda x: x[0])
166.                 insert_to_HTML.insert(0, [0, ''])
167.
168.                 #Make sure that start tags are placed after same-in-
169.                 dex html tags and stop tags before same-index HTML tags
170.                 for p in range(0, len(insert_to_HTML)-1):
171.                     if insert_to_HTML[p][0]==in-
172.                         sert_to_HTML[p+1][0]: #Two of the same indexes in a row
173.                         if insert_to_HTML[p][1][8]=='##start_':
174.                             insert_to_HTML.insert(p+2, in-
175.                                 sert_to_HTML[p]) #move right
176.                             insert_to_HTML.pop(p)
177.                 insert_to_HTML.reverse()

```

```

163.         for p in range(0,len(insert_to_HTML)-1):
164.             if insert_to_HTML[p][0]==in-
sert_to_HTML[p+1][0]: #Two of the same indexes in a row
165.                 if insert_to_HTML[p][1][:7]=='##stop_':
166.                     insert_to_HTML.insert(p+2,in-
sert_to_HTML[p]) #move right
167.                     insert_to_HTML.pop(p)
168.             insert_to_HTML.reverse()
169.
170.
171.             #Insert everything back into HTML
172.             HTML_with_tags=''
173.             for s in range(1,len(insert_to_HTML)):
174.                 HTML_with_tags+=HTML_clean[insert_to_HTML[s-1][0]:in-
sert_to_HTML[s][0]]+insert_to_HTML[s][1]
175.                 HTML_with_tags+=HTML_clean[insert_to_HTML[-
1][0]:] #The rest of text after last annotation
176.
177.                 doc_HTML_with_tags.add_paragraph(HTML_with_tags)
178.
179.             else: #if paragrph is not a broedtekst
180.                 doc_HTML_with_tags.add_paragraph(paragraphs_HTML[i])
181.             print('Done')
182.             doc_HTML_with_tags.save('C:\\Users\\marti_000\\Desktop\\Speciale_for-
budt online\\EXCEL MED HTML\\Klar\\VTE2_2 - del nr. 4_HTML.docx')

```

Appendix H: Test of Stanza and NLTK tokenizer

Table H.1 shows the original text and how the Stanza and NLTK tokenizers have split the sentences. A ‘ || ’ tag has been placed in the original text where the manual sentence split occurs around a positive or negative sample marked by the annotators. In the tokenized text, a ‘ || ’ tag has been placed where the sentence split was made by the tokenizer. There are no manual splits for *Abbreviations*. Green marks tokenized sentences similar to the manual. Yellow marks a sentence where the tokenizers has included more of the text passage in the sentence. Red marks a sentence where the tokenizer has included less than in the manual sentence tokenization.

Manual	Stanza	NLTK
Summary-style		
<p>Indlæggelsesårsag. Respirationsinsufficiens. Resumé. Kendt med hjertesvigt med nærnormal EF. Kronisk let nyreinsufficiens. RA. Hyp. art. Tidl cerebralt infarkt.. </p> <p>GI-blødninger x flere (tyndtarms hæmangiomer).. </p> <p>April 2015 pulmonalt ødem på baggrund af type 2 infarkt.. August 2015 pneumoni og pulmonal stase, udskrevet 19/8-</p>	<p>Indlæggelsesårsag. Respirationsinsufficiens. Resumé. </p> <p>Kendt med hjertesvigt med nærnormal EF. </p> <p>Kronisk let nyreinsufficiens. RA. Hyp. art. </p> <p>Tidl cerebralt infarkt.. </p> <p>GI-blødninger x flere (tyndtarms hæmangiomer).. </p> <p>April 2015 pulmonalt ødem på baggrund af type 2 infarkt.. </p>	<p>Indlæggelsesårsag. Respirationsinsufficiens. </p> <p>Resumé. </p> <p>Kendt med hjertesvigt med nærnormal EF. </p> <p>Kronisk let nyreinsufficiens. </p> <p>RA. </p> <p>Hyp. </p> <p>art. </p> <p>Tidl cerebralt infarkt.. GI-blødninger x flere (tyndtarms hæmangiomer).. </p> <p>April 2015 pulmonalt ødem på</p>

15 (med penicillin t.o.m. 27/8)..	August 2015 pneumoni og pulmonal stase, udskrevet 19/8-15 (med penicillin t.o.m. 27/8)..	baggrund af type 2 infarkt.. August 2015 pneumoni og pulmonal stase, udskrevet 19/8-15 (med penicillin t.o.m. 27/8)..
Klinisk kontakt. Henvisningsdiagnose. Kronisk eller ikke specificeret mavesår med blødning(DK254). Henvisningsdato. 2014-08-31 00:00:00. Henvisningsmåde. Henvist fra sygehusafsnit(F). Henvisningstekst. Ingen henvisning.	Klinisk kontakt. Henvisningsdiagnose. Kronisk eller ikke specificeret mavesår med blødning(DK254). Henvisningsdato. 2014-08-31 00:00:00. Henvisningsmåde. Henvist fra sygehusafsnit(F). Henvisningstekst. Ingen henvisning.	Klinisk kontakt. Henvisningsdiagnose. Kronisk eller ikke specificeret mavesår med blødning(DK254). Henvisningsdato. 2014-08-31 00:00:00. Henvisningsmåde. Henvist fra sygehusafsnit(F). Henvisningstekst. Ingen henvisning.
Klinisk kontakt. Henvisningsdiagnose. Mikroskopisk hæmaturi UNS(DR319B). . Henvisningsdato. 2014-12-09 00:00:00. Henvisningsmåde. Alment praktiserende læge(1).	Klinisk kontakt. Henvisningsdiagnose. Mikroskopisk hæmaturi UNS(DR319B).. Henvisningsdato. 2014-12-09 00:00:00. Henvisningsmåde. Alment praktiserende læge(1).	Klinisk kontakt. Henvisningsdiagnose. Mikroskopisk hæmaturi UNS(DR319B).. Henvisningsdato. 2014-12-09 00:00:00. Henvisningsmåde. Alment praktiserende læge(1).

<p>Henvisningstekst. hæmaturi + 2 blod stix gange 3 siden start november (in- tet i foråret). Aldrig makrosko- pisk, % røg, % eksp..</p>	<p>Henvisningstekst. hæmaturi + 2 blod stix gange 3 siden start november (intet i foråret). Aldrig makrosko- pisk, % røg, % eksp..</p>	<p>Henvisningstekst. hæmaturi + 2 blod stix gange 3 siden start november (intet i foråret). Aldrig makrosko- pisk, % røg, % eksp..</p>
<p>Behandlingsplan. Konklusion og plan. AMD-AMBULATO- RIET. Patienten har d.d. været til kontrol. Se Well-dokument.. Foto OD: Uændret cicatrise, hæm svun- det. OCT OD: Fortsat ødem. Indikation for, rp Eylea x 1 OD. Kon- trol ca 6 uger efter..</p>	<p>Behandlingsplan. Konklusion og plan. AMD-AMBULATO- RIET. Patienten har d.d. været til kontrol. Se Well-dokument.. Foto OD: Uændret cicatrise, hæm svun- det. OCT OD: Fortsat ødem. Indikation for, rp Eylea x 1 OD. Kontrol ca 6 uger efter..</p>	<p>Behandlingsplan. Konklusion og plan. AMD-AMBULATO- RIET. Patienten har d.d. været til kontrol. Se Well-dokument.. Foto OD: Uændret cicatrise, hæm svun- det. OCT OD: Fortsat ødem. Indikation for, rp Eylea x 1 OD. Kontrol ca 6 uger ef- ter..</p>
<p>Pt. skal ringes op mandag morgen med tid mandag forudgået af visus, dil. og Optos o.u.. Diagnoser. Akti- onsdiagnose: DH431. Blødning i øjets</p>	<p>Pt. skal ringes op mandag morgen med tid mandag forudgået af visus, dil. og Optos o.u.. Diagnoser. Aktionsdiagnose: DH431. </p>	<p>Pt. skal ringes op mandag morgen med tid mandag forudgået af visus, dil. og Optos o.u.. Diagnoser. Aktionsdiagnose:</p>

<p>glaslegeme </p> <p>?: TUL1. højresidig.</p> <p>Procedurer. "Dokumentreference - kan ikke vises her". ?:</p> <p>ZZ0150A. Anamneseoptagelse. Skal sendes.</p> <p>Ja. Afsendelse behandlet.</p>	<p>Blødning i øjets</p> <p>glaslegeme. </p> <p>?: TUL1. højresidig.</p> <p> </p> <p>Procedurer. </p> <p>"Dokumentreference - kan ikke vises her". </p> <p>?: ZZ0150A. Anamneseoptagelse. </p> <p>Skal sendes. </p> <p>Ja. </p> <p>Afsendelse behandlet.</p>	<p>DH431. </p> <p>Blødning i øjets</p> <p>glaslegeme. </p> <p>? </p> <p>: TUL1. </p> <p>højresidig. </p> <p>Procedurer. </p> <p>"Dokumentreference - kan ikke vises her". </p> <p>? </p> <p>: ZZ0150A. </p> <p>Anamneseoptagelse.</p> <p> </p> <p>Skal sendes. </p> <p>Ja. </p> <p>Afsendelse behandlet.</p>
Abbreviations (marked in red if wrongly split on abbreviation)		
<p>BT: 137/65 P: 93</p> <p>SAT: 99% RF: 16</p> <p>GCS: 15 (er orienteret x tre, men virker hukommelsessvækket).</p> <p>Pt. er engangskath. for 250 ml. let konc. urin..</p> <p>Urinstix: er ikke sendt.</p>	<p>BT: 137/65 P: 93</p> <p>SAT: 99% </p> <p>RF: 16 GCS: 15 (er orienteret x tre, men virker hukommelsessvækket). </p> <p>Pt. er engangskath. for 250 ml. let konc. urin..</p> <p> Urinstix: er ikke sendt.</p>	<p>BT: 137/65 P: 93</p> <p>SAT: 99% RF: 16</p> <p>GCS: 15 (er orienteret x tre, men virker hukommelsessvækket). </p> <p>Pt. er engangskath. </p> <p>for 250 ml. </p> <p>let konc. </p> <p>urin.. Urinstix: er ikke sendt. </p>
<p>Ingen mavesmerter eller kvalme. Der er faldende carbamid til</p>	<p>Ingen mavesmerter eller kvalme. </p> <p>Der er faldende</p>	<p>Ingen mavesmerter eller kvalme. </p> <p>Der er faldende</p>

<p>11,1, stigende Hb til 6,9. cont. PPI-infusion. rp. biokemi inkl. Hb til aften. Der er svar på Hp-test, som er negativ. Pt. får ASA, som kan være årsag til ulcus. Vi genoptager. rp. Fragmin 12.500 til aften x 1 dagligt.</p>	<p>carbamid til 11,1, stigende Hb til 6,9. cont. PPI-infusion. rp. biokemi inkl. Hb til aften. Der er svar på Hp-test, som er negativ. Pt. får ASA, som kan være årsag til ulcus. Vi genoptager. rp. Fragmin 12.500 til aften x 1 dagligt.</p>	<p>carbamid til 11,1, stigende Hb til 6,9. cont. PPI-infusion. rp. biokemi inkl. Hb til aften. Der er svar på Hp-test, som er negativ. Pt. får ASA, som kan være årsag til ulcus. Vi genoptager. rp. Fragmin 12.500 til aften x 1 dagligt.</p>
<p>Hud...: Problemer. Forb. mangler på hø. coll. fem. cikatrice til morgen.. Uvist om pt. selv har fået pillet det af.. Hud...: Udførte handlinger. Pga. vandladning i sengen er der gjort grundig, steril afvaskning..</p>	<p>Hud...: Problemer. Forb. mangler på hø. coll. fem. cikatrice til morgen.. Uvist om pt. selv har fået pillet det af. Hud...: Udførte handlinger. Pga. vandladning i sengen er der gjort grundig, steril afvaskning..</p>	<p>Hud...: Problemer. Forb. mangler på hø. coll. fem. cikatrice til morgen.. Uvist om pt. selv har fået pillet det af. Hud...: Udførte handlinger. Pga. vandladning i sengen er der gjort grundig, steril afvaskning..</p>

<p>Ved indl. i går aftes opstartet iv benzylpenicillin 2 MIE x 4 dgl. Rtg. af thorax beskrevet at vise bilat. infiltrater, flere tætte, der anbefales opfølgende kontrol rtg (pt. i 1996 beh. for c. mammae). Pt. angiver sig dårlig, stakåndet, har hovedpine.. Objektiv undersøgelse. Vågen, bevidsthedsklar men træt, påskyndet let rallende resp., målte værdier som ovenfor anført, BT 146 systolisk</p>	<p>Ved indl. i går aftes opstartet iv benzylpenicillin 2 MIE x 4 dgl. </p> <p>Rtg. af thorax beskrevet at vise bilat. infiltrater, flere tætte, der anbefales opfølgende kontrol rtg (pt. i 1996 beh. for c. mammae). </p> <p>Pt. angiver sig dårlig, stakåndet, har hovedpine.. </p> <p>Objektiv undersøgelse. </p> <p>Vågen, bevidsthedsklar men træt, påskyndet let rallende resp., målte værdier som ovenfor anført, BT 146 systolisk</p>	<p>Ved indl. </p> <p>i går aftes opstartet iv benzylpenicillin 2 MIE x 4 dgl. </p> <p>Rtg. </p> <p>af thorax beskrevet at vise bilat. </p> <p>infiltrater, flere tætte, der anbefales opfølgende kontrol rtg (pt. i 1996 beh. </p> <p>for c. mammae). </p> <p>Pt. angiver sig dårlig, stakåndet, har hovedpine.. </p> <p>Objektiv undersøgelse. </p> <p>Vågen, bevidsthedsklar men træt, påskyndet let rallende resp., målte værdier som ovenfor anført, BT 146 systolisk</p>
<p>Patienten får fjernet sep. CVK dags dato. Der lægges ekstra i.v. adgang så patienten har to i.v. adgange. Fortsætter med Tazocin og Ciprofloxacin i.v. Regner med at patienten kan gå over til peroral Ciprofloxacin</p>	<p>Patienten får fjernet sep. </p> <p>CVK dags dato. </p> <p>Der lægges ekstra i.v. adgang så patienten har to i.v. adgange. </p> <p>Fortsætter med Tazocin og Ciprofloxacin i.v. Regner med at</p>	<p>Patienten får fjernet sep. </p> <p>CVK dags dato. </p> <p>Der lægges ekstra i.v. </p> <p>adgang så patienten har to i.v. </p> <p>adgange. </p> <p>Fortsætter med Tazocin og Ciprofloxacin</p>

om nogle dage.	patienten kan gå over til peroral Ciprofloxacin om nogle dage.	i.v. Regner med at patienten kan gå over til peroral Ciprofloxacin om nogle dage.
Bleeding occurrences (doctors' decision on split marked in original with)		
<p>Smerterne ledsaget af kvalme og enkelte opkastninger. </p> <p>Pt. har angiveligt haft blod i urinen enkelte gange igennem det sidste døgn samt smerter ved vandladning. </p> <p>Angiver desuden også blod i afføringen igennem det sidste døgn. </p> <p>Fortæller, at han har haft afføring i går morges, som var normal i konsistensen..</p>	<p>Smerterne ledsaget af kvalme og enkelte opkastninger. </p> <p>Pt. har angiveligt haft blod i urinen enkelte gange igennem det sidste døgn samt smerter ved vandladning. </p> <p>Angiver desuden også blod i afføringen igennem det sidste døgn. </p> <p>Fortæller, at han har haft afføring i går morges, som var normal i konsistensen. </p> <p>.</p>	<p>Smerterne ledsaget af kvalme og enkelte opkastninger. </p> <p>Pt. har angiveligt haft blod i urinen enkelte gange igennem det sidste døgn samt smerter ved vandladning. </p> <p>Angiver desuden også blod i afføringen igennem det sidste døgn. </p> <p>Fortæller, at han har haft afføring i går morges, som var normal i konsistensen..</p>
<p>91-årig herre indlagt til pacemakerimplantation grundet 3. grads AV blok. </p> <p>Undertegnede tilkaldes grundet øget hæmatom samt at pt.</p>	<p>91-årig herre indlagt til pacemakerimplantation grundet 3. grads AV blok. </p> <p>Undertegnede tilkaldes grundet øget hæmatom samt at pt.</p>	<p>91-årig herre indlagt til pacemakerimplantation grundet 3. grads AV blok. </p> <p>Undertegnede tilkaldes grundet øget hæmatom samt at pt.</p>

klager over smerter i venstre arm .. Objektiv undersøgelse.	klager over smerter i venstre arm.. Objektiv undersøgelse.	klager over smerter i venstre arm.. Objektiv undersøgelse.
Konklusion og plan. Det drejer sig om et lille men meget smertevoldende hæmatom. . Det vil svinde spontant.. Pt har smertestillende..	Konklusion og plan. Det drejer sig om et lille men meget smertevoldende hæmatom.. Det vil svinde spontant.. Pt har smertestillende..	Konklusion og plan. Det drejer sig om et lille men meget smertevoldende hæmatom.. Det vil svinde spontant.. Pt har smertestillende..
Der er enkelte mørke kanter svarende til sutureringen i venstre lyske de øvrige sårkanter ser pæne ud. Der er serøst/blodig sivning. Der er podet fra alle sår..	Der er enkelte mørke kanter svarende til sutureringen i venstre lyske de øvrige sårkanter ser pæne ud. Der er serøst/blodig sivning. Der er podet fra alle sår..	Der er enkelte mørke kanter svarende til sutureringen i venstre lyske de øvrige sårkanter ser pæne ud. Der er serøst/blodig sivning. Der er podet fra alle sår..
Konklusion og plan. Corpushæmoragi efter Ozurdex-implantation o.dx. for 9 dage siden. Patienten angiver debut af sløret syn kort efter og har været	Konklusion og plan. Corpushæmoragi efter Ozurdex-implantation o.dx. for 9 dage siden. Patienten angiver debut af sløret syn kort efter og har været	Konklusion og plan. Corpushæmoragi efter Ozurdex-implantation o.dx. for 9 dage siden. Patienten angiver debut af sløret syn kort efter og har været

uændret lige siden.	uændret lige siden.	uændret lige siden.
---------------------	---------------------	---------------------

Table H.1: Sentence tokenization test between Stanza and NLTK. //: Split. Green: Similar to manual. Yellow: More text than manual. Red: Less text than manual.

Appendix I: Code for extracting sentence samples

Extract and de-identify annotated sentences

```

1. import xlwt
2. from xlwt import Workbook
3. import docx
4. import os
5. import re #regular expressions
6. import glob
7. #Named Entity Recognition
8. from flair.data import Sentence
9. from flair.models import SequenceTagger
10.
11. #Load Sequence Tagger
12. ner_tagger = SequenceTagger.load('da-ner')
13.
14. #Load docx
15. dir = os.path.dirname(os.path.abspath(__file__))
16. files = glob.glob(os.path.join(dir, "*.docx"))
17. print(files)
18.
19. # Excel Workbook is created and a sheet is added
20. wb = Workbook()
21. sheet1 = wb.add_sheet('Sheet 1')
22.
23. row=0 #Rows for excel sheet
24. p=0 #Positive samples
25. n=0 #Negative samples
26. corrupt_files=[]
27. read_files=[]
28. #For each word file in folder
29. for file in files:
30.     try:
31.         doc = docx.Document(file)
32.     except:
33.         print("\n",file," is corrupt\n")
34.         corrupt_files.append(file)
35.         break
36.     read_files.append(file)
37.     text=[]
38.
39.     #Append text from word file
40.     for i in doc.paragraphs:
41.         text.append(i.text)
42.
43.     #Look for positive samples
44.     for i in range(len(text)):
45.         positive = re.findall('##start_bleeding##(.*)##stop_bleed-
46.         ing##',text[i]) #Find annotations and append to positive list
47.         for j in range(len(positive)):
48.             trans-
49.             formed=re.sub(r'(^|\D)(,|\)|\(|\(|;|/|\|)(\D|$)', r'\1 \2 \3', posi-
50.             tive[j]) #Force space before and after some special charac-
51.             ters for NER to work
52.             sentence = Sentence(transformed) #Make sen-
53.             tence for flair NER tagger
54.             ner_tagger.predict(sentence) #Predict sentence

```

```

50.     entities=sentence.to_dict(tag_type='ner')[ 'entities' ] #Read en-
        tities dict from flair. Contains list of dicts of start and end char posi-
        tions of NER entities
51.     start_end_pos=[0] #Append 0 til start with first charac-
        ter of string
52.     for entity in range(len(entities)): #For each found NER entity
53.         dict=entities[entity] #Open dict of that entity
54.         if dict['type']=='PER': #Only look for person enti-
            ties, not organisation or place
55.             start_end_pos.append(dict['start_pos']) #Ap-
            pend start position
56.             start_end_pos.append(dict['end_pos']) #Append end posi-
            tion
57.             start_end_pos.append(len(transformed)) #Append posi-
            tion of last char in string
58.             anonymized=""
59.             for pos in range(int(len(start_end_pos)/2)):
60.                 anonymized += trans-
                    formed[start_end_pos[(pos*2)]:start_end_pos[(pos*2)+1]] #Anony-
                    mized is the chars not belonging to PER entity
61.                 if pos!=int(len(start_end_pos)/2)-1:
62.                     anonymized += 'NAME' #Add name be-
                        tween breaks as long as not final block of string
63.                     #PAS PÅ... DEN IKKE FJERNER 1/3 hvis det er en brÅ.k
64.                     #anonymi-
                        zed=re.sub(r'(\d\d|\d)(\.|:|/)(\d\d|\d)(\.|:|/)(\d\d\d\d(\s|$|\.)|\d\d(\s|$|
                        \.))',r'DAYMONTHYEAR_TIME ',anonymized) #Anonymize data
65.                     #anonymi-
                        zed = re.sub(r'(\d\d|\d)(\.|:|/)(\d\d(\s|$|\.)|\d(\s|$|\.))', r'DAY-
                        MONTH_TIME ',anonymized) # Anonymize data
66.                     sheet1.write(row,0,anonymized)
67.                     sheet1.write(row,1,'positive')
68.                     row+=1
69.                     print("Bleeding: ",transformed)
70.                     print("Bleeding: ", anonymized)
71.                     p+=1
72.
73.     #Look for negative samples (see comments for positive)
74.     for i in range(len(text)):
75.         negative = re.findall('##start_negative##(.*)##stop_neg-
            ative##',text[i])
76.         for j in range(len(negative)):
77.             trans-
                formed=re.sub(r'(^|\D)(,|\)|\(|:|;|/|\)|\D|$)', r'\1 \2 \3', negative[j])
78.             sentence = Sentence(transformed)
79.             ner_tagger.predict(sentence)
80.             entities=sentence.to_dict(tag_type='ner')[ 'entities' ]
81.             start_end_pos=[0]
82.             for entity in range(len(entities)):
83.                 dict=entities[entity]
84.                 if dict['type']=='PER':
85.                     start_end_pos.append(dict['start_pos'])
86.                     start_end_pos.append(dict['end_pos'])
87.             start_end_pos.append(len(transformed))
88.             anonymized=""
89.             for pos in range(int(len(start_end_pos)/2)):
90.                 anonymized += trans-
                    formed[start_end_pos[(pos*2)]:start_end_pos[(pos*2)+1]]
91.                 if pos != int(len(start_end_pos) / 2) - 1:
92.                     anonymized += 'NAME'
93.                 #PAS PÅ... DEN IKKE FJERNER 1/3 hvis det er en brÅ.k

```

```

94.         #anonymi-
zed=re.sub(r'(\d\d\d)(\.\.|\:|/)(\d\d\d)(\.\.|\:|/)(\d\d\d\d\d(\s|$|\.)|\d\d(\s|$|
\.)\.)',r'DAYMONTHYEAR_TIME ',anonymized) #Anonymize data
95.         #anonymi-
zed = re.sub(r'(\d\d\d)(\.\.|\:|/)(\d\d\d(\s|$|\.)|\d(\s|$|\.)\.)', r'DAY-
MONTH_TIME ',anonymized) # Anonymize data
96.         sheet1.write(row,0,anonymized)
97.         sheet1.write(row,1,'negative')
98.         row+=1
99.         print("Negative: ",transformed)
100.        print("Negative: ", anonymized)
101.        n+=1
102.    print("Positive samples: ",p,". Negative samples: ",n,". Total sam-
ples: ",p+n)
103.    print("\nCorrupt files:\n")
104.    print(corrupt_files)
105.    print("\nRead files:\n")
106.    print(read_files)
107.    wb.save('Samples.xls')

```

Extract and de-identify random sentences

```

1. import xlwt
2. import docx
3. import stanza
4. import random
5. from xlwt import Workbook
6. import os
7. import re #regular expressions
8. import glob
9. #Named Entity Recognition
10. from flair.data import Sentence
11. from flair.models import SequenceTagger
12. from pathlib import Path
13.
14. #Load sentence tokenizer
15. #stanza.download('da')
16. nlp = stanza.Pipeline(lang='da', processors='tokenize')
17.
18. #Load Sequence Tagger
19. ner_tagger = SequenceTagger.load('da-ner')
20.
21. # Excel Workbook is created and a sheet is added
22. wb = Workbook()
23. sheet1 = wb.add_sheet('Sheet 1')
24.
25. paths = [str(x) for x in Path(r"C:\Users\marti_000\Desktop\test\Extract ran-
dom").glob("**/*.docx")]#("./eo_data/").glob("**/*.txt")]
26. row=0 #Rows for excel sheet
27.
28. # broedtext_index=[] #Indeholder alle index med broedtekst
29. for path in paths:
30.     sentences_pr_doc = 0
31.     text = []
32.     broedtext_index = [] # Indeholder alle index med broedtekst
33.
34.     doc = docx.Document(path)
35.     for i in doc.paragraphs:
36.         text.append(i.text)
37.
38.     for i in range(0, len(text)):
39.         if text[i][0:10] == 'Broedtekst':

```

```

40.         if "##" not in text[i]: # sørg for pos og neg sætnin-
            ger ikke er med
41.             broedtext_index.append(i)
42.             random_note_index = random.sample(range(len(broed-
text_index)), 100) # Tag random sætning fra broedtekst
43.             for random_note in random_note_index: # take XX broedtexter random
44.                 doc = nlp(text[broedtext_index[random_note]])
45.                 broedtext_sentences = ([s.text for s in doc.sentences])
46.                 if len(broedtext_sentences) > 8:
47.                     random_sentence_index=random.sample(range(len(broedtext_senten-
ces)), 8) # Tag random sætning fra broedtekst
48.                     for random_sentence in random_sentence_index: # Tag 10 ran-
dom sætninger
49.                         trans-
formed = re.sub(r'(^|\D)(,|\)|\(|:|;|/|\|)(\D|$)', r'\1 \2 \3', broedtext_se-
ntences[random_sentence]) # Force space before and after some special char-
acters for NER to work
50.                         sentence = Sentence(transformed) # Make sen-
tence for flair NER tagger
51.                         ner_tagger.predict(sentence) # Predict sentence
52.                         entities = sentence.to_dict(tag_type='ner')['enti-
ties'] # Read entities dict from flair. Con-
tains list of dicts of start and end char positions of NER entities
53.                         start_end_pos = [0] # Append 0 til start with first charac-
ter of string
54.                         for entity in range(len(enti-
ties)): # For each found NER entity
55.                             dict = entities[entity] # Open dict of that entity
56.                             if dict['type'] == 'PER': # Only look for person enti-
ties, not organisation or place
57.                                 start_end_pos.append(dict['start_pos']) # Ap-
pend start position
58.                                 start_end_pos.append(dict['end_pos']) # Ap-
pend end position
59.                                 start_end_pos.append(len(transformed)) # Append posi-
tion of last char in string
60.                                 anonymized = ""
61.                                 for pos in range(int(len(start_end_pos) / 2)):
62.                                     anonymized += trans-
formed[start_end_pos[(pos * 2):start_end_pos[
63.                                         (pos * 2) + 1]] # Anonymized is the chars not be-
longing to PER entity
64.                                     if pos != int(len(start_end_pos) / 2) - 1:
65.                                         anonymized += 'NAME' # Add name be-
tween breaks as long as not final block of string
66.                                     sheet1.write(row, 0, anonymized)
67.                                     sheet1.write(row, 1, 'rand')
68.                                     row += 1
69.                                     sentences_pr_doc += 1
70.             else: # tag alle sætninger
71.                 for sent in broedtext_sentences:
72.                     trans-
formed = re.sub(r'(^|\D)(,|\)|\(|:|;|/|\|)(\D|$)', r'\1 \2 \3', sent) # For-
ce space before and after some special characters for NER to work
73.                     sentence = Sentence(transformed) # Make sen-
tence for flair NER tagger
74.                     ner_tagger.predict(sentence) # Predict sentence
75.                     entities = sentence.to_dict(tag_type='ner')[
76.                         'entities'] # Read entities dict from flair. Con-
tains list of dicts of start and end char positions of NER entities
77.                     start_end_pos = [0] # Append 0 til start with first charac-
ter of string

```

```

78.         for entity in range(len(entities)): # For each found NER entity
79.             dict = entities[entity] # Open dict of that entity
80.             if dict['type'] == 'PER': # Only look for person entities, not organisation or place
81.                 start_end_pos.append(dict['start_pos']) # Append start position
82.                 start_end_pos.append(dict['end_pos']) # Append end position
83.                 start_end_pos.append(len(transformed)) # Append position of last char in string
84.                 anonymized = ""
85.                 for pos in range(int(len(start_end_pos) / 2)):
86.                     anonymized += transformed[start_end_pos[(pos * 2)]:start_end_pos[(pos * 2) + 1]] # Anonymized is the chars not belonging to PER entity
87.                 if pos != int(len(start_end_pos) / 2) - 1:
88.                     anonymized += 'NAME' # Add name between breaks as long as not final block of string
89.                 sheet1.write(row, 0, anonymized)
90.                 sheet1.write(row, 1, 'rand')
91.                 row += 1
92.                 sentences_pr_doc += 1
93.                 print('Document {29} Number of sentences: {3d} Total sentences: {4d}'.format(path[32:], sentences_pr_doc,
94.
95.
96.                     row))
97.                 # print("Document: ", path[32:], "Number of sentences: ", sentences_pr_doc, "Total sentences", row)
98. wb.save('Samples.xls')

```

Appendix J: Code for extracting paragraph samples

Extract annotated paragraphs

```

1. #Create samples to the paragraph dataset
2.
3. #Load word word files:
4. from pathlib import Path
5. paths = [str(x) for x in Path(r"C:/Users/Jannik/Desktop/new_da-
   taset/").glob("**/*.docx")]
6.
7. #Extract all broedtekst
8. import docx
9. text=[]
10. for path in paths:
11.     doc = docx.Document(path)
12.     for i in doc.paragraphs:
13.         #print(i.text[0:10], "\n")
14.         if i.text[0:10]=='Broedtekst':
15.             print(i.text, "\n\n")
16.             text.append(i.text)
17.
18. ##Make samples
19. def new_data_tokenize(note): #Annotates and return samples from one broedtekst at a time
20.     #HTML tag cases:
21.     ## h1 * /h1 = append * to samples
22.     ## li * /li = -- if <li> foran </li>: append * to temp, ap-
   pend temp to samples, pop() temp -- | -- if </ul> foran </li> it is fi-
   nal string, append * to temp and append temp to samples -- | --else ap-
   pend *+':' to temp, ':' because it there is a sublist to the current list
23.     ## li * br/ = temp append *, append temp to samples, pop() temp
24.     ## br/ * END = append to temp, append temp to samples
25.     ## br/ * br/ = if * empty break (dont append anything). Else ap-
   pend * to temp, append temp to samples, pop() temp. If ko-
   lon in front of last br/: append to temp and ekstra_pop = true
26.     ## br/ * /li = append * to temp, append temp to sam-
   ples. If <li> in front of </li>: temp.pop() (if list point is fin-
   ished by making new point with <li> instead of the more normal proce-
   dure where first </ul> then <ul>)
27.     ## if /ul: temp.pop
28.
29.
30.     samples = []
31.     temp=[]
32.     ekstra_pop = False
33.     ekstra_pop_li = False
34.
35.     for i in range(len(note)):
36.         #####h1 * /h1 #####
37.         if note[i:i+4]=='<h1>':
38.             start = i+4
39.             for j in range(i+4, len(note)): #range(start, stop)
40.                 if note[j:j+5]=='</h1>':
41.                     temp.append(note[start:j]+':')
42.                     break;
43.
44.         ##### li * /li #####
45.         if note[i:i+4]=='<li>':
46.             start = i+4

```

```

47.         for j in range(i+4, len(note)):
48.             if note[j:j+5]=='</li>': # if /li foran - eksempel: <li>[j]
1. reservelæge </li>[j+5]
49.                 if note[j+5:j+5+4]=='<li>': ##if li foran /li. eksem-
pel: <li>[j] 1. reservelæge </li><li>[j+5+4] angiver at(..)
50.                     temp.append(note[start:j])
51.                     samples.append(temp[:])
52.                     temp.pop()
53.                     break;
54.
55.                 if note[j+5:j+5+5]=='</ul>': #if /ul foran li - Its not
the final string, append string without ':'
56.                     temp.append(note[start:j])
57.                     samples.append(temp[:])
58.                     #temp.pop()
59.                     break;
60.
61.             else: #Append string with ':' (der star-
tes ny list <ul>)
62.                 temp.append(note[start:j]+':')
63.                 break;
64.
65.             ##### li * br/ #####
66.             if note[j:j+5]=='<br/>':
67.                 if note[j-1:j+5]=='<br/>':
68.                     temp.append(note[start:j])
69.                     #print(note[start:j+20])
70.                     ekstra_pop_li = True
71.                     break;
72.                 temp.append(note[start:j])
73.                 samples.append(temp[:])
74.                 temp.pop()
75.                 if ekstra_pop_li == True:
76.                     temp.pop()
77.                     ekstra_pop_li = False
78.                 break;
79.
80.
81.             #####br/ * br/ #####
82.             if note[i:i+5]=='<br/>':
83.                 start = i+5
84.                 for j in range(i+5,len(note)+1):
85.                     if j==len(note): ##Hvis vi er i slutningen af noten
86.                         temp.append(note[start:j])
87.                         samples.append(temp[:])
88.
89.
90.                 if note[j:j+5]=='<br/>':
91.                     if start==j:
92.                         break;
93.                     if note[j-1:j+5]=='<br/>':
94.                         temp.append(note[start:j])
95.                         #print(note[start:j+20])
96.                         ekstra_pop = True
97.                         break;
98.                 temp.append(note[start:j])
99.                 samples.append(temp[:])
100.                 temp.pop()
101.                 if ekstra_pop == True:
102.                     temp.pop()
103.                     ekstra_pop = False
104.                 break;
105.

```

```

106.         #####br * /li #####
107.         if note[j:j+5]=='</li>':
108.             temp.append(note[start:j])
109.             samples.append(temp[:])
110.             if note[j+5:j+5+4]=='<li>':
111.                 temp.pop()
112.                 break;
113.
114.         ##### if /ul #####
115.         if note[i:i+5]=='</ul>':
116.             temp.pop()
117.
118.         #print("\ntemp",temp)
119.         return samples
120.
121.     samples = []
122.     for broedtekst in text: #Annotate one broedtekst at a time
123.         samples += new_data_tokenize(broedtekst)
124.     for i in range(10):
125.         print(samples[i])
126.
127.     #Make the samples as one string (instead of tokens)
128.     samples_final=[]
129.     for sample in samples:
130.         string=''
131.         for entry in sample:
132.             string+=entry+' '
133.         samples_final.append(string)
134.     for i in range(10000):
135.         print(samples_final[i])
136.
137.     #Extract negative and positive samples to word
138.     import xlwt
139.     from xlwt import Workbook
140.
141.     #Initialize excel document
142.     excel_name = 'New_dataset.xls'
143.     wb = Workbook()
144.     sheet1 = wb.add_sheet('Sheet 1')
145.     row=1
146.     sheet1.write(0, 0, 'Sample')
147.     sheet1.write(0,1, 'Label')
148.     positives_lost = []
149.     negative_lost = []
150.
151.     for sample in samples_final:
152.         ##Append positive samples
153.         if '##start_bleeding##' in sample and '##stop_bleed-
154.         ing##' not in sample:
155.             positives_lost.append(sample)
156.         elif '##stop_bleeding##' in sample and '##start_bleed-
157.         ing##' not in sample:
158.             positives_lost.append(sample)
159.         elif '##start_bleeding##' in sample and '##stop_bleeding##' in sam-
160.         ple:
161.             sheet1.write(row, 0, sample.replace('##start_bleed-
162.             ing##', ' ').replace('##stop_bleeding##', ' '))
163.             sheet1.write(row, 1, 'positive')
164.             sheet1.write(row, 2, sample)
165.             row+=1
166.             #print(samples_final, "\n")
167.
168.         ##Append negative samples

```



```

165.         elif '##start_negative##' in sample and '##stop_negati-
            tive##' not in sample:
166.             negative_lost.append(sample)
167.         elif '##stop_negative##' in sample and '##start_negati-
            tive##' not in sample:
168.             negative_lost.append(sample)
169.         elif '##start_negative##' in sample and '##stop_negative##' in sam-
            ple:
170.             sheet1.write(row, 0, sample.replace('##start_negati-
                tive##', ' ').replace('##stop_negative##', ' '))
171.             sheet1.write(row, 1, 'negative')
172.             sheet1.write(row, 2, sample)
173.             row+=1
174.     wb.save(excel_name)

```

Extract random paragraphs

```

1. import docx
2. import random
3. sentence_holder=[] #Indeholder random sætninger
4. for path in paths:
5.     sentences_pr_doc = 0
6.     text=[]
7.     broedtext_index=[] #Indeholder alle index med broedtekst
8.
9.     doc = docx.Document(path)
10.    for i in doc.paragraphs:
11.        text.append(i.text)
12.
13.    for i in range(0, len(text)):
14.        if text[i][0:10]=='Broedtekst':
15.            if "##" not in text[i]: #sørg for pos og neg sætning-
                er ikke er med
16.                broedtext_index.append(i)
17.
18.    random_broedtext_index = random.sample(range(len(broedtext_in-
                dex)), 100) #Chose index of xx random broedtexts
19.    for rand in random_broedtext_index: #take XX broedtexter random
20.        broedtext_sentences = new_data_tokenize(text[broedtext_in-
                dex[rand]]) #Create samples of that broedtext
21.        if len(broedtext_sentences)>4:
22.            rand_sen = random.sample(range(len(broedtext_sentences)), 4)
23.            for rand_in rand_sen: # Tag xx tilfældige sætninger
24.                sentence_holder.append(broedtext_sentences[rand_in])
25.                sentences_pr_doc+=1
26.        else: #tag alle sætninger
27.            for sentence in broedtext_sentences:
28.                sentence_holder.append(sentence)
29.                sentences_pr_doc+=1
30.        print('Document {29} Number of sentences: {3d} Total sen-
            tences: {4d}'.format(path[32:], sentences_pr_doc, len(sentence_holder)))
31.        #print("Document: ", path[32:], "Number of sentences: ", sen-
            tences_pr_doc, "Toal sentences", len(sentence_holder))

```

De-identify extracted paragraphs

```

1. #Excel

```

```

2. import xlrd #To read Excel csv file
3. from xlwt import Workbook
4. import os
5. import re #regular expressions
6. #Named Entity Recognition
7. from flair.data import Sentence
8. from flair.models import SequenceTagger
9.
10. # Open Excel file: old
11. wb_old = xlrd.open_workbook("New_dataset.xls")
12. sheet_old = wb_old.sheet_by_index(0)
13.
14. # Make Excel file: new
15. wb_new = Workbook()
16. sheet_new = wb_new.add_sheet('Sheet 1')
17.
18. #Load Sequence Tagger
19. ner_tagger = SequenceTagger.load('da-ner')
20.
21. for row in range(sheet_old.nrows):
22.     for col in range(sheet_old.ncols):
23.         if != 0:
24.             sheet_new.write(row,col,sheet_old.cell_value(row,col))
25.         else:
26.             paragraph=sheet_old.cell_value(row,col)
27.             trans-
28.             formed = re.sub(r'(^|\D)(,|\)|\(|\(|;|/|\|)(\D|$)', r'\1 \2 \3', paragraph)
29.             sentence = Sentence(transformed) # Make sen-
30.             tence for flair NER tagger
31.             ner_tagger.predict(sentence) # Predict sentence
32.             entities = sentence.to_dict(tag_type='ner')['enti-
33.             ties'] # Read entities dict from flair. Con-
34.             tains list of dicts of start and end char positions of NER entities
35.             start_end_pos = [0] # Append 0 til start with first charac-
36.             ter of string
37.             for entity in range(len(entities)): # For each found NER en-
38.             tity
39.                 dict = entities[entity] # Open dict of that entity
40.                 if dict['type'] == 'PER': # Only look for person enti-
41.                 ties, not organisation or place
42.                     start_end_pos.append(dict['start_pos']) # Ap-
43.                     pend start position
44.                     start_end_pos.append(dict['end_pos']) # Append end po-
45.                     sition
46.                     start_end_pos.append(len(transformed)) # Append posi-
47.                     tion of last char in string
48.                     anonymized = ""
49.                     for pos in range(int(len(start_end_pos) / 2)):
50.                         anonymized += trans-
51.                         formed[start_end_pos[(pos * 2)]:start_end_pos[
52.                         (pos * 2) + 1]] # Anonymized is the chars not belong-
53.                         ing to PER entity
54.                     if pos != int(len(start_end_pos) / 2) - 1:
55.                         anonymized += 'NAME' # Add name be-
56.                         tween breaks as long as not final block of string
57.                     sheet_new.write(row, col, anonymized)
58. wb_new.save('Samples anonymized.xls')

```

Appendix K: List of stop-words

The following words and signs were considered stop-words for sentence dataset analysis:

. , og i med er til på af der har at fra for en om : som men til den

Appendix L: Training of recurrent neural network

This appendix presents the choosing and training of a recurrent neural network model. Figure L.1 visualizes the structure of a general recurrent architecture and the hyperparameters that can be tuned to improve the model.

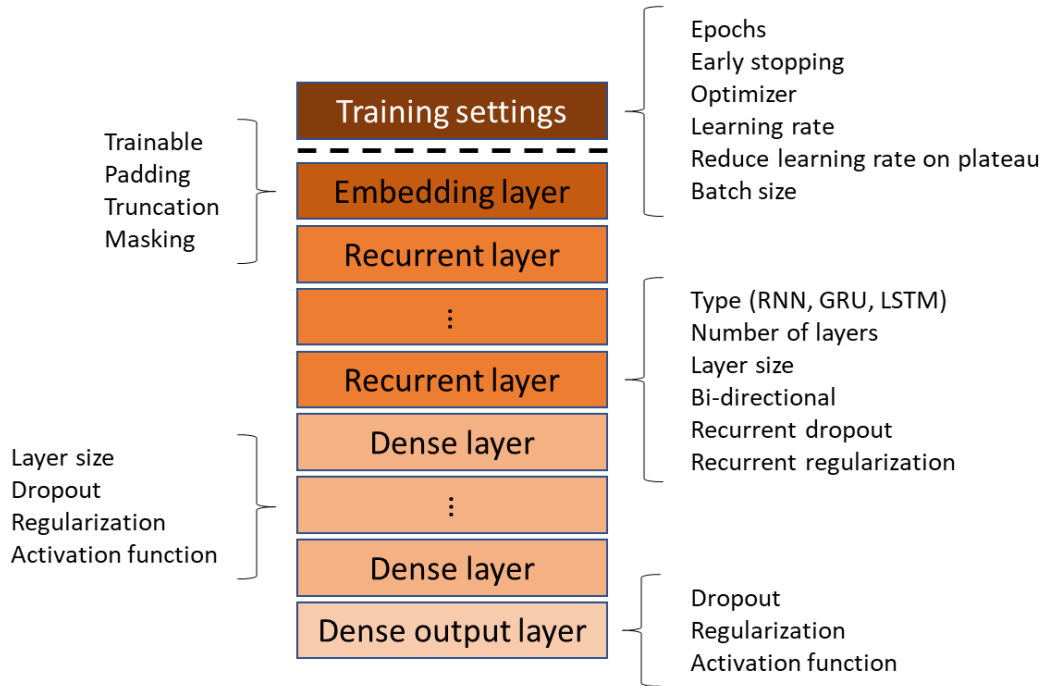


Figure L.1: General structure of a RNN and the hyperparameters associated

The hyperparameters related to the training setting are the same as described in section 11.3.

The embedding layer also has the same hyperparameters except for, with a recurrent neural network, it is common to mask zero-padded inputs, making sure that the additional ‘empty’ timesteps are not considered in the forward pass or backpropagation.

A general recurrent architecture has one or more recurrent layers. These can generally be of type RNN, GRU, or LSTM. If there are multiple, they are stacked on each other. These layers can be set to different sizes and be bi-directional or not. Recurrent dropout and regularization can be applied to the layer but the former was not examined for this thesis as it is not compatible with GPU training and therefore extremely slow to train. The recurrent layers all have activation functions but they are considered part of the unit as

described in section 6.3.3 and will not be changed during training. The input weights were initialized with the Glorot uniform method and the hidden weights were initialized with the orthogonal method as described in section 6.5.2.3.

The recurrent layer(s) is followed by one or more dense layers. The output dense layer must have the same number of neurons as the number of classes that the model classifies when using the softmax activation function. If there are dense layers prior to the output layer, they can be of variable size and with different activation functions applied. The ReLU was used for all dense layers but the output layer and the weights were initialized using Glorot uniform method. Dropout and regularization can be applied to all dense layers.

The further sections first test which type of recurrent layer to be used, then how many dense layers, followed by the number of recurrent layers. Next, the best optimizer and learning rate for that architecture is found. Finally, the last hyperparameters are tuned and the best recurrent architecture is chosen.

Architecture hyperparameters

The Adam optimizer with default settings [84] and learning rates 5.0E-4, 1.0E-4, and 5.0E-5 were used throughout the tuning of the architecture hyperparameters because it was found that it is very robust to changes in the model. Additionally, the batch size was set to 128 and embeddings to non-trainable for all training loops as they were not expected to interact with the architecture hyperparameters.

Recurrent layer type

A training loop was set up to test which of the RNN, GRU, and LSTM recurrent layers, including their bidirectional versions, performed best. A single recurrent layer of sizes 32, 64, and 128 was tested. Since the bidirectional layers each consist of two layers of different direction concatenated at the last timestep, they double the number of neurons. Dropout of either 0 or 0.3 was applied before the output dense layer.

Figure L.2 shows the minimum loss produced by each of the different recurrent layer types across all tested hyperparameters.

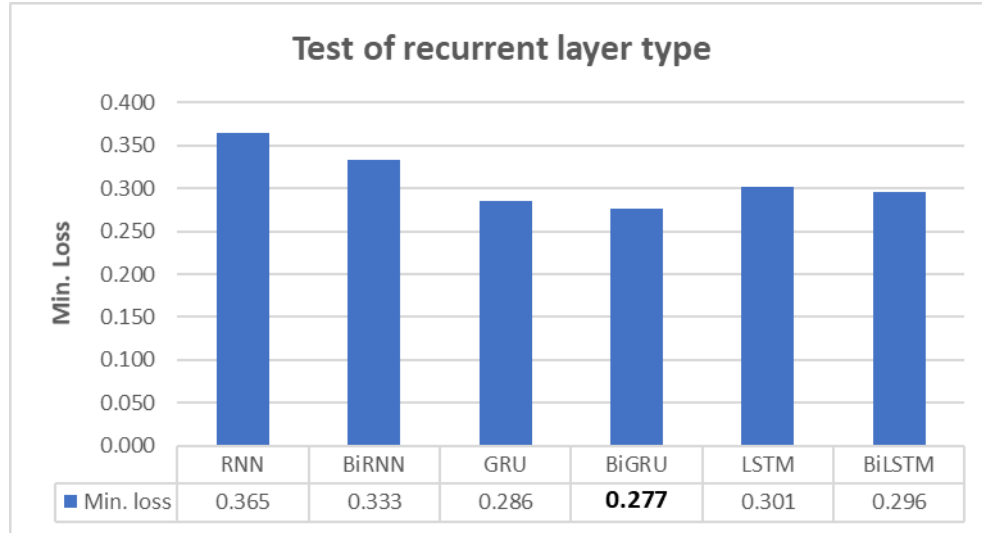


Figure L.2: Test of recurrent layer type

It can be seen that the BiGRU layer gives the lowest loss across all tested hyperparameters and is therefore used moving forward.

Number of dense layers

The number of dense layers to be used between the recurrent layer and the final output dense layer was tested with the following training loop: 0, 1, or 2 dense layers of size 64, 128, or 256 was placed before the output layer. The number of dense layers was tested with only 1 BiGRU recurrent layer to have the model performance depend more on the dense layer setting. The recurrent layer had sizes 32, 64, or 128 as the performance of the dense layer could rely on the output size from the recurrent layer. Dropout of either 0 or 0.3 was applied before all dense layers in the architecture.

Figure L.3 shows the minimum loss produced across all tested hyperparameters for 0, 1, and 2 dense layers.

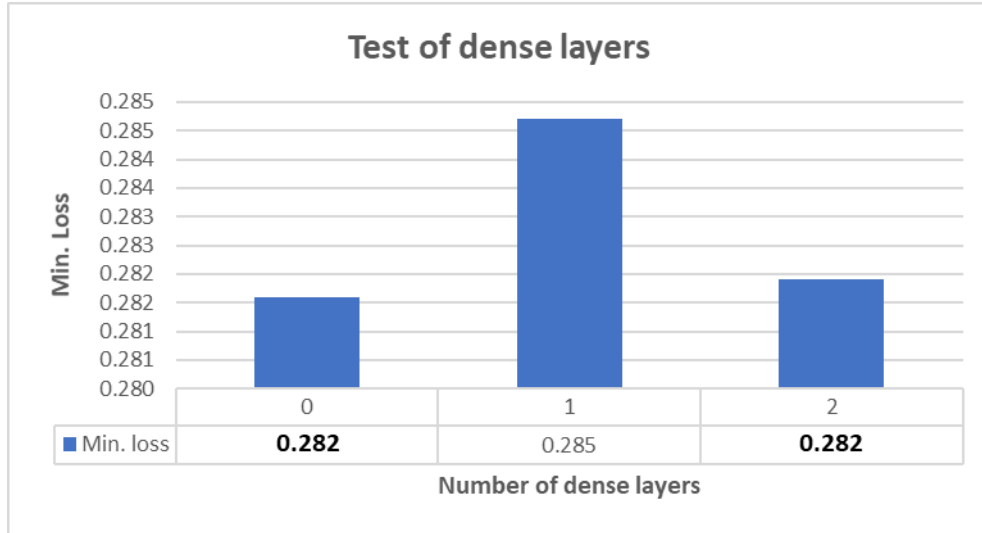


Figure L.3: Test of RNN dense layers

While there is virtually no difference in the minimum loss obtained with the different number of dense layers, it is chosen to go forward with no dense layers apart from the final output layer to make the model as simple as possible. This mitigates overfitting and increases model speed.

Number of recurrent layers

The optimal number of stacked recurrent layers to be used with the BiGRU based architecture with a single output dense layer was tested with the following training loop: 1, 2, or 3 BiGRU layers of sizes 16, 32, 64, 128, 256, 512, and 1024. A wide range of layer sizes was used to test if e.g. one large BiGRU layer of size 1024 would perform better than three stacked BiGRU layers of size 32. Dropout of 0, 0.25, or 0.50 was applied before the output dense layer to account for possible overfitting with the highest BiGRU layer sizes. For the same reason, weight regularization of 0 or 0.01 was applied to the output dense layer and BiGRU layer.

Figure L.4 shows the minimum loss produced across all tested hyperparameters for 1, 2, and 3 recurrent layers.

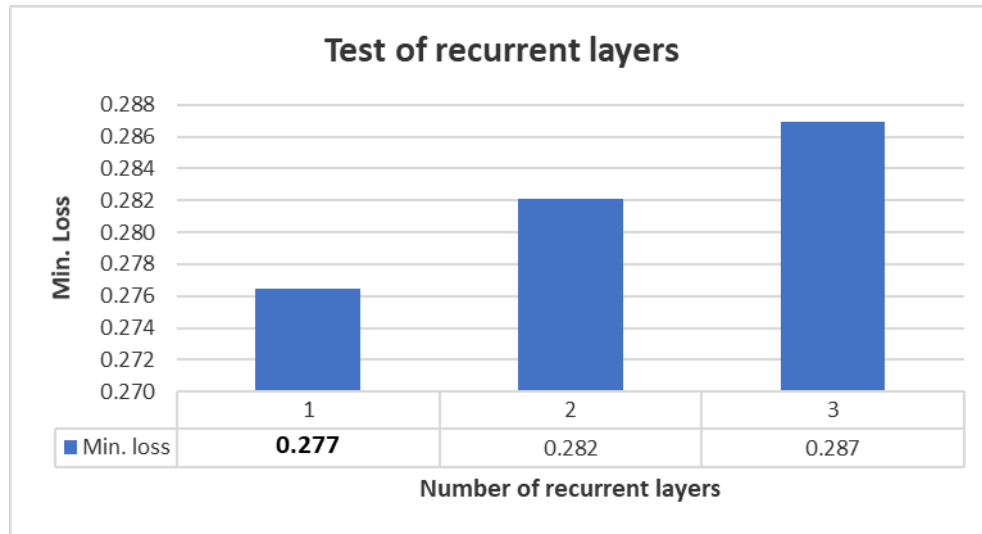


Figure L.4: Test of recurrent layers

A single recurrent layer produced the lowest loss across all tested hyperparameters and is used moving forward.

Recurrent layer size

The optimal recurrent layer size was explored with the architecture hyperparameters as chosen in the previous tests. Layer sizes 4, 8, 16, 32, 64, 128, 256, 512, and 1024 were tested. There were variations in dropout (0, 0.25, 0.30, 0.50) and regularization (0, 0.01) as these were thought to be able to interact with the hyperparameter in question. Figure L.5 shows the minimum loss achieved with each layer size.

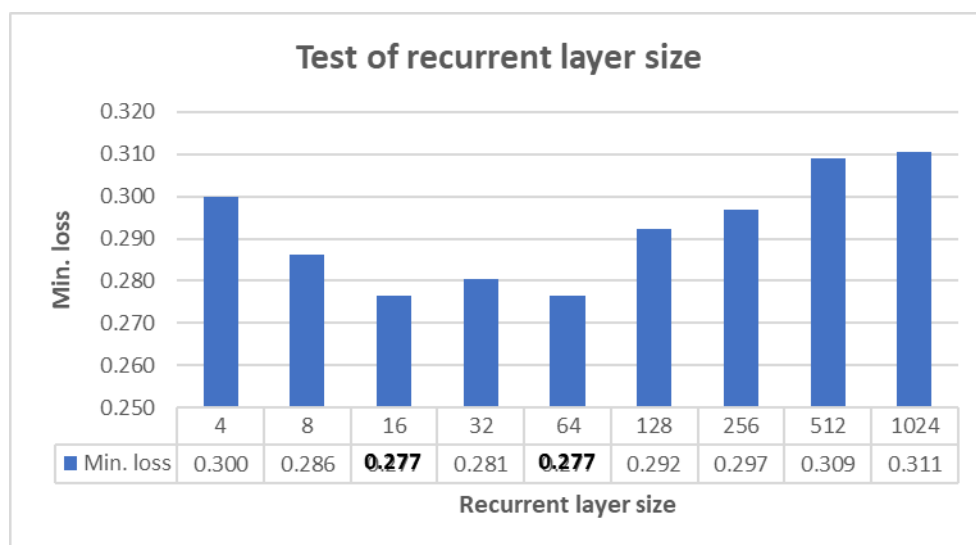


Figure L.5: Test of recurrent layer size

The figure shows that the optimum recurrent layer size seems to be around 16 to 64 neurons with both 16 and 64 giving minimum losses of 0.277. It is chosen to move forward with a recurrent layer size of 16 to have the simplest model possible.

Optimizer and learning rate

To find the best optimizer and learning rate to use with the chosen architecture, Adam with default settings and SGD with momentum of 0.9 were tested in a range of learning rates until an ‘optimum’ was found. The architecture hyperparameters were used with dropout 0, 0.25, and 0.5 in the training loop.

Figure L.6 shows the minimum loss produced across all hyperparameters for the Adam and SGD optimizers.

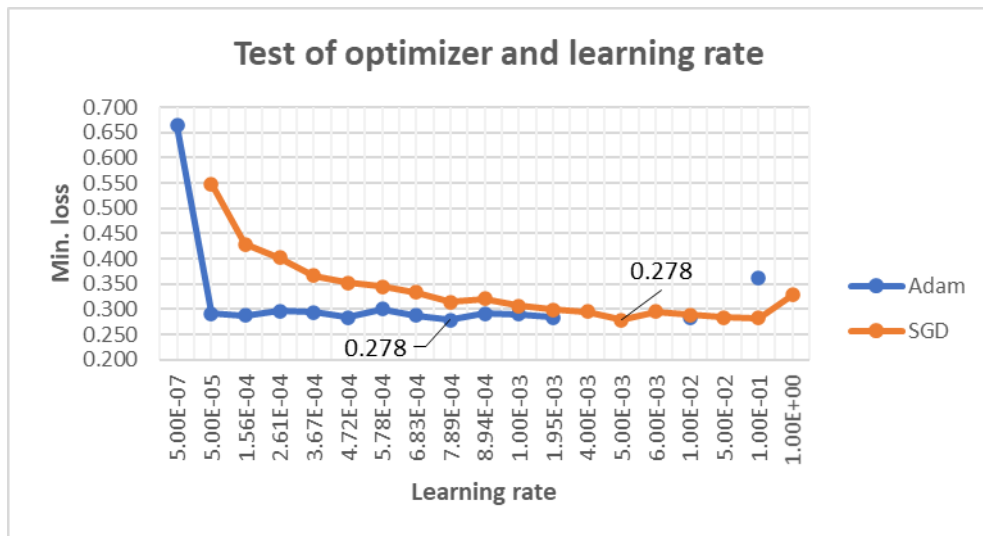


Figure L.6: Test of optimizer and learning rate

Both optimizers produced the same minimum loss of 0.278 across all tested hyperparameters. Adam with a learning rate of 7.89E-04 and SGD with a learning rate of 5.00E-03. It is noteworthy that Adam performed well with almost all tested learning rates except for the extremes of the interval. As the two optimizers perform similarly, the rate of learning for the best model of each optimizer is explored on Figure L.7.

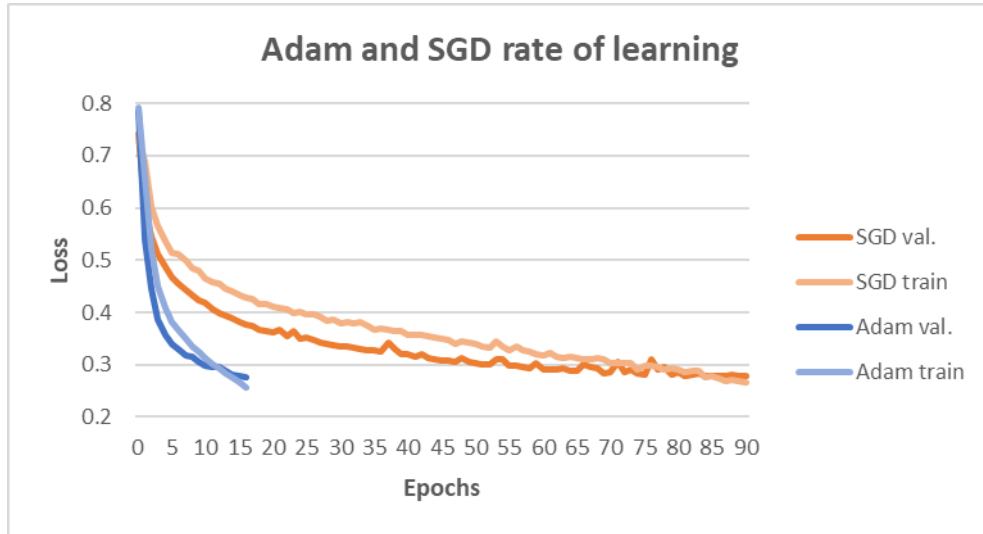


Figure L.7: Rate of learning for Adam and SGD

It is seen that while both reach a similar average loss of the last 3 epochs before no further improvement, Adam reaches it after 21 epochs while SGD needs 96 epochs. It was chosen to use Adam with learning rate $7.89\text{E-}04$ because it trains faster and is more robust to model variations.

Setting hyperparameters

The setting hyperparameters were tested on the chosen architecture hyperparameters: A single BiGRU layer of size 16 with an output dense layer. Dropout between the BiGRU layer and the output dense layer at 0.1, 0.2, 0.3, 0.4, and 0.5 was tested. Regularization of 0, 0.0001, 0.001, and 0.01 was applied to the dense and BiGRU layer. The embedding layer was tested as trainable and non-trainable. The model was trained with the Adam optimizer with a learning rate of $7.89\text{E-}04$ and at batch sizes 8, 16, 32, 64, 128, 256, and 512.

Table L.1 shows the final model with the chosen best architecture and setting hyperparameters.

	Output
Embedding layer Trainable	32x100x100
BiGRU layer	32x32

Size 16x2, regularization 0	
Dropout layer Dropout 0.3	32x32
Dense layer Size 2, regularization 0	32x2

Table L.1: Final RNN model architecture and hyperparameters

The best model achieved a minimum loss averaged over the best and 2 prior epochs of 0.2729 and a validation accuracy of 90.24 %.

Appendix M: Code for rule-based classifier

Import libraries

```
1. import copy
2. import stanza
3. import pandas as pd
4. import matplotlib.pyplot as plt
5. %matplotlib inline
```

Import data

```
1. df = pd.read_excel (r'C:\Users\marti_000\OneDrive - Syddansk Universi-
   tet\SPECIALE\Rule Based\Rule_Based_Sam-
   ples_train_val_test.xlsx') #(use "r" before the path string to address spe-
   cial character, such as '\'). Don't for-
   get to put the file name at the end of the path + '.xlsx'
2. #Find length
3. rows=df.shape[0]
4. #Select index
5. df.iloc[0,0]
6. x_val=[]
7. y_val=[]
8. for i in range(rows):
9.     if df.iloc[i,4]=='val':
10.         x_val.append(df.iloc[i,0])
11.         if df.iloc[i,1]=='negative' or df.iloc[i,1]=='rand':
12.             y_val.append('negative')
13.         if df.iloc[i,1]=='positive':
14.             y_val.append('positive')
15.
16. rows=len(x_val)
17. print('val samples: ',rows)
```

Predict function

```
1. import re
2.
3. #Takes input sample, window where negation words count, list of posi-
   tive words, list of negtions
4. def predictSample(text,window,pos_list,neg_list):
5.     positive_index=[]
6.     for pos in pos_list:
7.         positive_in-
           dex+= [m.start() for m in re.finditer(pos, text)] #Find the index of all pos-
           itive words in the sample
8.     if len(positive_index)==0:
9.         return 'unclassified' #If none are found, return unclassi-
           fied for now, but they are negative
10.
11.     for i in range(len(positive_index)): #For each positive word
12.         words_around=[m.start() for m in re.finditer(' ', text)] #Find the s
           paces in the text (representing words)
13.         words_around.insert(0,0) #Include start of string
14.         words_around.append(len(text)) #...and end of string
```

```

15.
16.     #Count window of negation from start to end
17.     dist=-1 #Starts at -1 because first space will be by the posi-
18.     tive word, not the first surrounding word
19.     for j in range(len(words_around)):
20.         if words_around[j]>positive_index[i]:
21.             dist+=1 #Count up distance when it is past the posi-
22.             tive 'center' word
23.             if dist==window or j==len(words_around)-1: #If win-
24.                 dow is done or no more words
25.                     cut_upper=words_around[0:j+1] #Cut window
26.                     break #Else it would include j==len(words_around)-1
27.
28.     #Count window of negation from end to start
29.     dist=-1
30.     for j in range(len(cut_upper)):
31.         if cut_upper[len(cut_upper)-(j+1)]<positive_index[i]:
32.             dist+=1
33.             if dist==window or j==len(cut_upper)-1:
34.                 cut_lower=cut_upper[len(cut_upper)-(j+1):len(cut_upper)]
35.                 break
36.
37.     #Search for negations around positive word
38.     pred_status=True
39.     for neg in neg_list:
40.         if text.find(neg,cut_lower[0],cut_lower[len(cut_lower)-1])!=
41.         1: #If negation is found, current status is 'negative'
42.             pred_status=False
43.             break #No need to search for more nega-
44.             tions for the same positive word
45.     if pred_status==True:
46.         return 'positive' #If one posi-
47.         tive word has been run though all negations and passed, the predic-
48.         tion is 'positive'
49.     return 'negative' #When all posi-
50.     tive words have been searched and all had negations

```

Classify

```

1. pos_list=[
2.     'koag',
3.     'frisk',
4.     'petek',
5.     'sag-m',
6.     'sagm',
7.     'sag m',
8.     'hæm',
9.     'haem',
10.    'blod',
11.    'epistaxis',
12.    'kaffegrums',
13.    'blød',
14.    'blood',
15.    'ulcerat',
16.    'gastropati',
17.    'hævelse',
18.    'haevelse',
19.    'hævet',
20.    'haevet',
21.    'melæn',

```

```

22.     'melaen',
23.     'blåt mærk',
24.     'blå mærk'
25.
26.
27. ]
28.
29. neg_list=[
30.     'ikke',
31.     'ingen',
32.     'intet',
33.     'muligvis',
34.     'ikke mistanke',
35.     'negativ',
36.     'risi',
37.     'regelmæs',
38.     'uden',
39.     'hvis',
40.     'sukker',
41.     'dyrkning',
42.     'fortynd',
43.     'procent'
44. ]
45.
46. y_pred=[]
47. for x in x_val:
48.     text=x.lower() #Lower-case
49.     y_pred.append(predictSample(text,4,pos_list,neg_list))
50.
51.
52. #ACCURACY OF CLASSIFIED
53. correct_classified=0
54. classified=0
55. for p in range(len(y_pred)):
56.     if y_pred[p]!='unclassified':
57.         classified+=1
58.         if y_val[p]==y_pred[p]:
59.             correct_classified+=1
60. acc=correct_classified/classified
61. print('Accuracy of classified: ',acc)
62. print('Part of total classified: ',classified/len(y_pred))
63.
64. #UNCLASSIFIED = NEGATIVE
65. for i in range(len(y_pred)):
66.     if y_pred[i]=='unclassified':
67.         y_pred[i]='negative'
68.
69. #TOTAL ACCURACY
70. correct=0
71. for p in range(len(y_pred)):
72.     if y_val[p]==y_pred[p]:
73.         correct+=1
74. acc=correct/len(y_pred)
75. print('Total accuracy: ',acc)
76.
77. #CONFUSION MATRIX
78. pos_true=0
79. pos_false=0
80. neg_true=0
81. neg_false=0
82. for i in range(len(y_val)):
83.     if y_val[i]==y_pred[i]:
84.         if y_val[i]=='positive':

```

```
85.         pos_true+=1
86.     else:
87.         neg_true+=1
88.     else:
89.         if y_val[i]=='positive':
90.             neg_false+=1
91.         else:
92.             pos_false+=1
93. print('\nConfusion Matrix\n\tPOS\tNEG\nPOS\t',pos_true,'\t',neg_false,'\nNEG\t',pos_false,'\t',neg_true,'\n')
94. precision = pos_true/(pos_true+pos_false)
95. recall = pos_true/(pos_true+neg_false)
96. F1 = 2*((precision*recall)/(precision+recall))
97. print("Precision: ", round(precision,2), "Recall: ",round(recall,2), "F1: ", round(F1, 2) )
98. #print('y_val','\t\t','y_pred','\t','x_val')
99. #for i in range(len(y_pred)):
100. #    print(y_val[i],'\t',y_pred[i],'\t',x_val[i])
```

Appendix N: Code for training CNN and RNN

```

1. import os
2. from tensorflow.keras.models import Sequential, Model, load_model
3. from tensorflow.keras.layers import Concatenate, Input, Embedding, SimpleRNN, Dense, Flatten, LSTM, GRU, Bidirectional, Dropout, Conv1D, MaxPooling1D, GlobalMaxPooling1D
4. from tensorflow.keras.datasets import imdb
5. from tensorflow.keras.preprocessing import sequence
6. import matplotlib.pyplot as plt
7. from tensorflow.keras.preprocessing import preprocessing
8. from tensorflow.keras.preprocessing.text import Tokenizer
9. from tensorflow.keras.preprocessing.sequence import pad_sequences
10. import numpy as np
11. from tensorflow.keras.optimizers import Adam, SGD
12. from tensorflow.keras.utils import plot_model
13. import tensorflow
14. from tensorflow.keras import regularizers
15. import pandas as pd
16. import matplotlib.pyplot as plt
17. from nltk import word_tokenize
18.
19. ##### VARIABLES #####
20. import pathlib
21. excel_samples='Sentence_dataset_samples.xlsx'
22. max_length_sample=100 #truncation
23. excel_name='test_dataset_size%.xls' #Output
24. filepath=str(pathlib.Path().absolute())+"/CNN weights/"
25.
26. #Load data
27. data = pd.read_excel(excel_samples)
28. print(data.head(2))
29. data['Sample'] = data['Sample'].str.strip()
30. print(data.head(2))
31.
32. #Preprocess
33. repl_list = {'#\.' : ' ',
34.             '\.' : ' ',
35.             '€' : ' ',
36.             '\$' : ' ',
37.             '@' : ' ',
38.             '\[' : ' ',
39.             '\]' : ' ',
40.             '\*' : ' ',
41.             '\^' : ' ',
42.             '-' : ' ',
43.             '~' : ' ',
44.             '/' : ' ',
45.             ';' : ' ',
46.             '&' : ' og ',
47.             ':' : ' ',
48.             '-' : ' ',
49.             '\+' : ' + ',
50.             '\(' : ' ',
51.             '\)' : ' ',
52.             '#' : ' ',
53.             '#' : ' ',
54.             '!' : ' ',
55.             '\`' : ' ',
56.             '°' : ' ',
57.             '"' : ' ',

```



```

58.         "'": ' ',
59.         '%': ' ',
60.         '\s+': ' '}
61.
62.
63. data.Sample.replace(repl_list, regex=True, inplace=True)
64.
65. #Create train, val, test
66. x_train= [sentence for sentence in data.Sample[data.Set=='train']]
67. y_train= [label for label in data.Label[data.Set=='train']]
68. y_train_num = [1 if label=='positive' else 0 for label in y_train]
69.
70. x_val= [sentence for sentence in data.Sample[data.Set=='val']]
71. y_val= [label for label in data.Label[data.Set=='val']]
72. y_val_num = [1 if label=='positive' else 0 for label in y_val]
73.
74. x_test= [sentence for sentence in data.Sample[data.Set=='test']]
75. y_test= [label for label in data.Label[data.Set=='test']]
76. y_test_num = [1 if label=='positive' else 0 for label in y_test]
77.
78. print(len(x_train), len(y_train_num), len(x_val), len(y_val_num), len(x_test
), len(y_test_num))
79.
80. ##Tokenize tekst
81. from nltk import word_tokenize
82. from collections import Counter
83. import stanza
84.
85. ##Function for tokenize tokens in sentences:
86. def token_tokenize(_class):
87.     nlp = stanza.Pipeline(lang='da', processors='tokenize', to-
kenize_no_ssplite=True)
88.     tokenized_sentences=[]
89.     for i in range(len(_class)): #For each row
90.         doc = nlp(_class[i])
91.         for i, sentence in enumerate(doc.sentences):
92.             tokens_list = []
93.             for token in sentence.tokens:
94.                 tokens_list.append(token.text)
95.             tokenized_sentences.append(tokens_list[:])
96.     return tokenized_sentences
97.
98. #Tokenize sentences
99. x_train_tokenized = token_tokenize(x_train)
100. x_val_tokenized = token_tokenize(x_val)
101. x_test_tokenized = token_tokenize(x_test)
102. print("Tokenized sentence example:\n ",x_train_tokenized[1])
103.
104. #Create one vector 'all_words', with all sentences com-
bined (needed for later use)
105. all_words_train = [word for sample in x_train_tokenized for word in sam-
ple]
106. all_words_train += [word for sample in x_val_tokenized for word in sam-
ple]
107. all_words_train += [word for sample in x_test_tokenized for word in sam-
ple]
108. print("Total number of words: ",len(all_words_train))
109.
110. #Count occurrences of each word using 'Counter'
111. word_counter = Counter(all_words_train)
112. print("\nCommon words and counts (key-value pairs): ", "\n", word_coun-
ter.most_common(8))
113.

```

```

114. #Sort the words from high to low after number of occurrences
115. words_sorted_by_count = sorted(word_counter, key=word_counter.get, re-
    verse=True)
116.
117. #Create dictionary of words with most frequent word first
118. word_index={} #Dictionary
119. for i in range(len(words_sorted_by_count)):
120.     word_index[words_sorted_by_count[i]]=i+1 #Creates dict start-
    ing from 1 (0 is saved for PAD words)
121. word_index['UNK'] = len(word_index)+1
122. print("\nWord_index (unique tokens: ", len(word_in-
    dex),") :", "\n", word_index)
123.
124. #Convert text to sequence of numbers
125. def text_to_seq(tokenized_text):
126.     text_sequence=[]
127.
128.     for sentence in tokenized_text:
129.         subsentence=[] #Subsentence
130.         for word in sentence:
131.             if word_index.get(word)==None:
132.                 subsentence.append(word_index['UNK'])
133.             else:
134.                 subsentence.append(word_index[word])
135.         text_sequence.append(subsentence)
136.     return text_sequence
137.
138. #Use the function for both train and val data
139. x_train_sequence = text_to_seq(x_train_tokenized)
140. x_val_sequence = text_to_seq(x_val_tokenized)
141.
142. print("Original train sentence:", x_train[1], "\nSequence sen-
    tence", x_train_sequence[1])
143. print("\nOriginal val sentence:", x_val[0], "\nSequence sen-
    tence", x_val_sequence[0])
144.
145. #Use zero_padding
146. from keras.preprocessing.sequence import pad_sequences
147.
148. x_train_sequence = pad_sequences(x_train_sequence,maxlen=max_length_sam-
    ple, padding='post', truncating='post')
149. x_val_sequence = pad_sequences(x_val_sequence,maxlen=max_length_sam-
    ple, padding='post', truncating='post' )
150.
151. print("Padded sentence: ", x_val_sequence[0])
152.
153. #Convert to numpy array
154. x_val_sequence=np.asarray(x_val_sequence)
155. y_val_num=np.asarray(y_val_num)
156. x_train_sequence=np.asarray(x_train_sequence)
157. y_train_num=np.asarray(y_train_num)
158.
159. print("Train shape:", x_train_sequence.shape, y_train_num.shape)
160. print("Val shape: ", x_val_sequence.shape, y_val_num.shape)
161.
162. ##ITU GLOVE##
163. glove_dir = r'clin.glove.txt'
164. embeddings_index = {}
165.
166. f = open(glove_dir, encoding="utf8")
167. for line in f:
168.     values = line.split()
169.     word = values[0]

```

```

170.     coefs = np.asarray(values[1:], dtype='float32')
171.     embeddings_index[word] = coefs
172.     f.close()
173.     print('Found %s word vectors.' % len(embeddings_index))
174.
175.     #Import and prepare the GloVe word-embeddings matrix
176.     embedding_dim=100
177.     vocab_size = len(word_index)+1
178.     embedding_matrix = np.zeros((vocab_size, embedding_dim))
179.
180.     oov_words=[]
181.
182.     for word, i in word_index.items():
183.         if i < vocab_size:
184.             embedding_vector = embeddings_index.get(word)
185.             if embedding_vector is not None:
186.                 embedding_matrix[i] = embedding_vector #Con-
187.                 tains only wordvectors,
188.                 elif word[0].isupper(): #If uppercased look for lower-
189.                 cased word in GloVe
190.                 embedding_vector = embeddings_in-
191.                 dex.get(word[0].lower()+word[1:])
192.                 if embedding_vector is not None:
193.                     print(word)
194.                     embedding_matrix[i] = embedding_vector #Con-
195.                     tains only wordvectors,
196.                     elif word[0].islower(): #If lowercased look for lower-
197.                     cased word in GloVe
198.                     embedding_vector = embeddings_index.get(word.capitalize-
199.                     size())
200.                     if embedding_vector is not None:
201.                         print(word)
202.                         embedding_matrix[i] = embedding_vector #Con-
203.                         tains only wordvectors,
204.                         else:
205.                             oov_words.append(word) #words not in vocab
206.     print("\nnumber of OOV words: ", len(oov_words), "\nExamples:")
207.
208.     for i in range(10):
209.         print(oov_words[i])
210.
211.     #Create function to load results into excel
212.     import xlwt
213.     from xlwt import Workbook
214.     from xlutils.copy import copy
215.     import xlrd
216.     from xlrd import open_workbook
217.
218.     def to_excel(sheet1, embedding_type, batch_size, learning_rate, opti-
219.     mizer, filter_, filter_size, drop-
220.     out, conv_layer, reg, dense_layer, dense_size, his-
221.     tory, train_begin, val_begin, row):
222.         #Training data loss
223.         sheet1.write(row,0, 'Train')
224.         sheet1.write(row,1, embedding_type)
225.         sheet1.write(row,2, batch_size)
226.         sheet1.write(row,3, optimizer)
227.         sheet1.write(row,4, learning_rate)
228.         sheet1.write(row,5, conv_layer)
229.         sheet1.write(row,6, filter_)
230.         sheet1.write(row,7, str(filter_size))

```

```

223.     sheet1.write(row,8, dense_layer)
224.     sheet1.write(row,9, dense_size)
225.     sheet1.write(row,10, dropout)
226.     sheet1.write(row,11, reg)
227.     sheet1.write(row,12, 'Loss')
228.     if len(history['loss'])>17:
229.         sheet1.write(row, 13, round((float(history['loss'][-
16])+float(history['loss'][-17])+float(history['loss'][-18]))/3,4))
230.     else:
231.         sheet1.write(row, 13, round((float(history['loss'][0])+float(history['loss'][1])+float(history['loss'][2]))/3,4))
232.     sheet1.write(row,14,round(float(train_begin[0]),4))
233.     for i in range(len(history['loss'])):
234.         sheet1.write(row, i+15, round(float(history['loss'][i]),4))
235.
236.     #Training data accuracy
237.     row+=1
238.     sheet1.write(row,0, 'Train')
239.     sheet1.write(row,1, embedding_type)
240.     sheet1.write(row,2, batch_size)
241.     sheet1.write(row,3, optimizer)
242.     sheet1.write(row,4, learning_rate)
243.     sheet1.write(row,5, conv_layer)
244.     sheet1.write(row,6, filter_)
245.     sheet1.write(row,7, str(filter_size))
246.     sheet1.write(row,8, dense_layer)
247.     sheet1.write(row,9, dense_size)
248.     sheet1.write(row,10, dropout)
249.     sheet1.write(row,11, reg)
250.     sheet1.write(row,12, 'Accuracy')
251.     if len(history['acc'])>17:
252.         sheet1.write(row, 13, round((float(history['acc'][-
16])+float(history['acc'][-17])+float(history['acc'][-18]))/3,4))
253.     else:
254.         sheet1.write(row, 13, round((float(history['acc'][0])+float(history['acc'][1])+float(history['acc'][2]))/3,4))
255.     sheet1.write(row,14,round(float(train_begin[1]),4))
256.     for i in range(len(history['acc'])):
257.         sheet1.write(row, i+15, round(float(history['acc'][i]),4))
258.
259.     #Validation data loss
260.     row+=1
261.     sheet1.write(row,0, 'Val')
262.     sheet1.write(row,1, embedding_type)
263.     sheet1.write(row,2, batch_size)
264.     sheet1.write(row,3, optimizer)
265.     sheet1.write(row,4, learning_rate)
266.     sheet1.write(row,5, conv_layer)
267.     sheet1.write(row,6, filter_)
268.     sheet1.write(row,7, str(filter_size))
269.     sheet1.write(row,8, dense_layer)
270.     sheet1.write(row,9, dense_size)
271.     sheet1.write(row,10, dropout)
272.     sheet1.write(row,11, reg)
273.     sheet1.write(row,12, 'Loss')
274.     if len(history['val_loss'])>17:
275.         sheet1.write(row, 13, round((float(history['val_loss'][-
16])+float(history['val_loss'][-17])+float(history['val_loss'][-
18]))/3,4))
276.     else:

```

```

277.         sheet1.write(row, 13, round((float(history[
    history['val_loss'][0])+float(history['val_loss'][1])+float(history[
    history['val_loss'][2]))/3,4))
278.         sheet1.write(row,14, round(float(val_begin[0]),4))
279.         for i in range(len(history['val_loss'])):
280.             sheet1.write(row, i+15, round(float(history[
    history['val_loss'][i]),4))
281.
282.         #Val data accuracy
283.         row+=1
284.         sheet1.write(row,0, 'Val')
285.         sheet1.write(row,1, embedding_type)
286.         sheet1.write(row,2, batch_size)
287.         sheet1.write(row,3, optimizer)
288.         sheet1.write(row,4, learning_rate)
289.         sheet1.write(row,5, conv_layer)
290.         sheet1.write(row,6, filter_)
291.         sheet1.write(row,7, str(filter_size))
292.         sheet1.write(row,8, dense_layer)
293.         sheet1.write(row,9, dense_size)
294.         sheet1.write(row,10, dropout)
295.         sheet1.write(row,11, reg)
296.         sheet1.write(row,12, 'Accuracy')
297.         if len(history['val_acc'])>17:
298.             sheet1.write(row, 13, round((float(history['val_acc'][-
    16])+float(history['val_acc'][-17])+float(history['val_acc'][-18]))/3,4))
299.         else:
300.             sheet1.write(row, 13, round((float(history[
    history['val_acc'][0])+float(history['val_acc'][1])+float(history[
    history['val_acc'][2]))/3,4))
301.             sheet1.write(row,14, round(float(val_begin[1]),4))
302.             for i in range(len(history['val_acc'])):
303.                 sheet1.write(row, i+15, round(float(history['val_acc'][i]),4))
304.
305.         row+=1
306.         return row
307.
308.     #Training loop
309.     import tensorflow as tf
310.     import keras
311.     #Plot results
312.     def plot(history, train_begin, val_begin):
313.         acc = history.history['acc']
314.         acc = [train_begin[1]]+acc
315.         val_acc = history.history['val_acc']
316.         val_acc = [val_begin[1]]+val_acc
317.         loss = history.history['loss']
318.         loss = [train_begin[0]]+loss
319.         val_loss = history.history['val_loss']
320.         val_loss = [val_begin[0]]+val_loss
321.
322.         epochs = range(1, len(acc) + 1)
323.
324.         plt.plot(epochs, acc, 'bo', label='Training acc')
325.         plt.plot(epochs, val_acc, 'b', label='Validation acc')
326.         plt.title('Training and validation accuracy')
327.         plt.legend()
328.
329.         plt.figure()
330.
331.         plt.plot(epochs, loss, 'bo', label='Training loss')
332.         plt.plot(epochs, val_loss, 'b', label='Validation loss')
333.         plt.title('Training and validation loss')

```

```

334.     plt.legend()
335.     plt.show()
336.
337.     #RNN model
338.     def build_rnn_model(dense_size, dense_layer, reg, recurrent_drop-
339.         out, GRU_layer, train_embed, rnn_dim, dropout):
340.         model = Sequential()
341.         model.add(Embedding(vocab_size, embedding_dim,
342.             input_length=None,
343.             weights=[embedding_matrix],
344.             trainable=train_embed, mask_zero=True))
345.         for i in range(GRU_layer-1):
346.             model.add(Bidirectional(GRU(rnn_dim, return_sequences=True, re-
347.                 current_dropout=recurrent_dropout, kernel_regularizer=regulariz-
348.                 ers.l2(reg), recurrent_regularizer=regularizers.l2(reg))))
349.             model.add(Bidirectional(GRU(rnn_dim, recurrent_dropout=recur-
350.                 rent_dropout, kernel_regularizer=regularizers.l2(reg), recurrent_regul-
351.                 arizer=regularizers.l2(reg))))
352.         for i in range(dense_layer):
353.             model.add(Dropout(dropout))
354.             model.add(Dense(dense_size, activation='relu', kernel_regul-
355.                 arizer=regularizers.l2(reg)))
356.             model.add(Dropout(dropout))
357.             model.add(Dense(2, kernel_regularizer=regularizers.l2(reg), activa-
358.                 tion='softmax'))
359.         return model
360.
361.     ##CNN model
362.     def build_cnn_model(train_embed, filter_size, filter_,
363.         dropout_, conv_layer, reg,
364.         dense_layer, dense_size):
365.         _input=Input(shape=(None,))
366.         embedding = Embedding(vocab_size, embedding_dim,
367.             input_length=max_length_sample,
368.             weights=[embedding_matrix],
369.             trainable=train_embed)(_input)
370.
371.         conv_blocks=[]
372.         for size in filter_size:
373.             conv = Conv1D(filter_, size, activation='relu', pad-
374.                 ding='valid')(embedding)
375.             for i in range(conv_layer-1):
376.                 conv = Conv1D(filter_, size, activation='relu', pad-
377.                     ding='valid')(conv)
378.             Global_max_pool = GlobalMaxPooling1D()(conv) #Global maxpool-
379.             ing
380.             conv_blocks.append(Global_max_pool)
381.             if len(filter_size)==1:
382.                 concat = Global_max_pool
383.             else:
384.                 concat = Concatenate()(conv_blocks)
385.             for i in range(dense_layer):
386.                 if i==0:
387.                     dropout = Dropout(dropout_)(concat)
388.                 else:
389.                     dropout = Dropout(dropout_)(dense)
390.                 dense = Dense(dense_size, activation='relu',
391.                     kernel_regularizer=regularizers.l2(reg))(drop-
392.                     out)
393.             if dense_layer==0:
394.                 dropout = Dropout(dropout_)(concat)

```

```

386.         else:
387.             dropout = Dropout(dropout_)(dense)
388.             output = Dense(2, activation='softmax',
389.                             kernel_regularizer=regularizers.l2(reg))(dropout)
390.             model = Model(_input, output)
391.             return model
392.
393.
394. #Initialize excel document
395. wb = Workbook()
396. sheet1 = wb.add_sheet('Sheet 1')
397. row=1
398.
399. embedding_type='ITU_Glove'
400. _epochs = 100
401. batch_sizes=[16]
402. learning_rates=[5e-6]
403. optimizers=['ADAM']
404. filters=[1792]
405. filter_sizes=[[1,2]]
406. dropouts=[0.4]
407. conv_layers=[2]
408. regs=[0.0001]
409. dense_layers=[0]
410. dense_sizes=[0]
411.
412.
413. callbacks_list=[keras.callbacks.EarlyStopping(monitor='val_loss',
414.                                                min_delta=0,
415.                                                patience=15,
416.                                                verbose=0,
417.                                                mode='auto',
418.                                                baseline=None,
419.                                                rest-
420. ore_best_weights=False),
421.                 keras.callbacks.ReduceLROnPlateau(moni-
422. tor='val_loss',
423. tor=0.2,
424. tor=0.2,
425. tor=0.2,
426. tor=0.2,
427. tor=0.2,
428. tor=0.2,
429. tor=0.2,
430. tor=0.2,
431. tor=0.2,
432. tor=0.2,
433. tor=0.2,
434. tor=0.2,
435. tor=0.2,
436. tor=0.2,
437. tor=0.2,
438. tor=0.2,
439. tor=0.2,
440. tor=0.2,
441. tor=0.2,
442. tor=0.2,
443. tor=0.2,
444. tor=0.2,
445. tor=0.2,
446. tor=0.2,
447. tor=0.2,
448. tor=0.2,
449. tor=0.2,
450. tor=0.2,
451. tor=0.2,
452. tor=0.2,
453. tor=0.2,
454. tor=0.2,
455. tor=0.2,
456. tor=0.2,
457. tor=0.2,
458. tor=0.2,
459. tor=0.2,
460. tor=0.2,
461. tor=0.2,
462. tor=0.2,
463. tor=0.2,
464. tor=0.2,
465. tor=0.2,
466. tor=0.2,
467. tor=0.2,
468. tor=0.2,
469. tor=0.2,
470. tor=0.2,
471. tor=0.2,
472. tor=0.2,
473. tor=0.2,
474. tor=0.2,
475. tor=0.2,
476. tor=0.2,
477. tor=0.2,
478. tor=0.2,
479. tor=0.2,
480. tor=0.2,
481. tor=0.2,
482. tor=0.2,
483. tor=0.2,
484. tor=0.2,
485. tor=0.2,
486. tor=0.2,
487. tor=0.2,
488. tor=0.2,
489. tor=0.2,
490. tor=0.2,
491. tor=0.2,
492. tor=0.2,
493. tor=0.2,
494. tor=0.2,
495. tor=0.2,
496. tor=0.2,
497. tor=0.2,
498. tor=0.2,
499. tor=0.2,
500. tor=0.2,
501. tor=0.2,
502. tor=0.2,
503. tor=0.2,
504. tor=0.2,
505. tor=0.2,
506. tor=0.2,
507. tor=0.2,
508. tor=0.2,
509. tor=0.2,
510. tor=0.2,
511. tor=0.2,
512. tor=0.2,
513. tor=0.2,
514. tor=0.2,
515. tor=0.2,
516. tor=0.2,
517. tor=0.2,
518. tor=0.2,
519. tor=0.2,
520. tor=0.2,
521. tor=0.2,
522. tor=0.2,
523. tor=0.2,
524. tor=0.2,
525. tor=0.2,
526. tor=0.2,
527. tor=0.2,
528. tor=0.2,
529. tor=0.2,
530. tor=0.2,
531. tor=0.2,
532. tor=0.2,
533. tor=0.2,
534. tor=0.2,
535. tor=0.2,
536. tor=0.2,
537. tor=0.2,
538. tor=0.2,
539. tor=0.2,
540. tor=0.2,
541. tor=0.2,
542. tor=0.2,
543. tor=0.2,
544. tor=0.2,
545. tor=0.2,
546. tor=0.2,
547. tor=0.2,
548. tor=0.2,
549. tor=0.2,
550. tor=0.2,
551. tor=0.2,
552. tor=0.2,
553. tor=0.2,
554. tor=0.2,
555. tor=0.2,
556. tor=0.2,
557. tor=0.2,
558. tor=0.2,
559. tor=0.2,
560. tor=0.2,
561. tor=0.2,
562. tor=0.2,
563. tor=0.2,
564. tor=0.2,
565. tor=0.2,
566. tor=0.2,
567. tor=0.2,
568. tor=0.2,
569. tor=0.2,
570. tor=0.2,
571. tor=0.2,
572. tor=0.2,
573. tor=0.2,
574. tor=0.2,
575. tor=0.2,
576. tor=0.2,
577. tor=0.2,
578. tor=0.2,
579. tor=0.2,
580. tor=0.2,
581. tor=0.2,
582. tor=0.2,
583. tor=0.2,
584. tor=0.2,
585. tor=0.2,
586. tor=0.2,
587. tor=0.2,
588. tor=0.2,
589. tor=0.2,
590. tor=0.2,
591. tor=0.2,
592. tor=0.2,
593. tor=0.2,
594. tor=0.2,
595. tor=0.2,
596. tor=0.2,
597. tor=0.2,
598. tor=0.2,
599. tor=0.2,
600. tor=0.2,
601. tor=0.2,
602. tor=0.2,
603. tor=0.2,
604. tor=0.2,
605. tor=0.2,
606. tor=0.2,
607. tor=0.2,
608. tor=0.2,
609. tor=0.2,
610. tor=0.2,
611. tor=0.2,
612. tor=0.2,
613. tor=0.2,
614. tor=0.2,
615. tor=0.2,
616. tor=0.2,
617. tor=0.2,
618. tor=0.2,
619. tor=0.2,
620. tor=0.2,
621. tor=0.2,
622. tor=0.2,
623. tor=0.2,
624. tor=0.2,
625. tor=0.2,
626. tor=0.2,
627. tor=0.2,
628. tor=0.2,
629. tor=0.2,
630. tor=0.2,
631. tor=0.2,
632. tor=0.2,
633. tor=0.2,
634. tor=0.2,
635. tor=0.2,
636. tor=0.2,
637. tor=0.2,
638. tor=0.2,
639. tor=0.2,
640. tor=0.2,
641. tor=0.2,
642. tor=0.2,
643. tor=0.2,
644. tor=0.2,
645. tor=0.2,
646. tor=0.2,
647. tor=0.2,
648. tor=0.2,
649. tor=0.2,
650. tor=0.2,
651. tor=0.2,
652. tor=0.2,
653. tor=0.2,
654. tor=0.2,
655. tor=0.2,
656. tor=0.2,
657. tor=0.2,
658. tor=0.2,
659. tor=0.2,
660. tor=0.2,
661. tor=0.2,
662. tor=0.2,
663. tor=0.2,
664. tor=0.2,
665. tor=0.2,
666. tor=0.2,
667. tor=0.2,
668. tor=0.2,
669. tor=0.2,
670. tor=0.2,
671. tor=0.2,
672. tor=0.2,
673. tor=0.2,
674. tor=0.2,
675. tor=0.2,
676. tor=0.2,
677. tor=0.2,
678. tor=0.2,
679. tor=0.2,
680. tor=0.2,
681. tor=0.2,
682. tor=0.2,
683. tor=0.2,
684. tor=0.2,
685. tor=0.2,
686. tor=0.2,
687. tor=0.2,
688. tor=0.2,
689. tor=0.2,
690. tor=0.2,
691. tor=0.2,
692. tor=0.2,
693. tor=0.2,
694. tor=0.2,
695. tor=0.2,
696. tor=0.2,
697. tor=0.2,
698. tor=0.2,
699. tor=0.2,
700. tor=0.2,
701. tor=0.2,
702. tor=0.2,
703. tor=0.2,
704. tor=0.2,
705. tor=0.2,
706. tor=0.2,
707. tor=0.2,
708. tor=0.2,
709. tor=0.2,
710. tor=0.2,
711. tor=0.2,
712. tor=0.2,
713. tor=0.2,
714. tor=0.2,
715. tor=0.2,
716. tor=0.2,
717. tor=0.2,
718. tor=0.2,
719. tor=0.2,
720. tor=0.2,
721. tor=0.2,
722. tor=0.2,
723. tor=0.2,
724. tor=0.2,
725. tor=0.2,
726. tor=0.2,
727. tor=0.2,
728. tor=0.2,
729. tor=0.2,
730. tor=0.2,
731. tor=0.2,
732. tor=0.2,
733. tor=0.2,
734. tor=0.2,
735. tor=0.2,
736. tor=0.2,
737. tor=0.2,
738. tor=0.2,
739. tor=0.2,
740. tor=0.2,
741. tor=0.2,
742. tor=0.2,
743. tor=0.2,
744. tor=0.2,
745. tor=0.2,
746. tor=0.2,
747. tor=0.2,
748. tor=0.2,
749. tor=0.2,
750. tor=0.2,
751. tor=0.2,
752. tor=0.2,
753. tor=0.2,
754. tor=0.2,
755. tor=0.2,
756. tor=0.2,
757. tor=0.2,
758. tor=0
```

```

440. sheet1.write(0,4, 'LR')
441. sheet1.write(0,5, 'CONV layers')
442. sheet1.write(0,6, 'Filters')
443. sheet1.write(0,7, 'Filter sizes')
444. sheet1.write(0,8, 'Dense layers')
445. sheet1.write(0,9, 'Dense size')
446. sheet1.write(0,10, 'Dense dropout')
447. sheet1.write(0,11, 'Regularization')
448. sheet1.write(0,12, 'metric')
449. sheet1.write(0,13, 'Average result')
450. for i in range(_epochs+1):
451.     sheet1.write(0,i+14, 'epoch '+str(i))
452. wb.save(excel_name)
453.
454.
455. #Train model
456. for _batch_size in batch_sizes:
457.     for _learning_rate in learning_rates:
458.         for _optimizer in optimizers:
459.             for _filter in filters:
460.                 for _filter_size in filter_sizes:
461.                     for _dropout in dropouts:
462.                         for _conv_layer in conv_layers:
463.                             for _reg in regs:
464.                                 for _dense_layer in dense_layers:
465.                                     for _dense_size in dense_sizes:
466.                                         if _optimizer=='ADAM':
467.                                             opt = Adam(learn-
468. ing_rate=_learning_rate)
469.                                         if _optimizer=='SGD':
470.                                             opt = SGD(learn-
471. ing_rate=_learning_rate,
472. momentum=momen-
473. tum=0.9, nesterov=False)
474.                                         model = build_cnn_mo-
475. del(dense_size=_dense_size,
476. dense_la
477. yer=_dense_layer,
478. reg=_reg
479. ,
480. conv_lay
481. er=_conv_layer,
482. train_em
483. bed = False,
484. fil-
485. ter=_filter,
486. fil-
487. ter_size=_filter_size,
488. drop-
489. out=_dropout)
490. model.summary()
491. print(_learning_rate)
492.
493. model.compile(opti-
494. mizer=opt, loss=tf.keras.losses.SparseCategoricalCrossentropy(), met-
495. rics=['acc']) #loss='sparse_categorical_crossentropy'
496. train_begin=model.evalu-
497. ate(x_train_sequence, y_train_num, verbose=0)
498. val_begin=model.evalu-
499. ate(x_val_sequence, y_val_num, verbose=0)
500. history = model.fit(x_train_se-
501. quence, y_train_num,
502. epochs=_epochs,

```



```
487.                                     batch_size=_batch_size,
488.                                     validation_data=(x_val_se-
    quence, y_val_num),
489.                                     shuffle=True,
490.                                     callbacks=callbacks_list)
491.
492.                                     #Save training
493.                                     wb=copy(open_workbook(ex-
    cel_name))
494.                                     row = to_ex-
    cel(wb.get_sheet(0), embedding_type, _batch_size, _learning_rate, _opti-
    mizer, _filter, _filter_size, _drop-
    out, _conv_layer, _reg, _dense_layer, _dense_size, history.his-
    tory, train_begin, val_begin, row)
495.                                     wb.save(excel_name)
496.                                     print(row)
497.                                     #Plot training
498.                                     #plot(his-
    tory, train_begin, val_begin)
499.
500.    model.save('model_save.h5')
```

Appendix O: Code for augmentation

Input dropout method

```

1. ##Function to tokenize tokens in sentences:
2. import stanza
3. def token_tokenize(_class):
4.     nlp = stanza.Pipeline(lang='da', processors='tokenize',
5.                             tokenize_no_sspl=True)
6.     tokenized_sentences=[]
7.     for i in range(len(_class)): #For each row
8.         doc = nlp(_class[i])
9.         for i, sentence in enumerate(doc.sentences):
10.            tokens_list = []
11.            for token in sentence.tokens:
12.                tokens_list.append(token.text)
13.            tokenized_sentences.append(tokens_list[:])
14.     return tokenized_sentences
15. #Import sentence dataset
16. import pandas as pd
17. data = pd.read_excel('Sentence_dataset_samples.xlsx')
18. print(data.head(2))
19. data['Sample'] = data['Sample'].str.strip()
20. print(data.head(2))
21. data = data[data.Set=='train'] #Only train data
22. x_train= [sentence for sentence in data.Sample[data.Label=='positive']] #only positive data labels
23.
24. #Tokenize dataset
25. x_train_tokenized = token_tokenize(x_train)
26.
27. #Drop random words
28. import random
29. x_train_drop = []
30. for sentence in x_train_tokenized:
31.     if len(sentence)<8: #Drop one word
32.         rand = random.randrange(0,len(sentence))
33.         sentence = sentence[0:rand]+sentence[rand+1:]
34.     elif len(sentence)<15: #Drop two words
35.         rand = random.randrange(0,len(sentence))
36.         sentence = sentence[0:rand]+sentence[rand+1:]
37.         rand = random.randrange(0,len(sentence))
38.         sentence = sentence[0:rand]+sentence[rand+1:]
39.     elif len(sentence)>14: #Drop three words
40.         rand = random.randrange(0,len(sentence))
41.         sentence = sentence[0:rand]+sentence[rand+1:]
42.         rand = random.randrange(0,len(sentence))
43.         sentence = sentence[0:rand]+sentence[rand+1:]
44.         rand = random.randrange(0,len(sentence))
45.         sentence = sentence[0:rand]+sentence[rand+1:]
46.
47.     #Make as one string
48.     string=''
49.     for token in sentence:
50.         string+=token+' '
51.     x_train_drop.append(string)
52. x_train_drop

```

Backtranslation method

```
1. from googletrans import Translator
2. translator = Translator()
3. def trans(sentence_list):
4.     translated_da=[]
5.     translated_en=[]
6.     for sentence in sentence_list:
7.         text = translator.translate(sentence) #Trans to english
8.         translated_en.append(text.text)
9.         text = translator.trans-
10.         late(text.text, src='en', dest='da') #Trans english to danish
11.         print(text.text)
12.         translated_da.append(text.text)
13.     return translated_da, translated_en
```

Appendix P: Code for Grad-CAM

```

1. #####Functions for classifying broedtext
2. import stanza #Tokenizer library
3. from tensorflow.keras.preprocessing.sequence import pad_sequences
4.
5. #Sentence tokenize the text
6. def sentence_tokenize(text):
7.     nlp = stanza.Pipeline(lang='da', processors='tokenize')
8.     doc = nlp(text)
9.     tokenized_sentences=[]
10.    for i, sentence in enumerate(doc.sentences):
11.        tokens_list=''
12.        for token in sentence.tokens:
13.            if token.text=='.':
14.                tokens_list+=token.text#.append(token.text)
15.            else:
16.                tokens_list+=' '+token.text
17.
18.        tokenized_sentences.append(tokens_list[:])
19.    tokenized_sentences = [sent.strip() for sent in tokenized_sentences]
20.    return tokenized_sentences
21.
22. ##Tokenize tokens in sentences:
23. def token_tokenize(_class):
24.     nlp = stanza.Pipeline(lang='da', processors='tokenize', to-
25.        kenize_no_sspl=True)
26.     tokenized_sentences=[]
27.     for i in range(len(_class)): #For each row
28.         doc = nlp(_class[i])
29.         for i, sentence in enumerate(doc.sentences):
30.             tokens_list = []
31.             for token in sentence.tokens:
32.                 tokens_list.append(token.text)
33.             tokenized_sentences.append(tokens_list[:])
34.     return tokenized_sentences
35.
36. def text_to_seq(tokenized_text):
37.     text_sequence=[]
38.     for sentence in tokenized_text:
39.         subsentence=[] #Subsentence
40.         for word in sentence:
41.             if word_index.get(word)==None: #If it is OOV word ap-
42.                 pend UNK value to string
43.                 subsentence.append(word_index['UNK'])
44.             else: #Else append the value
45.                 subsentence.append(word_index[word])
46.         text_sequence.append(subsentence)
47.     return text_sequence
48.
49. ##Classifier
50. def classify(text, model, max_length_sample):
51.     text_class = ['negative', 'positive'] #for print function
52.     x = token_tokenize(text)
53.     x = text_to_seq(x)
54.     x = pad_sequences(x, maxlen=max_length_sam-
55.        ple) #slet for lstm? ikke for cnn
56.     x = np.asarray(x)
57.     x = model.predict(x)
58.     predictions = [text_class[np.argmax(sample)] for sample in x]
59.     return predictions, x

```

```

57.
58.
59. ###-----###
60.
61. #Create test sentence
62. import numpy as np
63.
64. # Load word_index dictionary from pretrained model
65. word_index = np.load('word_index_dictionary.npy', allow_pickle='TRUE').item()
66.
67. #Write sentence to visualize
68. sentence = 'Ukompliseret ned til duodenum 2. stykke hvor man ser det uregelmæssige papilkompleks.'
69. print(sentence)
70. sentence_tokenized = token_tokenize([sentence])
71. print(sentence_tokenized)
72. a = text_to_seq(sentence_tokenized)
73. print(a)
74. if len(a[0]) < 100: #Only pad if smaller than 100, keep full sentence otherwise
75.     a = pad_sequences(a, maxlen=100, padding='post', truncating='post')
76. print(a)
77. a = np.asarray(a)
78.
79.
80. #Load pretrained model
81. import tensorflow as tf
82. from tensorflow.keras.models import load_model
83. import pathlib
84. filepath = str(pathlib.Path().absolute())
85. model1 = load_model(filepath + '/CNN_weights/weights.0.2357-62-0.9168.hdf5')
86. model1.summary()
87.
88. #Grad-CAM calculation
89. import math
90. string = ['negative', 'positive']
91. extraction_layer = 5 #Which layer to perform Grad-CAM on e.g layer 4 is conv layer with filtersize 1
92.
93. for model in [model1]:
94.     #Create model that outputs convolution activation and predictions
95.     grad_model = tf.keras.models.Model([model.inputs], [model.layers[extraction_layer].output, model.output])
96.
97.     #Use gradient tape to track gradients
98.     with tf.GradientTape() as tape:
99.         #Extract convolution activation and prediction
100.         conv_outputs, predictions = grad_model(a)
101.         #Get softmax prediction for predicted class
102.         pred = predictions[:, np.argmax(predictions)]
103.         print(pred)
104.         output = conv_outputs[0] #output of conv layer (feature maps) of size (sentence_length, n_filters)
105.
106.         #Gradients for all scalar values in all featuremaps size (sentence_length, n_filters)
107.         grads = tape.gradient(pred, conv_outputs)[0]
108.
109.         #Take mean of each gradient filter map (its importance) shape (n_filters,)
110.         weights = tf.reduce_mean(grads, axis=0)

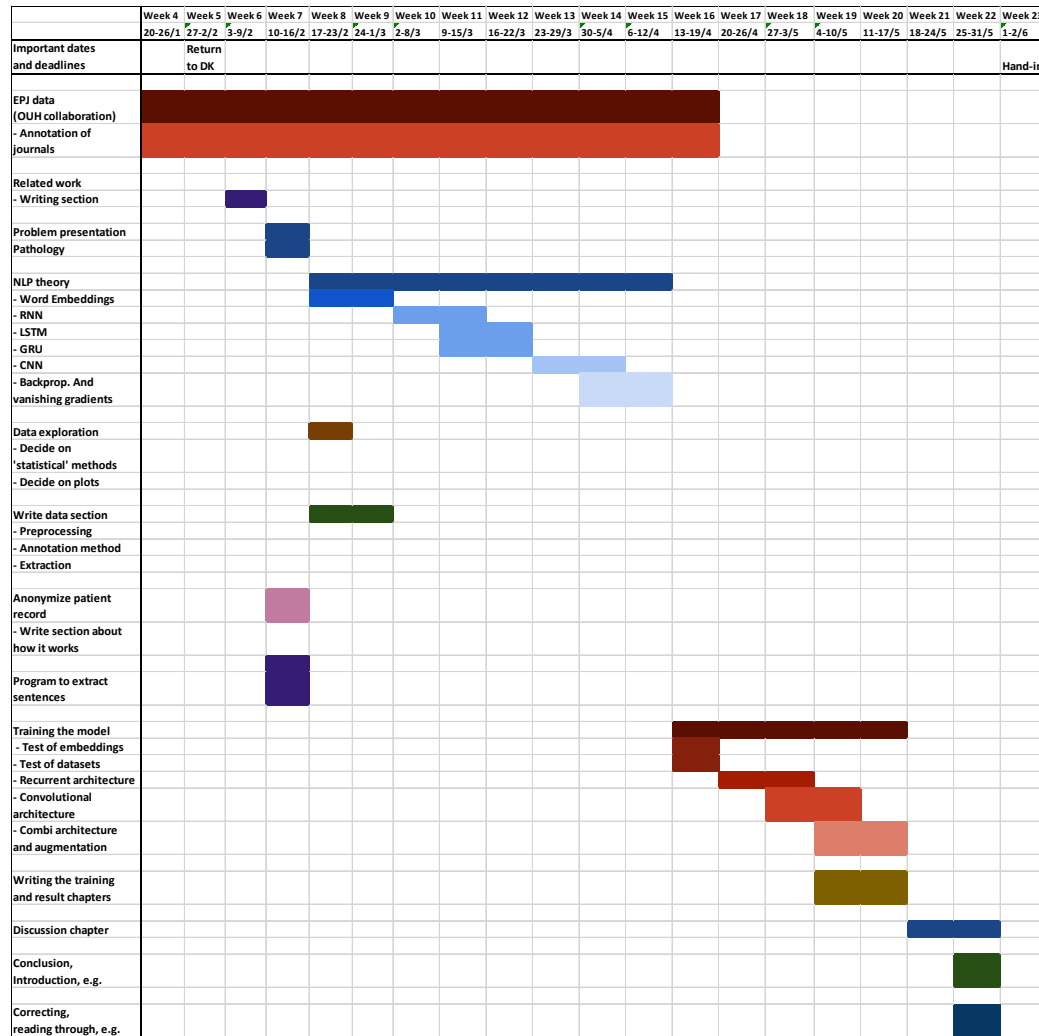
```

```
111.  
112.     #create heatmap of zeros shape (sentence_length)  
113.     cam = np.zeros(output.shape[0:1], dtype = np.float32)  
114.  
115.     #Scale all featuremaps with its importance  
116.     for i, w in enumerate(weights):  
117.         cam+=w*output[:,i]  
118.  
119.     #cam=cv2.resize(cam.numpy(), (100,1)) #resize if downscaled  
120.     print(cam)  
121.  
122.     #Relu  
123.     cam = np.maximum(cam, 0)  
124.  
125.     #As percentage  
126.     cam_percentage = cam/sum(cam)  
127.     #do not care about percentage<1%  
128.     cam_percentage = cam_percentage*(cam_percentage>0.01)  
129.  
130.     print(sentence, "-->",string[np.argmax(predictions)], predic-  
        tions.numpy()[0], "\n",sentence_tokenized, "\n", np.round(cam_per-  
        sentage[:len(sentence_tokenized[0])],3))#)  
131.  
132.     #Spread values between 0 and 1, for visual image  
133.     heatmap = (cam - cam.min()) / (cam.max() - cam.min())
```

September - February

[illegible]

March - June



Details

The time plan has been split in two: The months from September until and including February, and the months from March until and including the hand-in deadline in June.

The thesis work was also in practice split in these two periods. The first period was dedicated mainly to acquiring the data and researching related work and machine learning techniques to be used. The second period was dedicated to the theory, training, and analyzing of the results.

The project went according to plan except for three minor changes:

- When starting the project, it was intended to first annotate all bleeding occurrences, then train a segmentation model to identify those, after which only relevant bleedings would be annotated, and identified using

a new model. There were several challenges connected to deciding which bleedings were relevant so it was dropped quickly because of time constraints.

- The quality control after the data acquisition showed some flaws in the extraction method that needed changing. This and the annotation process itself took longer than expected. The problem was worked around by researching and writing the theory sections during the annotation process. Likewise, the sample extraction methods were developed on a subset of the annotated data to be ready for training as soon as all data was annotated.
- The collaboration with SAS did not work out as planned because of a misunderstanding. Some weeks during the first period of the project were spent on studying their machine learning platform which was finally not used for training.

Appendix R: Attached files

The following files have been attached in a ZIP-file:

- Calculation of kappa score
- Results for test of word embeddings
- Results for test of datasets

References

- [1] Unknown, "Tromboseprofylakse, medicinske tilstande," *Sundhed.dk*, 2020.
<https://www.sundhed.dk/sundhedsfaglig/laegehaandbogen/hjertekar/tilstande-og-sygdomme/tromboembolisk-sygdom/tromboseprofylakse-medicin/> (accessed Apr. 21, 2020).
- [2] J. A. Heit *et al.*, "Incidence of venous thromboembolism in hospitalized patients vs community residents," *Mayo Clin. Proc.*, 2001, doi: 10.4065/76.11.1102.
- [3] F. A. Spencer, D. Lessard, C. Emery, G. Reed, and R. J. Goldberg, "Venous thromboembolism in the outpatient setting," *Arch. Intern. Med.*, 2007, doi: 10.1001/archinte.167.14.1471.
- [4] A. Leizorovicz, A. T. Cohen, A. G. G. Turpie, C. G. Olsson, P. T. Vaitkus, and S. Z. Goldhaber, "Randomized, placebo-controlled trial of dalteparin for the prevention of venous thromboembolism in acutely ill medical patients," *Circulation*, 2004, doi: 10.1161/01.CIR.0000138928.83266.24.
- [5] M. M. Samama *et al.*, "A comparison of enoxaparin with placebo for the prevention of venous thromboembolism in acutely ill medical patients," *N. Engl. J. Med.*, 1999, doi: 10.1056/NEJM199909093411103.
- [6] K. K. Sogaard, M. Schmidt, L. Pedersen, E. Horváth-Puhó, and H. T. Sørensen, "30-Year mortality after venous thromboembolism a population-based cohort study," *Circulation*, 2014, doi: 10.1161/CIRCULATIONAHA.114.009107.
- [7] C. Kearon *et al.*, "Antithrombotic therapy for VTE disease: Antithrombotic therapy and prevention of thrombosis, 9th ed: American College of Chest Physicians evidence-based clinical practice guidelines," *Chest*, 2012, doi: 10.1378/chest.11-2301.
- [8] H. Decousus *et al.*, "Factors at admission associated with bleeding risk in medical patients: Findings from the improve investigators," *Chest*, 2011, doi: 10.1378/chest.09-3081.
- [9] F. for L. til A. Forebyggelse, samt L. til trombocythæmning hos Behandling, P. med C. Lidelser, and under R. for A. af D. S. Sygehusmedicin, "RADS baggrundsnotat vedr. farmakologisk tromboseprofylakse til medicinske patienter," 2015. [Online]. Available: <https://www.regioner.dk/media/2080/bgn-tromboseprofylakse-med-pt-12082015-ny-til-hjemmesiden.pdf>.

-
- [10] A. C. Spyropoulos *et al.*, “Predictive and associative models to identify hospitalized medical patients at risk for VTE,” *Chest*, 2011, doi: 10.1378/chest.10-1944.
 - [11] A. Khorana, N. M. Kuderer, E. Culakova, G. H. Lyman, and C. W. Francis, “Development and validation of a predictive model for chemotherapy- associated thrombosis,” *Blood*, 2008, doi: 10.1182/blood-2007-10-116327.
 - [12] A. Amin, S. Stemkowski, J. Lin, and G. Yang, “Thromboprophylaxis rates in US medical centers: Success or failure?,” *J. Thromb. Haemost.*, 2007, doi: 10.1111/j.1538-7836.2007.02650.x.
 - [13] J. P. Rwabihama *et al.*, “Prophylaxis of Venous Thromboembolism in Geriatric Settings: A Cluster-Randomized Multicomponent Interventional Trial,” *J. Am. Med. Dir. Assoc.*, 2018, doi: 10.1016/j.jamda.2018.02.004.
 - [14] A. Amin, S. Stemkowski, J. Lin, and G. Yang, “Appropriate thromboprophylaxis in hospitalized cancer patients.,” *Clin. Adv. Hematol. Oncol.*, 2008.
 - [15] W. H. Geerts *et al.*, “Prevention of venous thromboembolism: The Seventh ACCP Conference on Antithrombotic and Thrombolytic Therapy,” in *Chest*, 2004, doi: 10.1378/chest.126.3_suppl.338S.
 - [16] V. E. Valkhoff *et al.*, “Validation study in four health-care databases: Upper gastrointestinal bleeding misclassification affects precision but not magnitude of drug-related upper gastrointestinal bleeding risk,” *J. Clin. Epidemiol.*, 2014, doi: 10.1016/j.jclinepi.2014.02.020.
 - [17] Unknown, “Venous Thromboembolism (VTE).” <https://www.webmd.com/dvt/what-is-venous-thromboembolism#1> (accessed Apr. 22, 2020).
 - [18] J. Huizen, “What’s the Difference Between Thrombosis and Embolism?,” 2017. <https://www.healthline.com/health/thrombosis-vs-embolism> (accessed Apr. 21, 2020).
 - [19] Unknown, “DEEP VEIN THROMBOSIS AND PULMONARY EMBOLISM A GUIDE FOR PRACTITIONERS,” 2017. <http://www.healthinfi.com/deep-vein-thrombosis-and-pulmonary-embolism-a-guide-for-practitioners/> (accessed Apr. 22, 2020).
 - [20] N. Eldrup, “Dyb venetrombose (DVT),” 2019. <https://www.sundhed.dk/borger/patienthaandbogen/hjerte-og-blodkar/sygdomme/blodpropsygdom/dyb-venetrombose-dvt/> (accessed Apr. 22, 2020).
 - [21] J. Kjærgaard, “Blodprop i lungerne (lungeemboli),” 2019.

- <https://www.sundhed.dk/borger/patienthaandbogen/hjerte-og-blod-kar/sygdomme/blodpropsygdom/blodprop-i-lungerne-lungeemboli/> (accessed Apr. 22, 2020).
- [22] N. Eldrup, “Blodproptendens,” 2020.
<https://www.sundhed.dk/borger/patienthaandbogen/blod/sygdomme/oevrige-blodsygdomme/blodproptendens/> (accessed Apr. 22, 2020).
- [23] Unknown, “The Link Between Cancer, Blood Clots, and DVT.”
<https://www.webmd.com/dvt/deep-vein-thrombosis-cancer-risk#1> (accessed Apr. 22, 2020).
- [24] D. Sullivan, “Blood Clots After Surgery: Tips for Prevention,” 2018.
<https://www.healthline.com/health/how-to-prevent-blood-clots-after-surgery#surgery-risk-factors> (accessed Apr. 22, 2020).
- [25] K. Blake, “What You Need to Know About Hemorrhage,” 2019.
automatically (accessed May 27, 2020).
- [26] M. Taggart *et al.*, “Comparison of 2 Natural Language Processing Methods for Identification of Bleeding Among Critically Ill Patients,” *JAMA Netw. open*, 2018, doi: 10.1001/jamanetworkopen.2018.3451.
- [27] R. Li *et al.*, “Detection of bleeding events in electronic health record notes using convolutional neural network models enhanced with recurrent neural network autoencoders: Deep learning approach,” *J. Med. Internet Res.*, vol. 21, no. 2, pp. 1–10, 2019, doi: 10.2196/10788.
- [28] Z. Tian, S. Sun, T. Egualé, and C. M. Rochefort, “Automated extraction of vte events from narrative radiology reports in electronic health records: A validation study,” *Med. Care*, 2017, doi: 10.1097/MLR.0000000000000346.
- [29] S. Santiso, A. Pérez, and A. Casillas, “Exploring Joint AB-LSTM with Embedded Lemmas for Adverse Drug Reaction Discovery,” *IEEE J. Biomed. Heal. Informatics*, 2019, doi: 10.1109/JBHI.2018.2879744.
- [30] C. M. Rochefort, A. D. Verma, T. Egualé, T. C. Lee, and D. L. Buckeridge, “A novel method of adverse event detection can accurately identify venous thromboembolisms (VTEs) from narrative electronic health record data,” *J. Am. Med. Informatics Assoc.*, 2015, doi: 10.1136/amiajnl-2014-002768.
- [31] L. Chen, “Assertion Detection in Clinical Natural Language Processing: A Knowledge-Poor Machine Learning Approach,” in *2019 IEEE 2nd International Conference on Information and Computer Technologies, ICICT 2019*, 2019, doi: 10.1109/INFOCT.2019.8710921.
- [32] D. Chen, G. Qian, and Q. Pan, “Breast Cancer Classification with Electronic Medical Records Using Hierarchical Attention Bidirectional

- Networks,” in *Proceedings - 2018 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2018*, 2019, doi: 10.1109/BIBM.2018.8621479.
- [33] Y. Deng, P. Dolog, J. M. Gass, and K. Denecke, “Obesity entity extraction from real outpatient records: When learning-based methods meet small imbalanced medical data sets,” in *Proceedings - IEEE Symposium on Computer-Based Medical Systems*, 2019, doi: 10.1109/CBMS.2019.00087.
- [34] F. Li, Y. Jin, W. Liu, B. P. S. Rawat, P. Cai, and H. Yu, “Fine-tuning bidirectional encoder representations from transformers (BERT)-based models on large-scale electronic health record notes: An empirical study,” *J. Med. Internet Res.*, 2019, doi: 10.2196/14830.
- [35] M. Tien *et al.*, “Retrospective derivation and validation of an automated electronic search algorithm to identify post operative cardiovascular and thromboembolic complications,” *Applied Clinical Informatics*. 2015, doi: 10.4338/ACI-2015-03-RA-0026.
- [36] K. Rajput, G. Chetty, and R. Davey, “Deep neural models for chronic disease status detection in free text clinical records,” in *IEEE International Conference on Data Mining Workshops, ICDMW*, 2019, doi: 10.1109/ICDMW.2018.00127.
- [37] F. Chollet, *Deep Learning with Phyton*. 2018.
- [38] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global vectors for word representation,” in *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 2014, doi: 10.3115/v1/d14-1162.
- [39] F. Chaubard, M. Fang, G. Genthial, R. Mundra, and R. Socher, “CS224n: Natural Language Processing with Deep Learning,” 2017. [Online]. Available: https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/lecture_notes/cs224n-2017-notes1.pdf.
- [40] H. Lane, H. Cole, and H. M. Hapke, “Natural Language Processing in Action MEAP Edition,” *Manning Publ.*, 2018, doi: 10.1310/sci2102-166.
- [41] Facebook, “<https://fasttext.cc/>.” <https://fasttext.cc/>.
- [42] CommonCrawl, “No Title.” <https://commoncrawl.org/>.
- [43] “Wikipedia.” <https://www.wikipedia.org/>.
- [44] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, “Enriching Word Vectors with Subword Information,” *Trans. Assoc. Comput. Linguist.*, 2017, doi: 10.1162/tacl_a_00051.

-
- [45] J. Pennington, R. Socher, and C. D. Manning, “GloVe,” 2014. <https://nlp.stanford.edu/projects/glove/> (accessed May 31, 2020).
 - [46] K. Kurita, “Paper Dissected: ‘Glove: Global Vectors for Word Representation’ Explained,” 2018. <https://mlexplained.com/2018/04/29/paper-dissected-glove-global-vectors-for-word-representation-explained/> (accessed May 27, 2020).
 - [47] M. Rasmussen and N. Berggrein, “Named Entity Recognition and Disambiguation in Danish Electronic Health Records,” IT-University Copenhagen, 2019.
 - [48] K. Pantazos, S. Lauesen, and S. Lippert, “Preserving medical correctness, readability and consistency in de-identified health records,” *Health Informatics J.*, 2017, doi: 10.1177/1460458216647760.
 - [49] S. Raaijmakers, *Deep Learning for Natural Language Processing*, 3rd ed. Manning Publications, 2019.
 - [50] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, “Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization,” *Int. J. Comput. Vis.*, 2020, doi: 10.1007/s11263-019-01228-7.
 - [51] P. Gudikandula, “Recurrent Neural Networks and LSTM explained,” 2019. <https://medium.com/@purnasaigudikandula/recurrent-neural-networks-and-lstm-explained-7f51c7f6bbb9> (accessed May 25, 2019).
 - [52] M. Mohammadi, R. Mundra, R. Socher, and L. Wang, “CS224n: Natural Language Processing with Deep Learning,” 2017. https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1174/lecture_notes/cs224n-2017-notes5.pdf (accessed May 28, 2020).
 - [53] K. Cho *et al.*, “Learning phrase representations using RNN encoder-decoder for statistical machine translation,” in *EMNLP 2014 - 2014 Conference on Empirical Methods in Natural Language Processing, Proceedings of the Conference*, 2014, doi: 10.3115/v1/d14-1179.
 - [54] “RNN, LSTM & GRU,” 2019. <http://dprogrammer.org/rnn-lstm-gru> (accessed May 25, 2020).
 - [55] J. Chung, “Gated Recurrent Neural Networks on Sequence Modeling arXiv: 1412.3555v1 [cs.LG] 11 Dec 2014,” *Int. Conf. Mach. Learn.*, 2015.
 - [56] S. Hochreiter and J. Schmidhuber, “Long Short-Term Memory,” *Neural Comput.*, 1997, doi: 10.1162/neco.1997.9.8.1735.
 - [57] C. Olah, “Understanding LSTM Networks,” 2015. <https://colah.github.io/posts/2015-08-Understanding-LSTMs/> (accessed May 25, 2020).

-
- [58] “Natural Language Processing with Deep Learning,” 2019.
<http://web.stanford.edu/class/cs224n/slides/cs224n-2020-lecture07-fancy-rnn.pdf> (accessed May 25, 2020).
 - [59] D. Britz, A. Goldie, M. T. Luong, and Q. V. Le, “Massive exploration of neural machine translation architectures,” in *EMNLP 2017 - Conference on Empirical Methods in Natural Language Processing, Proceedings*, 2017, doi: 10.18653/v1/d17-1151.
 - [60] P. JAUMIER, “Backpropagation in a convolutional layer,” 2019.
<https://towardsdatascience.com/backpropagation-in-a-convolutional-layer-24c8d64d8509> (accessed May 29, 2020).
 - [61] R. Pascanu, T. Mikolov, and Y. Bengio, “On the difficulty of training recurrent neural networks,” in *30th International Conference on Machine Learning, ICML 2013*, 2013.
 - [62] A. M. Saxe, J. L. McClelland, and S. Ganguli, “Exact solutions to the nonlinear dynamics of learning in deep linear neural networks,” in *2nd International Conference on Learning Representations, ICLR 2014 - Conference Track Proceedings*, 2014.
 - [63] “Recurrent layers,” 2019.
https://keras.io/api/layers/recurrent_layers/ (accessed May 28, 2020).
 - [64] S. Merity, “Explaining and illustrating orthogonal initialization for recurrent neural networks,” 2016.
https://smerity.com/articles/2016/orthogonal_init.html (accessed May 27, 2020).
 - [65] X. Glorot and Y. Bengio, “Understanding the difficulty of training deep feedforward neural networks,” in *Journal of Machine Learning Research*, 2010.
 - [66] “CS2321n Convolutional Neural Networks for Visual Recognition.”
<https://cs231n.github.io/neural-networks-2/#init> (accessed May 25, 2020).
 - [67] D. Godoy, “Hyper-parameters in Action! Part II — Weight Initializers,” 2018. <https://towardsdatascience.com/hyper-parameters-in-action-part-ii-weight-initializers-35aee1a28404> (accessed May 25, 2020).
 - [68] “tf.keras.regularizers.l2,” 2019.
https://www.tensorflow.org/api_docs/python/tf/keras/regularizers/l2 (accessed May 25, 2020).
 - [69] D. Wu, “L2 Regularization and Batch Norm,” 2019.
<https://blog.janestreet.com/l2-regularization-and-batch-norm/>

- (accessed May 25, 2020).
- [70] “Overfit and underfit,” 2019.
https://www.tensorflow.org/tutorials/keras/overfit_and_underfit
(accessed May 25, 2020).
- [71] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, “Dropout: A simple way to prevent neural networks from overfitting,” *J. Mach. Learn. Res.*, 2014.
- [72] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” in *3rd International Conference on Learning Representations, ICLR 2015 - Conference Track Proceedings*, 2015.
- [73] S. Ruder, “An overview of gradient descent optimization algorithms,” *CoRR*, vol. abs/1609.0, 2018, [Online]. Available: <http://arxiv.org/abs/1609.04747>.
- [74] J. L. Fleiss, “Measuring nominal scale agreement among many raters,” *Psychol. Bull.*, 1971, doi: 10.1037/h0031619.
- [75] J. R. Landis and G. G. Koch, “The Measurement of Observer Agreement for Categorical Data,” *Biometrics*, 1977, doi: 10.2307/2529310.
- [76] P. Qi, Y. Zhang, Y. Zhang, J. Bolton, and C. Manning, D., “Stanza: A Python Natural Language Processing Toolkit for Many Human Languages,” *ArXiv Prepr.*, 2020, [Online]. Available: <https://arxiv.org/abs/2003.07082>.
- [77] “UD Danish DDT.”
https://universaldependencies.org/treebanks/da_ddt/index.html
(accessed May 25, 2020).
- [78] E. Klein, S. Bird, and E. Loper, *Natural Language Processing with Python*. O’Reilly Media Inc, 2009.
- [79] T. Kiss and J. Strunk, “Unsupervised multilingual sentence boundary detection,” *Comput. Linguist.*, 2006, doi: 10.1162/coli.2006.32.4.485.
- [80] V. Yogarajan, B. Pfahringer, and M. Mayo, “Automatic end-to-end De-identification: Is high accuracy the only metric?,” *ArXiv Prepr.*, 2019, [Online]. Available: <https://arxiv.org/pdf/1901.10583.pdf>.
- [81] A. Akbik, D. Blythe, and R. Vollgraf, “Contextual string embeddings for sequence labeling,” *Proc. 27th Int. Conf. Comput. Linguist.*, 2018.
- [82] “GitHub: DaNLP.” <https://github.com/alexandrainst/danlp> (accessed May 25, 2020).
- [83] R. Hvingelby and A. B. Pauli, “Named Entity Recognition.”
https://github.com/alexandrainst/danlp/blob/master/docs/models/ne_r.md (accessed May 25, 2020).

-
- [84] “Adam.” <https://keras.io/api/optimizers/adam/> (accessed May 25, 2020).
- [85] “Why is my training loss much higher than my testing loss?” https://keras.io/getting_started/faq/#why-is-my-training-loss-much-higher-than-my-testing-loss (accessed May 25, 2020).
- [86] S. Edunov, M. Ott, M. Auli, and D. Grangier, “Understanding back-translation at scale,” in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, 2020, doi: 10.18653/v1/d18-1045.
- [87] S. Schleifer, “Low Resource Text Classification with Backtranslation,” 2019, [Online]. Available: <https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15846784.pdf>.
- [88] “Model ensemble,” 2019. <https://cs231n.github.io/neural-networks-3/#ensemble> (accessed May 25, 2020).
- [89] L. Kaiji, P. Mardziel, F. Wu, and P. Amancharla, “Gender Bias in Neural Natural Language Processing,” *ArXiv Prepr.*, 2019, [Online]. Available: <https://arxiv.org/pdf/1807.11714.pdf>.
- [90] T. Bolukbasi, K. W. Chang, J. Zou, V. Saligrama, and A. Kalai, “Man is to computer programmer as woman is to homemaker? Debiasing word embeddings,” in *Advances in Neural Information Processing Systems*, 2016.
- [91] A. Conneau, H. Schwenk, Y. Le Cun, and L. Barrault, “Very deep convolutional networks for text classification,” in *15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017 - Proceedings of Conference*, 2017, doi: 10.18653/v1/e17-1104.
- [92] K. Kawamoto, C. A. Houlihan, E. A. Balas, and D. F. Lobach, “Improving clinical practice using clinical decision support systems: A systematic review of trials to identify features critical to success,” *British Medical Journal*. 2005, doi: 10.1136/bmj.38398.500764.8f.
- [93] E. S. Berner and T. J. La Lande, “Overview of Clinical Decision Support Systems,” 2007.
- [94] T. Ganegedara, “Intuitive Guide to Understanding GloVe Embeddings,” 2019. <https://towardsdatascience.com/light-on-math-ml-intuitive-guide-to-understanding-glove-embeddings-b13b4f19c010> (accessed May 28, 2020).
- [95] J. Johnson, “<http://cs231n.stanford.edu/handouts/derivatives.pdf>,” 2017. <http://cs231n.stanford.edu/handouts/derivatives.pdf> (accessed

- Jun. 01, 2020).
- [96] R. Wang, “Matrix norms,” 2015.
<http://fourier.eng.hmc.edu/e161/lectures/algebra/node12.html>
(accessed Jun. 01, 2020).
- [97] D. Cave, R. J. Saltzman, and C. A. Travis, “Evaluation of suspected small bowel bleeding (formerly obscure gastrointestinal bleeding),” *UpToDate*, 2015.
- [98] J. Barham, “Is It a Bruise or a Hematoma?,” 2019.
<https://www.verywellhealth.com/bruises-and-hematomas-4178410>
(accessed Apr. 21, 2019).
- [99] G. Rogers, “Von Willebrand Disease: Types, Causes, and Symptoms,” 2016. <https://www.healthline.com/health/von-willebrand-disease>
(accessed Apr. 22, 2020).
- [100] A. Kahn, “Hemophilia,” 2016.
<https://www.healthline.com/health/hemophilia> (accessed Apr. 22, 2020).