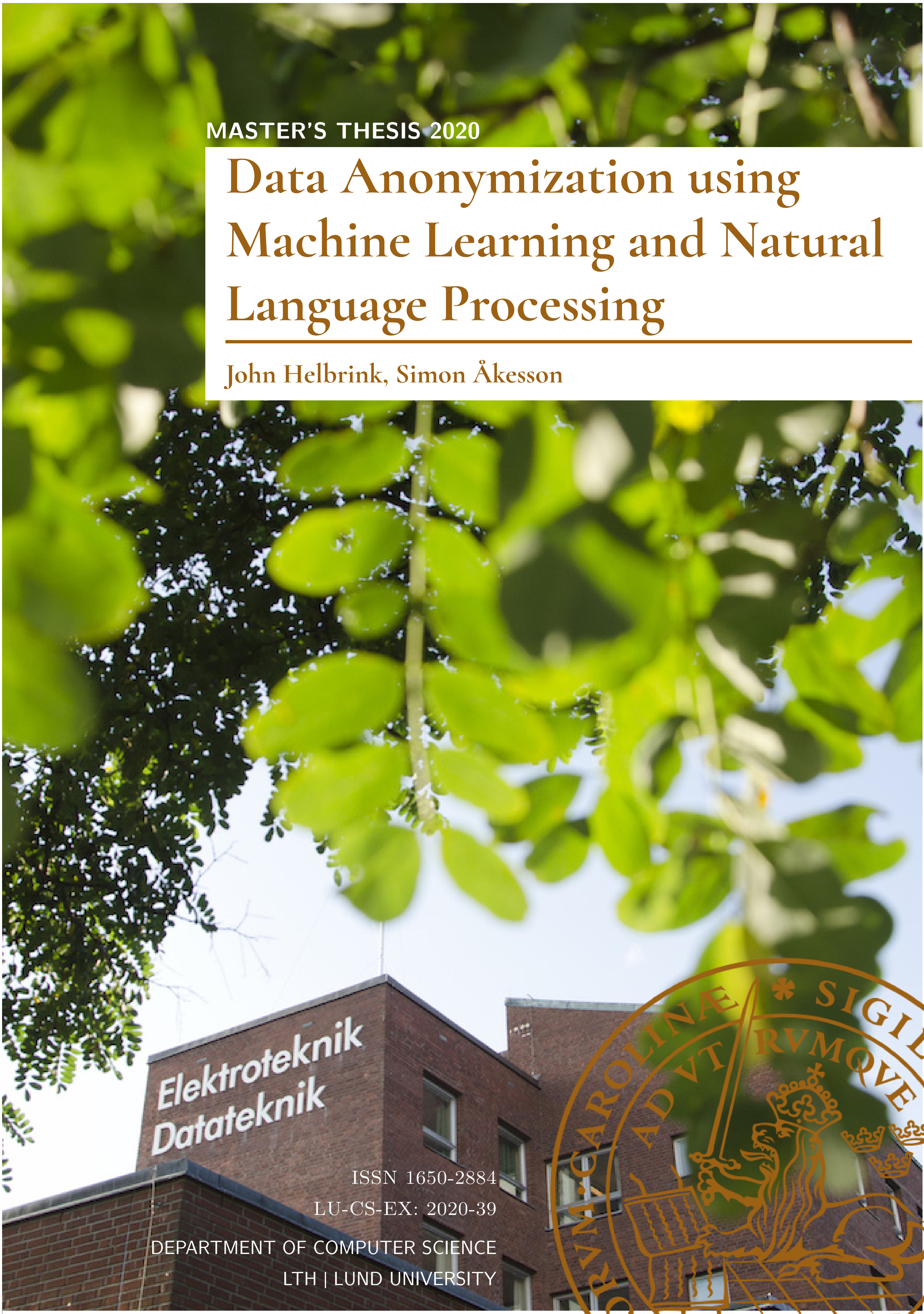


MASTER'S THESIS 2020

Data Anonymization using Machine Learning and Natural Language Processing

John Helbrink, Simon Åkesson

ISSN 1650-2884
LU-CS-EX: 2020-39

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2020-39

Data Anonymization using Machine Learning and Natural Language Processing

John Helbrink, Simon Åkesson

Data Anonymization using Machine Learning and Natural Language Processing

John Helbrink
john.helbrink@gmail.com

Simon Åkesson
simonakesson4@gmail.com

July 1, 2020

Master's thesis work carried out at Sinch AB.

Supervisors: Michael Truong, Jianhua Cao, Pierre Nugues michael.truong@sinch.com, jianhua.cao@sinch.com, pierre.nugues@cs.lth.se

Examiner: Jacek Malec, mailto:jacek.malec@cs.lth.se

Abstract

With the introduction of the General Data Protection Regulation (GDPR) the demand to protect personal identifiable information from unnecessary exposure is growing. The objective of this thesis is to explore and construct a system that will use natural language processing and machine learning to anonymize data. This will provide further analysis opportunities and additional protection for the individual person. We implemented and compared different neural network architectures and frameworks. Using a bidirectional Long Short-Term Memory network together with contextualized embeddings and conditional random fields, state-of-the-art results were reached with the best F_1 score at 95.58 %. This shows the importance of contextual awareness as well as the impact of pre-trained word embeddings. However, human supervision will still be needed for it to completely guarantee the anonymization of the messages. Ultimately, the resulting system fulfills its purpose of masking texts, removing the risk of unnecessary information exposure.

Keywords: Machine Learning, FLAIR, Named Entity Recognition, Natural Language Processing, LSTM

Acknowledgements

We would like to thank Sinch for providing us with a interesting topic for our thesis, as well as a friendly environment. A big thank you to our supervisor at the Department of Computer Science, Pierre Nugues for great feedback and innovative ideas regarding both our thesis work and the report. Additionally, we would like to thank Jianhua Cao and Michael Truong for providing us with guidance throughout our thesis work.

Contents

Chapter 1
Introduction

During recent years, digital privacy has been a growing concern for citizens all around the world. With the increased availability of smart devices, i.e units with some sort of processing capability, the amount of generated data has skyrocketed.

Access to enormous amounts of data has been said to be beneficial for customers, as it can be used to create a better user experience. However, if the data ends up in the wrong hands, it could potentially be dangerous, with possibilities of tracking an individual person's behavior. This scenario is dreadful to think about, and that is where de-identification would be a way to limit the risk of such abuse. This would result in the individual being protected.

The application of natural language processing combined with machine learning would one way to solve this automatic de-identification issue. This would make it possible to create a model that could identify the sensitive parts of the data and mask them out – creating an anonymous message, only containing data relevant when improving the user experience for a potential customer.

In 2016, the European Union passed a regulation bill to make it possible for the customer to control her/his data. Additionally, this bill unified the regulatory environment for businesses acting internationally within the European Union.

During this thesis, the focus will be put on using machine learning to identify the sensitive parts of a text-based message. This will make it possible to mask the identified information to avoid personal identifiable information exposure, thus allow better protection for the user.

1.1 Background

1.1.1 General Data Protection Regulation

On May 25th 2018, the European Parliament and the Council of the European Union implemented the General Data Protection Regulation. This is a regulation in EU law regarding data protection and privacy, covering the European Union and the European Economic Area.

The general objective of the regulation is to provide people with further control over their data. This was done by applying restrictions on organizations regarding the processing of personal data. Personal data effectively means any information relating to any identifiable natural person. Such information could for example be a name, ID-number, location, physical and mental health.

The regulation is applied to all organizations operating and providing services to European citizens. Any infringement can be detrimental to the organization in question. In January 2019, Google was issued a fine of €50 million due to failure to oblige to the regulations, provide sufficient information about the data consent policies, and lack of control of how the information was used (Rosemain, 2019). Evidently, the risks of abusing personal data are immense. Given the importance of personal data analysis in terms of innovation and revenue, there is a demand for an anonymization technique. This will prevent unnecessary leakage of personally identifiable information (PII) while being safe and efficient.

1.1.2 Data Anonymization

According to Vollmer (2019), the General Data Protection Regulation does not apply to anonymous data. Like mentioned previously, the principles of data protection apply to any information which can be linked to a natural person. However, data protection should not be applied to anonymous information due to it no longer being related to an identifiable natural person. This means that for this project to be successful, masking is needed to be done to remove the sensitive parts of a message. Table 1.1 shows this masking process, where the sensitive parts would be: location (LOC) and organization (ORG).

input = ['eu', 'rejects', 'german', 'call', 'to', 'boycott', 'british', 'lamb', '.']
output = [<ORG>, 'rejects', <LOC>, 'call', 'to', 'boycott', <LOC>, 'lamb', '.']

Table 1.1: Example of anonymization

1.1.3 Motivation

The motivation for writing our thesis regarding data anonymization through NLP and machine learning has many reasons. Both the areas of natural language processing and machine learning are evolving very quickly, with many actors pushing the boundaries a little bit further every year.

Combining these technologies with the fact that personal data is valued more than ever and containing more information than most of us would want to reveal. This makes up for an interesting approach on how to use new data-driven technologies, meeting the needs of data-hungry companies whilst protecting the individual who wants to keep its privacy.

With the help of machine learning and NLP, we believe that it is possible to construct a solution that can identify sensitive data and filter this out. Ultimately making it completely anonymous and harmless for the individual, yet powerful for the companies that want to create a better user experience.

The work will be a valuable addition to the way of how companies prevent unnecessary personally identifiable information exposure. Additionally, this will allow Sinch to conduct a further study regarding the performance of anonymized versus un-anonymized data.

1.1.4 Aim

The goal of this thesis was to create an automation process for cleaning text messages. All messages to be processed pass through Sinch's APIs. The aim is thus to anonymize text data with state-of-the-art results to provide further protection.

To solve the task, we followed a deep learning approach based on Tensorflow together with Keras and FLAIR. Utilizing these frameworks, together with a variety of different state-of-the-art NLP techniques, we strove to reach state-of-the-art results in the identification of sensitive data.

Firstly, we will explain the background of our project, mentioning the necessary information to be able and follow the later parts of our report. Following this brief background, we will explain the theoretical foundation of our thesis.

Having established the approach of the thesis and the theoretical background of how it works, we will bring up a description of how we created the system. This will include how annotation was done, structuring of the datasets, and lastly how we constructed the different architectures.

Following the theory, we will discuss the results of the different model architectures. Furthermore, we will mention the differences between systems and the reason for the variations in the results.

Lastly, a discussion of what types of further research that could be carried out to improve upon our results will be brought up.

1.1.5 Limitations

A limitation while writing this thesis was to focus on implementations using Keras with Tensorflow as well as FLAIR. This was decided as both of us had prior experience with these frameworks and their simple yet powerful tools that allowed us to build, optimize, and test a large number of models efficiently.

Furthermore, a limitation was the amount of data available. The data that was extracted from Sinch was unstructured and raw, meaning that a lot of cleaning as well as manual annotation had to be done before being usable. These tasks were both very times consuming, resulting in a data set smaller than desired.

Throughout our experimentation, we only had access to one shared GPU. This made our data processing power limited resulting in training being time-consuming. Also during the time of writing this report, the pandemic COVID-19 struck resulting in a lot of time being spent away from the office. This affected the computation time available for us, as the GPU initially was not accessible from outside of the office.

Chapter 2
Theoretical background and related work

To fulfill the aim of this thesis, the task of data anonymization was divided into smaller sub-tasks. One of the tasks was to create a numeric representation of the language, making the computer able to understand and learn what was passed to it. Another aspect was to use this representation to make it possible for the model to distinguish between sensitive and non-sensitive parts. Furthermore, to benchmark the performance of the models, we used an evaluation script to output four different metrics for analysis.

The following theoretical section is structured for the reader to get the necessary background information to understand and follow the solution presented later in the report.

2.1 Previous Work

The starting point for this project was mainly the paper by Lange et al. (2019), where the authors described how they tackled the Medical Document Anonymization (MEDDOCAN 2019), a topic similar to ours.

The MEDDOCAN challenge was to develop a system for automatically identifying protected health information (PHI) from Spanish clinical records and masking these due to privacy concerns when being shared. Supplied with this task came a corpus of 1000 documents containing artificially constructed medical records. The task consisted of two sub-tasks, where the objective of the first task is to identify and classify named entities of the corpus. The other sub-task is described to only determine sensitive tokens of the text.

Lange et al. (2019) described how their team, *Neither Language Nor Domain Experts* (NLNDE), chose to tackle the anonymization task as a sequence-labeling problem. The team used a combination of various state-of-the-art techniques within the field of NLP to create a solution for the task. They developed a system including a BiLSTM in combination with a conditional random field output layer. Furthermore NLNDE chose to make use of contextualized string embeddings (a technique developed by Akbik et al. (2018)) for Spanish that would act as input to their network, in addition to using domain-specific fastText embeddings. Lastly the team utilized rule-based postprocessing to find and remove data not present in the training set. This postprocessing further improved the performance of their system, achieving impressive results.

NLNDE's article showcases promising results achieved with interesting architectures that apply to our topic. This sparked an interest in creating similar systems, constructed to fit our domain, based on their report.

2.2 Artificial Neural Networks

During the 1940s the foundation of modern machine learning was created. McCulloch and Pitts (1943) had an idea that was inspired by how the neurons of the human brain are constructed. Neural networks were born out of this idea.

When implementing these neural networks into software, they are called artificial neural networks as they no longer are physical objects as those of the brain. This technology, replicating the functionality of the brain, has not always been used for development in artificial intelligence. However, since the 1990s it has steadily gained traction, with it now being industry standard. This is mainly due to computational power being more accessible, giving way for larger amounts of data being processed by larger models according to Géron (2019).

During the late 1950s, Rosenblatt (1958) further developed upon McCulloch and Pitts (1943) idea of mimicking the neurons of the human brain. Rosenblatt purposed the classical perceptron, which simply put is a linear classifier. This idea was further developed and refined upon in 1969 when Minsky and Papert created the perceptron model that is still used today. Minsky and Papert suggested a more general computational model than that of McCulloch and Pitts (1943). This new solution allowed for the use of weights which made it possible to determine importance for various inputs, as well as learning from them. This made it possible to create more generalized and useful models.

Using these discoveries with its roots in the 1940s, a neuron can be explained to be a nonlinear, parameterized function of its input variables. Thus, a collection of neurons (or perceptrons) connected in a network is what we call an artificial neural network.

There are two different types of neural networks: feedforward and recurrent (also known as feedback) networks (Dreyfus, 2005). Recurrent neural networks will be discussed further later in this report.

2.3 Machine learning

Machine learning is a subset of the umbrella term artificial intelligence. Géron (2019) defines it as *the science of programming computers so that they can learn from data*. This learning process is done by feeding the system large amounts of data. The system is then able to discover connections and patterns within the data itself. When the learning/training phase has finished, the system should be able to carry out predictions on new data based on previous discoveries.

Ultimately, the goal is to allow computers to learn automatically and make adjustments without human intervention.

The term machine learning has a lot of different translations, however one a little more general than the one mentioned above what coined by Mitchell (1997):

A computer program is said to learn from experience ***E*** with respect to some task ***T*** and some performance measure ***P***, if its performance on ***T***, as measured by ***P***, improves with experience ***E***.

To put this into context Géron mentions the following example. Consider a spam filter program that is often used in email applications. This filter can learn how to identify and flag emails that are considered to be spam as well as non-spam. For the filter to learn and differentiate between the (in this case two different) types, a *training set* has to be created and used. This training set contains examples of emails of both categories (spam and non-spam) and will make it possible for the machine learning algorithm to get an understanding of what emails correspond to what category. Applying this to our previously mentioned definition, the task ***T*** is to correctly identify spam emails that arrive in our inbox, the experience ***E*** is the training data, and the performance measure ***P*** needs to be defined; which in this case could be the ratio of correctly classified emails. This particular performance measure is called accuracy and it is often used in classification tasks.

As big data is emerging, the range of fields where machine learning can be applied grows. These fields range from computational finance to energy production and natural language processing. It essentially depends on what type of problem it is and if it is solvable by using machine learning.

Machine learning typically gets categorized into different categories, namely supervised, unsupervised, semi-supervised, and reinforcement machine learning algorithms. However, in this thesis the focus will be on supervised learning and its applications.

2.3.1 Supervised Learning

Among the different categories, supervised learning is the most common type, and the vast majority of applications of deep learning fall under this category. Supervised learning algorithms can be explained as a learning algorithm that associates an input with one output. The output is often manually labeled. Initially, the learning algorithm analyzes a known training data set and the mapping to a corresponding label. As we run the algorithm, the model gets trained and model parameters get tuned to optimize the predictive capabilities of the model. Supervised learning is generally split into two subcategories, classification and regression problems. Both of these share the same goal of constructing a model able to predict output values based on the input values.

A regression problem is when the output is a continuous or real quantity, e.g. weight. Regression algorithms try to estimate a mapping function based on input variables to a continuous output variable, hence the model tries to predict the output quantity. The performance is then measured by the error in the prediction against the actual true quantity.

Classification problems are similar to regression problems, but with the difference in that the mapping from the input variables is to a discrete output variable. A classification problem typically consists of a model attempting to predict the category which an input variable belongs to. A typical example of such a problem is if an email is spam or not. This is a binary classification problem. Additionally, there are categorical classification problems which are when there are more than two categories. This is also the problem we face in this thesis (Chollet, 2017).

2.4 Natural language processing

Natural language processing (NLP), can (just as machine learning) be explained as a branch of the umbrella term “Artificial Intelligence”. NLP is simply put the interaction between humans and computers using our human *natural* language. The reason for NLP to exist is to make computers read and understand the language humans use in a valuable way. One way to interpret and understand the term was mentioned by Kapetanios et al. (2013). In his book *Natural Language Processing: Semantic Aspects*, he discusses how a natural language is built up from two major parts:

- **Syntax** is describing the grammar of a language, more specifically, how the words are arranged to make sense. In the domain of natural language processing, syntactic analysis is used to determine how the language is constructed in terms of grammatical rules. When feeding a computer with text for analysis, algorithms are used to apply specific grammar for different words. When this has been done, the systems extract the meaning from the text that it has been fed. One technique that is related to syntax would be part-of-speech tagging (POS).
- **Semantics** is describing what meaning a text has. This type of analysis is one of the harder aspects of NLP and has not reached its full potential yet. Semantic analysis is done by using algorithms to derive how sentences and words are constructed so that an interpretation and meaning can be determined. This is the technique that will be utilized through our thesis, more specifically named-entity-recognition (NER).

Going back in time, natural language processing started by utilizing only grammar rules together with other linguistic structures according to Kapetanios et al. (2013). This early approach was successful. Jumping forward a couple of years, when the computing power is cheaper and the amount of data is greater. The chosen approach for NLP practitioners has developed to a primary focus on machine learning.

With this data-driven focus, NLP is being used in a wide range of common applications areas such as language translation systems, word processors, as well as digital assistants without many people knowing that natural language processing is the foundation for it all. Needless to say there are many areas where NLP can be used. However, due to the ambiguous and imprecise nature of a language, it is difficult for a computer to understand.

Natural language processing utilizes algorithms to discover and retrieve information. This will make it possible for the system to transfer and understand the unstructured data of the language.

2.4.1 Named Entity Recognition

The term Named entity (NE) was first coined in the mid-1990s during the Sixth Message Understanding Conference. However, the first mention of a NER task was done when Rau (1991) purposed a solution for extracting company names from text.

The purpose of the NER subtask was to develop techniques for deriving named entities from sentences. The early focus was mainly set on information extraction tasks connected to business and defense activities (Grishman and Sundheim, 1996). The entities that were to be extracted came from a collection of unstructured texts – articles in newspapers. However, people quickly realized that the important parts to be extracted were names (of both persons and organizations), locations, and entities related to numbers (for example data and time). This formulation of named entity recognition subtasks is still used today.

As mentioned by Kapetanios et al. (2013), there are two main approaches currently used within the field. Firstly, hand-made rules that are linguistic and grammar-based methods of solving issues within the task. Secondly, more modern methods based on machine learning (and thus statistical models) When considering the two different approaches, there are a few main differences that are worth bringing up:

- Hand-made rule models allow for results with higher precision, however at the cost of a lower recall. These rule-based models demand a lot of experience and time to develop.
- Machine learning approach demands a lot of processed data, meaning that manual labor has to be put in. Additionally, a lot of computational power for the model to be trained properly.

Named entity recognition is the task of extracting parts of a text for categorization and identification of a predetermined collection of categories. These categories could potentially be anything but are, as mentioned earlier, often related to names, locations, and organizations. An example can be seen in Table 2.1

Table 2.1: Example of named entity recognition

x = [’eu’, ’rejects’, ’german’, ’call’, ’to’, ’boycott’, ’british’, ’lamb’, ’.’]
y = [’I-ORG’, ’O’, ’I-MISC’, ’O’, ’O’, ’O’, ’I-MISC’, ’O’, ’O’]

Here we can see that the sentence (which has been converted to all lowercase)

eu rejects german call to boycott british lamb.

has been labeled with labels following the IOB label scheme (Inside, Outside, Between). The class labels for this sentence are *ORG, MISC, O*. These have been extended to use I- and B- as prefixes to indicate if the entity is in direct connection to the previous chunk of the same entity. As an example the word *eu* is labeled *I-ORG* which indicates it is one lonely entity. Another example, following the same scheme, would be *Alice Bob and Carl* that would be labeled *I-PER B-PER O I-PER* (Ramshaw and Marcus, 1995).

Another annotation scheme commonly used is called IOB2 (Inside-Outside-Begin) that has a scheme with similar construction to IOB. The main difference is that it begins every entity with the B-prefix, followed by I-prefix if the entity consists of more than one word. An example would be the sentence *EU is an abbreviation for the European Union*, that would be tagged as *B-ORG O O O O O B-ORG I-ORG* (Krishnan and Ganapathy, 2005).

For named entity recognition to be effective, an annotation scheme must be in place with a fixed number of categories. When this scheme has been created, the annotation has to be done for the training, validation, and test datasets. This means that the entities that the system should recognize must be predefined and checked by the developer.

Furthermore, one important aspect to consider is the robustness of a named entity recognition system. A system that has been constructed for one specific domain will not be able to be utilized properly in other domain types. This is solved by fine-tuning the models for the domain it is used, which in itself is a time-consuming task.

2.5 Deep learning

Deep learning, also called deep neural networks, is a subfield of machine learning techniques that strive to teach computers what comes naturally for a human. Deep learning is not limited to supervised learning solely, meaning it can be applied to unsupervised as well when dealing with e.g. a clustering problem.

Deep learning is, as mentioned, a part of machine learning. However, deep learning uses these machine learning algorithms on more than one layer. Each of these layers interpret the data conveyed differently. As the data is processed through each layer, the transformations of the input become a more useful representation which is closer to the expected output for the given task. The stack of algorithms essentially forms a network of which is called an artificial neural network. The keyword *deep* is derived from the usage of several layers, creating the depth of the model.

2.5.1 Deep learning for text and sequences

In general, a text consists of a sequence of words or characters. As computers cannot understand the actual semantics of text as a human can, they merely recognize patterns and sequences. This can be done using tools such as deep learning. As deep learning models only work with numeric tensors, the input needs to be vectorized.

Vectorization. There are several ways to vectorize text and all of them break down the text into tokens. Tokens can be e.g. the words or the characters. For the vector space model a simple and common method for vectorization is where every possible word is mapped to an integer. Every word fits into a slot an array and the value of that index represents the number of occurrences of the specific word.

The simplest vector model is the bag-of-words vector. It is essentially a list of words and their word count, giving the frequency of each word in the set. The frequency obtained can be used in more elaborate vectorization methods, such as TF-IDF. In addition, TF-IDF takes into account the rarity of a word in the corpus, which indicates the importance of that specific word in the context.

However, there are some drawbacks to this method since it cannot capture phrases and multiword expressions. Other N-grams can be described as a contiguous sequence of n items extracted from a given sequence of text. The n-gram method gives the vectorization process insight into how frequently certain words occur together in the corpus (Chollet, 2017).

2.5.2 Backpropagation

Backpropagation is an algorithm used for training feedforward neural networks in supervised learning. The algorithm fine-tunes the weights of the neural network based on the error rate obtained from the most previously run epoch. Done properly, the tuning of the weights ensures a lower error rate and thus is a crucial part of training a model. However, it can be simply explained as a way to compute the gradient for the error function to the weights.

The backpropagation algorithms start with the final loss value and traverse backward from the last to the first layer. For a single perceptron, we simply minimize the differentiable error function (a function of the weights) by using the gradient descent method. When working with multilayered perceptrons, we have to exploit the chain rule (Goodfellow et al., 2016; Chollet, 2017).

The task presented in this thesis is a multi-class classification problem. This requires the use of a softmax output layer to determine the different class predictions. A categorical cross-entropy error function is then used to find the error distance between the correct and the predicted class during training. Succinctly, with calculations provided from Ohlsson and Edén (2019), we want our neural network to model:

$$\hat{y}_i(x_n) = P(C_i|x_n) \tag{2.1}$$

or in words the probability to anticipate a specific class for an output given a feature vector **x**. Furthermore the softmax activation function is:

$$y_i(x_n) = \frac{e^{a_{ni}}}{\sum_{i'} e^{a_{ni'}}} \tag{2.2}$$

This is essentially a generalization of the sigmoid/logistic function, which in other words is an activation function that determines the output of a layer. Here the ith unit, a_{ni}, is the input vector to output node i for a pattern n. Each value is then normalized by the sum of all the exponentials. The notion d in the following calculations is the one-hot encoding of the target classes, giving the true probability. A one-hot encoding is a sparse vector containing the same number of columns as classes present within the model. This vector is then used to describe what class best represents a word. The index of the corresponding class column is set to 1 and the rest is set to 0. More details about this will be discussed later in the report.

For multi-class output, the cross-entropy loss function is:

$$E(\omega) = -\sum_{n=1}^{N} \sum_{i=1}^{nbrClass} d_{ni} log(\hat{y}_i(x_n)) \tag{2.3}$$

Then computing the gradient yields:

$$\frac{\partial E_n}{\partial a_{ni}} = \sum_{i'} \frac{\partial y_{ni'}}{\partial a_{ni}} \frac{\partial E_n}{\partial y_{ni'}} = -\sum_{i'} \frac{\partial y_{ni'}}{\partial a_{ni}} \frac{d_{ni'}}{y_{ni'}} \tag{2.4}$$

As we can see in the last term of Eq. 2.4:

$$\frac{\partial E_n}{\partial y_i} = -\frac{d_i}{y_i} \tag{2.5}$$

Then calculating the gradient for the output y with respect to the input a, we get the expressions

$$\frac{\partial y_i}{\partial a_{ni}} = \begin{cases} \frac{e^{a_{ni}}}{\sum_{i'} e^{a_{ni'}}} - (\frac{e^{a_{ni}}}{\sum_{i'} e^{a_{ni'}}})^2 & = i = i', \\ \frac{e^{a_{ni}} e^{a_{ni'}}}{(\sum_{i'} e^{a_{ni'}})^2} & = i \neq i' \end{cases} = \begin{cases} y_i(1-y_i) & = i = i' \\ -y_i y_{i'} & = i \neq i' \end{cases} \tag{2.6}$$

Given from the design of one-hot encoded vectors, we also know:

$$\sum_i d_{ni} = 1 \tag{2.7}$$

This eventually renders:

$$\frac{\partial E}{\partial a_{ni}} = \sum_{i'}^{nbrClass} \frac{\partial E}{\partial y_{i'}} \frac{\partial y_{i'}}{\partial a_{ni}} \tag{2.8}$$

$$= \frac{\partial E}{\partial y_i} \frac{\partial y_i}{\partial a_{ni}} - \sum_{i' \neq i} \frac{\partial E}{\partial y_{i'}} \frac{\partial y_{i'}}{\partial a_{ni}} \tag{2.9}$$

Then using the division in Eq. 2.5, the expressions in Eq. 2.6 and the sum in Eq. 2.7, we can simplify the expression in Eq. 2.9 to

$$= -d_i(1 - y_i) + \sum d_{i'} y_i \tag{2.10}$$

$$= y_i - d_i \tag{2.11}$$

Subsequently, we can use this to retrieve an expression for the weights between the last hidden layer and output layer:

$$\frac{\partial E}{\partial \omega_{ij}} = -\frac{1}{N} \sum_n \delta_{ni} h_{nj} \tag{2.12}$$

$$\delta_{ni} = d_{ni} - y_i(x_n)) \tag{2.13}$$

From this we get the gradient of the error with respect to the weights in the last layer. To get the gradients for the layers prior to the last, we need to apply the chain rule once more, and so on. However this turns out to be problematic.

2.5.3 Vanishing Gradient Problem

The use of backpropagation and other gradient-based learning methods can cause certain problems. When using recurrent neural networks or deep networks, the feedback signal used for training the neural networks needs to be propagated through several layers, which can cause the signal to disappear completely. Concretely, the backpropagation algorithm computes gradients while applying the chain rule effectively causing a series of multiplications of many Jacobians.

Essentially, if either the weights or the gradients are small then it will cause the vanishing gradient problem. The overall consequence of this is a very slow training process, particularly for sequence data with long term dependencies (Goodfellow et al., 2016; Goldberg and Hirst, 2017). This will be discussed later in the report.

2.6 Recurrent Neural Networks

The classic and simple implementation of artificial neural networks that process inputs chronologically does not have any memory. Each input is processed independently and does not contain any states which are being updated between inputs. What such a network essentially does is to process the input in one go, which is why these are called feed-forward networks (Dreyfus, 2005).

However, there exist networks that are not exclusively feed-forward. In addition, there is a type of network called recurrent neural networks (RNN). These networks are constructed to process sequential elements while still maintaining an internal state that contains information about the input that was previously processed. The network obtains the information by utilizing a feedback connection, effectively creating an internal loop of itself. In short, RNN work by reusing information that was computed during previous iterations, see Figure 2.1.

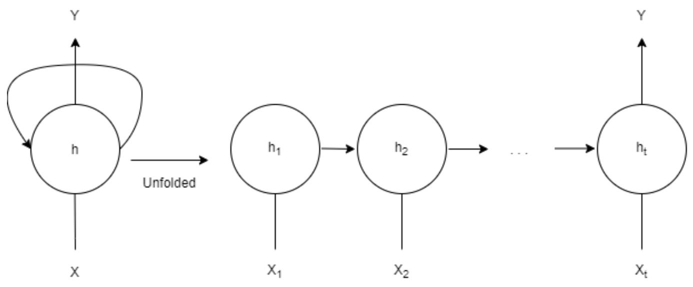


Figure 2.1: Visualization of Recurrent neural network unfolding

As an example, imagine reading a report. The reader can understand and interpret the content of the sentence better by having knowledge gained from the sentence before the one being read. This means that to understand the content of a sequence, one needs information from prior segments of the same sequence. This way of processing information applies to how we, as humans. interpret many different types of sequential data (such as text or audio).

2.6.1 Long Term Dependencies

One reason to use a recurrent neural network is to take the information learned from previous time steps into account in a later scenario.

This functionality is the foundation and core of the recurrent neural networks and is what differs them from feed-forward solutions. However, RNNs suffer from a major flaw in the ability to learn from previous time steps. The issue arises when the distance between previous information relevant to the present task is too large, due to the vanishing gradient.

An example of different distances would be if a system were to predict the next word in a text sequence based on information acquired earlier. Considering the sentence in Table 2.2.

Table 2.2: Example of short term dependencies.

The fish lives underwater.

The system will be able to determine that the last word should be *underwater* without any issue. In scenarios like these, the distance is deemed short and thus makes it possible for the network to use previously acquired relevant knowledge. However when the distance increases between relevant information and the system's current task, the system struggles. This can be seen by challenging our system to this time predict the last word of the longer sequence seen in Table 2.3. Worth mentioning is that the first entry is in an earlier time step of the sequence, the second entry is the entry to predict upon.

Information gathered from the proximity of the word to be predicted suggests that the word is a brand of car, however, there is still ambiguity about what car as this is related to

Table 2.3: Example of issue with long term dependencies.

All cars in Sweden are Volvos...
When I lived in Sweden I of course drove a Volvo.

information acquired many time steps earlier from the sentence *All cars in Sweden are Volvos.*, resulting in the system not being able to conclude what brand is going to be.

This issue was discussed in depth by Bengio et al. (1994).

2.6.2 Long Short-Term Memory – LSTM

One extension of recurrent neural networks is long short-term memory networks (LSTM networks). This solution was created in the late 1990s to tackle the long term dependency issue within simple RNNs (Hochreiter and Schmidhuber, 1997).

The foundation of today's LSTM was to introduce gating mechanisms for both input and output within the cells themselves. This construction would be extended further in 2000 by utilizing additional gates such as the forget gate. These refinements would improve the LSTM's ability to learn long-term dependencies (Gers et al., 2000).

In its essence, the LSTM network is a RNN, that instead of having a simple and shallow construction, has several different *gates*. These gates are introduced to make it possible for the cell to regulate how much information that is supposed to be carried down to the next state.

The unit itself consists of a hidden layer that has one memory block with at least one memory cell and as well as the gating units. These gating units are responsible for regulating the input and output of all of the cells within the memory block.

Furthermore, each cell contains, at its core, one recurrent self-connected linear unit which is named *constant error carousel*. These units are one of the features that makes LSTM networks powerful, as they solve the vanishing gradient issue.

Figure 2.2 shows a visual representation of a memory cell, where the arrows at the bottom, bottom-right, and top-right of the picture show the input flow of data to the memory block. This input is passed to all of the different gates (seen as the black circles), responsible for regulating the information presented to them. The gates are constructed by neural net layers based on non-linear sigmoid activation functions. The gating units can be either open or closed, regulated by either a zero or a one, which is evaluated through every time step. The variable G_{yin} is the input passed into the current state of the memory cell, denoted as S_c.

Essentially the addition that is seen in the figure: $S_c = S_c y^\rho + g y^{in}$ is what makes the difference of the LSTM. The computation enables a constant error that will be kept through backpropagation. This will make it possible to determine future cell states by adding the two data sources, (recurrent data and new input data) instead of through multiplication that would enable the problem of vanishing gradient. One thing worth remembering is that the forget gate itself is still dependent on multiplication to *forget*.

A step-by-step of the process of how the LSTM works:

1. Determine how much information is going to be kept from the previous cell state and how much is going to be discarded. This action is done by the *forget gate* which uses a

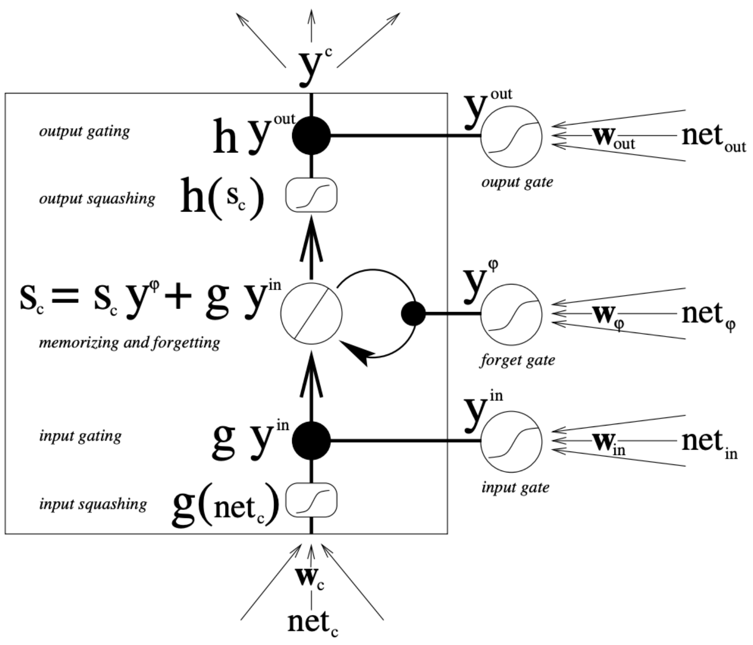


Figure 2.2: LSTM Memory cell, taken from Gers (2001)

logistic activation function (Gers et al., 2000) defined as:

$$y^{\varphi j}(t) = f_{\varphi j}(\sum_m w_{\varphi jm} y^m(t-1)), \tag{2.14}$$

where j indicated the memory block, and $w_{\varphi lm}$ is the weight on the unit connected from m to l.

2. Determine how much/which of the new information from the sequence that is going to be saved by determining which values to update. The *input gate* handles this, and is split into two separate steps which start with the input being computed using a logistic function to choose what values to be updated:

$$y^{in_j}(t) = f_{in_j}(\sum_m w_{in_j m} y^m(t-1)). \tag{2.15}$$

The second part runs the input through a **tanh** layer that will help choose new candidate values to be added to the state, the same principle that is used for simple RNNs.

3. Determine what the output is going to be. This output will be based on our state, however, it will be determined to only contain certain information. This action is carried out by the *output gate*, which also uses a logistic function to determine what values to be outputted.

$$y^{out_j}(t) = f_{out_j}(\sum_m w_{out_j m} y^m(t-1)). \tag{2.16}$$

The output will be based on what information is included in the current state of the cell, however, it will be a *cleaned* version of the state. The process starts by letting the current state pass through both the logistic function as well as a **tanh** layer (squishing the values to be between -1 and 1). The two results are then multiplied so that the logistic function can determine and produce the correct output.

This use of different gating units will allow the network to retain information in later time steps, decreasing the risk of vanishing gradient.

2.6.3 Gated Recurrent Unit – GRU

The gated recurrent unit network (GRU) is a fairly young network that was created by Cho et al. (2014). It can be seen as a simplified version of the LSTM network, designed to be an alternative that requires less computing power. The main difference is the number of gates the networks have, making GRU significantly faster. The GRU node only has two gates, namely a reset gate denoted r, and an update gate denoted z. These gates determine different values for the node. The reset gate determines how to combine previous values with new input, while the update gate determines how much of the current value will remain.

The update and reset gates are computed using a function that is dependent on the input symbol and the previous memory state. Initially these gates were defined to be binary. Meaning that whatever function used to compute the value for those gates would have a derivative of 0. This would be true for practically every case except the moment of transition between 0 and 1 occurs. As mentioned earlier in backpropagation and gradient computation, the gates need to be considered as real-valued coefficient vectors. Redefining the gates in this way allows for efficient computations, however, it also allows unwanted information to pass through (Cho, 2015).

The actual computations of the gates of the GRU nodes, as provided by Chung et al. (2014), are done with an activation h_t^j which at time t is a linear interpolation between the candidate activation $\tilde{h}_t^j$ and previous activation h_{t-1}^j. The equation is as follows:

$$h_t^j = (1 - z_t^j)h_{t-1}^j + z_t^j \tilde{h}_t^j \tag{2.17}$$

Then the update is computed with a sigmoid function and the parameter matrices U and W:

$$z_t^j = \sigma(W_z X_t + U_z h_{t-1})^j \tag{2.18}$$

The reset gate is computed in a similar fashion as the update gate:

$$r_t^j = \sigma(W_r X_t + U_r h_{t-1})^j \tag{2.19}$$

Finally, the candidate activation $\tilde{h}_t^j$ is computed in a similar way of traditional recurrent networks:

$$\tilde{h}_t^j = \tanh(WX_t + U(r_t \odot h_{t-1}))^j \tag{2.20}$$

2.6.4 Bidirectional Recurrent Neural Networks

Simple RNNs are by design dependent on the order of how they traverse the sequence that is being processed. This because they compute their input in the ascending order of time steps. If this order were to be changed, it would affect how the system interprets the sequence passed

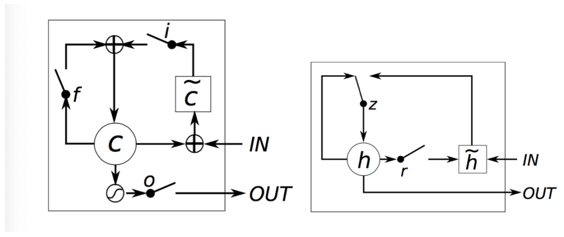


Figure 2.3: Image taken from Chung et al. (2014) *Empirical Evaluation of Gated Recurrent Neural Networks on Sequence Modeling*. (Left) LSTM node with three gates and extra memory c. (Right) GRU node with only two gates and not the separate memory cell.

to it. This construction of traversing the input in order is why the RNNs are well equipped to tackle challenges where the order is important.

During the mid-2000s, Graves and Schmidhuber (2005) developed a new technique for extracting the contextual dependencies between words in text sequences. This solution would offer a boost in performance compared to regular RNN's on specific tasks. Graves and Schmidhuber's idea was to exploit how the RNNs traversed sequences by introducing a secondary RNN layer. This additional layer would process the input data in reversed order compared to the first layer resulting in traversal from both left and right. When this had been done, the output from both networks would be concatenated into the state of the cell. This can be seen in Figure 2.4.

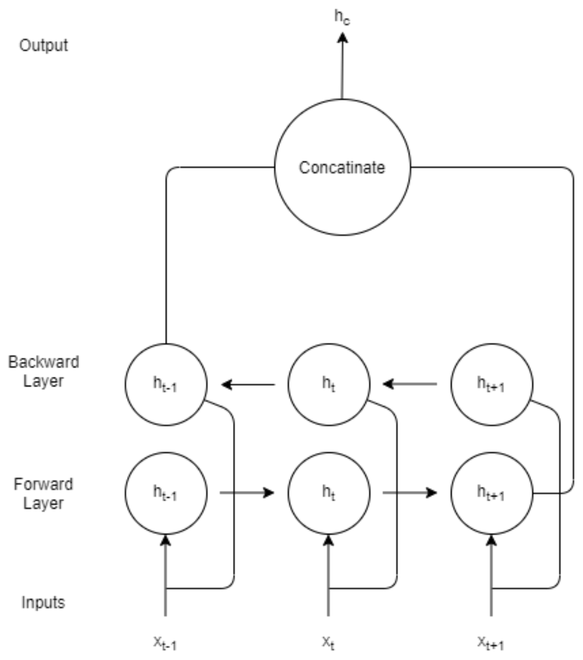


Figure 2.4: Example of bidirectionality.

An example of how the bidirectional traversal would operate on a text:

- *I went to the store.*
- *store the to went I.*

By allowing the system to process the data in the two opposite directions, it will discover different representations that would be missed by the traditional one-way chronological traversal. According to Chollet (2017), the word-order does have a big impact on machines to understand a language, however which order is not important. The importance lies in the different interpretations of data that are usable, are always worth exploiting.

2.6.5 Word Embeddings

As mentioned earlier, when dealing with natural language, the different units of language needs to be broken down into tokens. These tokens can represent characters, words, or n-grams. The simplest way of turning a token into a vector is by one-hot encoding. This essentially means that we are associating a specific index to each word.

Each word has a vector with a length equivalent to the number of tokens in the vocabulary. This vector is a sparse vector where every index has the value 0 except for one index, which is the index that represents that token. For example, we have a vocabulary of 30,000 tokens, each token will have a 30,000-dimensional vector. Further, if the token which corresponds to the word *tree* has the dimensional number of 15,347, then the vector for the word *tree* is set to all zeros except for index 15,347 which is set to 1 (Goldberg and Hirst, 2017).

Consider the sentence

Be careful with that butter knife.

First the sentence needs to be tokenized, hence we get:

$$\text{V} = [\text{Be, careful, with, that, butter, knife, ., ...}]$$

And using one-hot encoding on these tokens will result in

$$\begin{aligned}\text{Be} &= [1, 0, 0, 0, 0, 0, 0, \ldots]\\ \text{careful} &= [0, 1, 0, 0, 0, 0, 0, \ldots]\\ \text{with} &= [0, 0, 1, 0, 0, 0, 0, \ldots]\end{aligned}$$

However, one-hot encoding is not convenient in certain cases. It is often preferable to use dense encoding instead which are low-dimensional floating-point vectors, also known as word embeddings. These were first explored by Bengio and his team in the early 2000s.

These vectors are based on the data where the core features are embedded in a d-dimensional space. The dimension d of these vectors are normally smaller than the total number of features. To build these vectors, a set of core linguistic features needs to be extracted. Each of these needs to be relevant to predicting the output. Then each feature retrieves its corresponding vector. These vectors are combined by e.g. concatenation which results in an input vector that is to be used as input in the neural network (Bengio et al., 2003).

As an example of a word embedding its feature vectors, we will use the popular example of man, woman, king, and queen.

	gender	royalty	verb	human	...
man	0.9	-0.1	-0.8	0.8	...
woman	-0.8	-0.2	-0.6	0.9	...
king	0.8	0.9	-0.7	0.6	...
queen	-0.9	0.8	-0.4	0.7	...

Here we can see that words that appear in a similar context and have similar features do share large similarities between the vectors.

One of the main components of working with natural language processing and neural networks is the proper usage of embeddings. With enough data, it is possible to initialize an embedding layer with a vector containing random values. Then training the model to adjust these into usable vectors. It is also possible to use pre-trained embeddings, which essentially means that we load a pre-computed word embedding into our model. A pre-trained embedding is produced using machine-learning algorithms designed for a different task than ours. These are usually computed using word co-occurrence statistics, which essentially are the observations of how and what words are used in conjunction with each other. This also goes by the name statistical language modeling (Goldberg and Hirst, 2017).

Language Modeling

Language modeling is used as a measurement to estimate the relative probability in different sentences in a language. In other words, the probability of forming a sentence based on a sequence of words. This can then be used in calculating the probability of the next word in the sequence.

From the conditional probability we can calculate the probability of a word, given the preceding sequence of words. Using the chain rule of probability, the initial equation can be written as:

$$P(w_{1:n}) = P(w_1)P(w_2|w_1)P(w_3|w_{1:2})P(w_4|w_{1:3})...P(w_n|w_{1:n-1}) \tag{2.21}$$

However this equation is problematic due to the condition on the last term which makes it unmanageable. To solve this we can use the Markov property. The Markov property states that the calculated probability of a word occurring only depends on a limited history of the preceding sequence. For example, the kth order Markov property assumes the next word in the sequence only depends on the k previous words. However, there are some issues with this technique, such as the 0 probability assignment. This occurs when a sequence of words has not been observed previously in the training corpus. Also, when producing k-grams for large k, the process becomes be computationally expensive. However, these problems can be solved using neural networks (Bengio et al., 2003).

Language modeling and models like Bengio presented inspired different word embedding algorithms such as word2vec.

2.6.6 Word2vec and GloVe

The Word2vec algorithm was first developed by Mikolov et al. (2013) at Google. In short, the algorithm constructs word embeddings with the overall objective to retrieve word representation in the corpus. The algorithm then has to distinguish which of these are useful and which are not. The model is based on neural networks that utilize either two different types to produce distributed representations. Distributed representation here means that each token is associated with a d-dimensional vector. The actual semantics of the token is then seen in the different dimensions of this vector (Goldberg and Hirst, 2017). These types are the Skip-gram and CBOW.

Continuous Bag-of-words, CBOW, essentially tries to predict a word based on the context, ergo the surrounding words as shown in Figure 2.5. This model is a bag-of-word model because the order of the surrounding words does not influence the prediction. This is even though the models use both words from the past and the future. Skip-gram is the complete opposite of CBOW. Instead of predicting the word based on the surrounding words, the models predict the surrounding words based on a specific word.

Training the word2vec model produces two embedding matrices, one representing the words and one the contexts. The latter is simply discarded once the training is finished while the word embedding is kept.

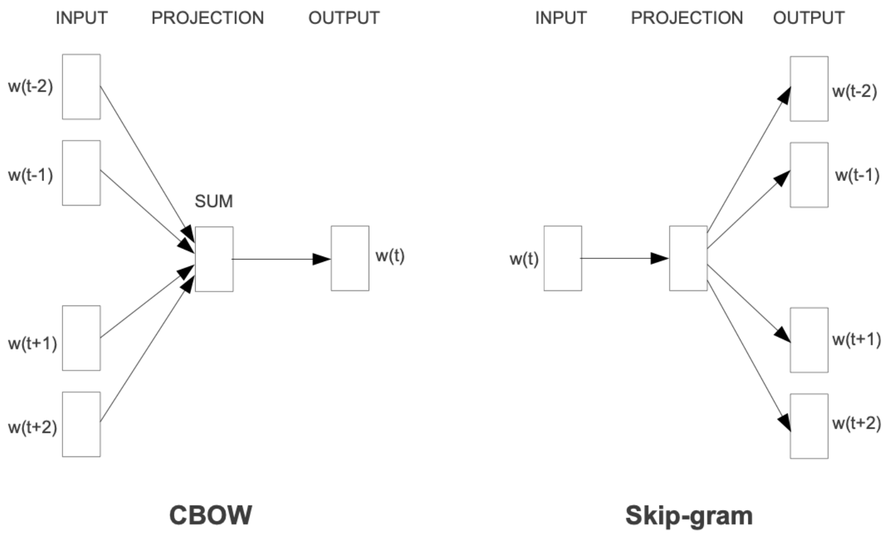


Figure 2.5: Image taken from Mikolov et al. (2013) *Efficient Estimation of Word Representations in Vector Space*. (Left) CBOW architecture which predicts the current word based on the surrounding words. (Right) Skip-gram architecture where the current word predicts the surrounding words.

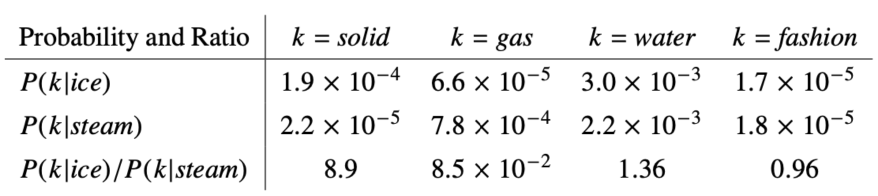
Another word embedding algorithm is the GloVe model introduced by Pennington et al. (2014) at Stanford University. The model is an unsupervised learning model designed to obtain distributed representations for words. The general technique is based on factorizing a matrix of word co-occurrence statistics, where the words are mapped into a vector space where the distance between words and semantic similarity is related.

A more formal description is provided by Pennington et al. (2014). Firstly, the matrix representing the word-word co-occurrence counts will be called X and the entries are labeled as X_{ij}. The label i here represents the context and the j the specific word. The actual entry is the number of times the word j occurs in context i. From this, the equivalence $X_i = \sum_k X_{ik}$ signifies the number of times a word appears in the context of the word i. Subsequently, the probability $P_{ij} = P(j|i) = \frac{X_{ij}}{X_i}$ represents the probability of word j appearing in the context of word i. Next, the co-occurrence probabilities are needed. These are defined as:

$$F(w_i, w_j, u_k) = \frac{P_{ik}}{P_{jk}}. \tag{2.22}$$

where w_i and w_j are word vectors which we want to check the co-relation to u_k, which is the probe word.

Figure 2.6 shows an example. Here we have two words, *ice* and *steam*. We want to see how these are related to the different probe words, *solid*, *gas*, *water*, and *fashion*. As seen, a large ratio implies that the probe word is related to w_i, while a low ratio means it is related to w_j. From the example we can see that *solid* and *ice* are more closely related than *solid* and *steam*. The opposite is true for the probe word *gas* which is closer to *steam*. Furthermore, both words are about equally related to the probe word *water* and unrelated to *fashion*



Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 2.6: Image taken from Pennington et al. (2014) *GloVe: Global Vectors for Word Representation.*

Next, as we need a model that encodes the information given by the division in equation 2.22 in the word vector space. Therefore restrictions are put on the function F to only depend on the difference of the target words is imposed. This reformulates the equation to:

$$F((w_i - w_j)^T u_k) = \frac{P_{ik}}{P_{jk}} \tag{2.23}$$

Next we make an assumption of homomorphism regarding F. This gives:

$$F((w_i - w_j)^T u_k)) = \frac{F(w_i^T u_k)}{F(w_j^T u_k)} = \frac{P_{ik}}{P_{jk}} \tag{2.24}$$

What this does is to ensure that the subtraction we did in the previous Equation 2.23 can instead be represented as a division. From this we can see that equation for the probability P_{ik} is:

$$F(w_i^T u_k) = P_{ik} = \frac{X_{ik}}{X_i} \tag{2.25}$$

Following this, another assumption is made. As we want to preserve the linear relationship between the embedding vectors, we assume that the function F is an exponential function, which in turn fulfills the property of homomorphism. Since $F(x) = exp(x)$ we get the equations:

$$Exp(w_i^T u_k) = P_{ik} = \frac{X_{ik}}{X_i} \tag{2.26}$$

$$w_i^T u_k = log(P_{ik}) = log(X_{ik}) - log(X_i). \tag{2.27}$$

Finally, introducing bias terms to restore the symmetry and knowing that the term X_i is independent of k, X_i can be absorbed into the constant bias term b_i. This yields the equation:

$$w_i^T u_k + bw_i + \tilde{b}u_k = log(X_i k) \tag{2.28}$$

Lastly, the cost function needs to be defined. Pennington et al. cast the equation as a least-squares problem along with the introduction of a weight function to alleviate issues arisen with equation 2.23, as it is ill-defined. This is due to if its arguments are zero and as well as all co-occurrences being valued equally. This is why we need to readjust the cost of each pair with the weight function f. This finally renders the model:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T u_j + bw_i + \tilde{b}u_k - log(X_{ij}))^2 \tag{2.29}$$

where:

$$f(X_{ij}) = \begin{cases} (x/x_{max})^{\alpha} & \text{if } x < x_{max}, \\ 1 & \text{otherwise} \end{cases} \tag{2.30}$$

Here Pennington et al. (2014) used $x_{max} = 100$ and $\alpha = \frac{3}{4}$ since it empirically proved to give the best performance.

2.6.7 FastText

FastText was developed by Facebook in 2015 and is an open-source library for making text representations usable in NLP tasks. Instead of creating a vector representation of a word as mentioned with word2vec, the computation of the embeddings is done for each character n-gram. This results in a word embedding that is a sum of all the n-grams present within a word. All the n-grams ranging between size 3 and 6 are used when training the representation.

An example of the process would be if we choose to extract tri-grams of words from a text. A vector created from the word *bicycle* would be the sum of representation based on the tri-grams: <bi, bic, icy, cyc, ycl, cle, le> (Bojanowski et al., 2017). Worth noting is that special boundary symbols, "<" and ">" are added to determine the start and end of a word. This construction allows for the generation of improved embeddings for words that are rare or have never been seen before in a corpus.

2.6.8 Conditional Random Field

As mentioned earlier, the most popular technique of tackling named entity recognition tasks is through supervised learning. The technique consists mainly of five different sub-categories; decision trees, maximum entropy models, support vector machines, hidden Markov models, and lastly conditional random fields (which in some sense is an extension to the Markov models).

Conditional random fields (CRF) can be explained further to be a discriminative probabilistic model (Lafferty et al., 2001). Simply put, this means that modeling of the probability distribution is done based on the outputs. Comparing this to a generative model that models based on both the input and output (Sutton and McCallum, 2011). Discriminative models are frequently used for tasks where predictions have to be made based on the surrounding context, such as named entity recognition.

In its essence, conditional random fields are very similar to Markov networks. Both of these supervised learning methods are using the Markov property (as mentioned previously) that can be described as:

If a sequence of states $\mathbf{Y} = y_1, y_2, y_3...y_n$, together with the outputs from said states as $\mathbf{X} = x_1, x_2, x_3...x_n$, is to be considered. The states themselves can only be affected by a previous state that is immediately connected to it. Meaning that y_2 could only be affected by the state of y_1.

A more mathematical approach to formulate it according to Sutton and McCallum (2011) would be to see it as:

$$P(y_t|y_{t-1}, y_{t-2}, ...) = P(y_t|y_{t-1}) \tag{2.31}$$

In the case of hidden Markov models, two assumptions are made to model the joint distribution mentioned previously. Namely the assumption that each state can only be affected by a previous immediate neighbor and that each observation x_t is directly dependent on the current state y_t. Together with these assumptions, the joint probability can be described as a combination of three distributions. Initial states- $p(y_1)$, transitions- $p(y_t|y_{t-1})$ and lastly the observation distribution $p(x_t|y_t)$.

$$p(y, x) = \prod_{t=1}^{T} p(y_t|p(y_{t-1})p(x_t|y_t)) \tag{2.32}$$

In Equation 2.32, a initial state y_0 set to 0 begins every state sequence. This will make it possible to express $p(y_0)$ as $p(y_1|y_0)$. The training of a hidden Markov model bases on the total number of times the transitions between sequences occur within the training data.

When using conditional random fields, the approach of determining the joint probability $P(x, y)$ is instead changed into computing $P(y|x)$ which is the conditional probability (Sutton and McCallum, 2011).

Thus, the distribution of the conditional random field model can be seen in Equation 2.33

$$p(y|x) = \frac{1}{Z(x)} \prod_{t=1}^{T} exp\left\{\sum_{k=1}^{K} \theta_k f_k(y_t, y_{t-1}, x_t)\right\} \tag{2.33}$$

Where X and Y can be seen as random vectors, θ as a parameter vector (containing weights) and $f_k(y_t, y_{t-1}, x_t)$ as a feature function. Furthermore, Equation 2.34 describes a normalization function $Z(x)$ that is input-dependent.

$$Z(x) = \sum_{y} \prod_{t=1}^{T} exp\left\{\sum_{k} = 1^K \theta_k f_k(y_t, y_t - 1, x_t)\right\} \tag{2.34}$$

Conditional random fields are in other words very similar to hidden Markov models. However the main reason for using them is that the maximization is done differently. In the case of CRF, the conditional distribution is subject to maximization whereas the approach for HMMs is to optimize the joint distribution. This difference in strategy makes CRF more fitted for tasks where sequence prediction is desired. However, this improved performance comes at a price of lesser flexibility, requiring annotated data and longer training time (Sutton and McCallum, 2011).

2.6.9 Contextualized String Embeddings

During the late 2010s Akbik et al. (2018) constructed a new way of extracting word embeddings based on contextual information. The idea behind the design was to pass whole sentences as split into sequences of characters into a pre-trained bidirectional character language model. The output of this CLM would then be passed into a sequence labeling model consisting of a biLSTM-CRF generating impressive results for named entity tasks. Figure 2.7 shows the architecture.

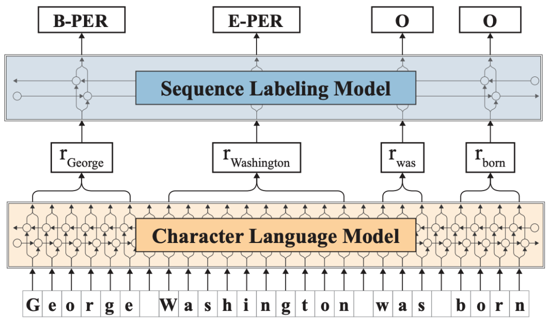


Figure 2.7: Use of contextualized string embeddings (image taken from (Akbik et al., 2018)).

Akbik et al. (2019b) purposed solution made it possible to create a combination of different popular embedding strategies. This construction gave the approach the ability to:

- Utilize pre-trained corpora, giving the model a lot of general knowledge.
- Extract the semantics of the words, based on their context. This allows for multiple embeddings for the same word to differentiate.
- Separate the word modeling into character level, improving the interpretation of for example misspelled words.

The language modeling architecture used for extracting the embeddings was an implementation of long short term memory networks that would maintain the long-term dependencies present in the sentence character sequences. The network was constructed to take in character sequences of sentences that would train the model to predict the next expected character. This results in the model having one hidden-state for each character presented to it. According to Akbik et al. (2018), the construction of a character language model is made in order to achieve the distribution $P(x_0 : T)$ for all the character within the passed sequence $x_0, x_1, x_2 ... x_T$. The model will then use the estimated predictive distribution of characters $P(x_t|x_0, ..., x_{t-1})$. This will allow the model to predict what the next character should be based on what the previous characters have been.

Furthermore, Akbik et al. (2018) explain that the joint distribution based on entire sentences can be seen as the product of the previously mentioned predictive distribution.

$$P(x_{0:T}) = \prod_{t=0}^{T} P(x_t|x_{0:t-1})) \tag{2.35}$$

By using the properties of the bidirectionality, as described earlier, the hidden states of the model are updated from both directions. The character computations themselves are handled separately and concatenated into a final word embedding when both directions have been processed. The forward language model is trained up until the last character of the word, and the backward language model until the first character. This will allow the embedding to understand and learn the word's meaning in a specific context.

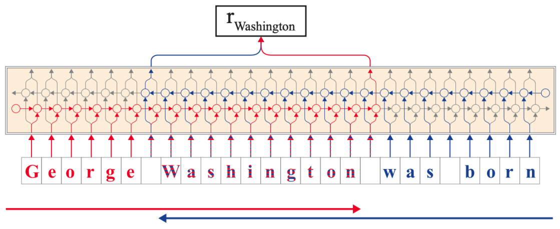


Figure 2.8: Embedding extraction from the word *washington* (image taken from (Akbik et al., 2018)).

An example of how the process of extracting the contextual embedding from the word *Washington*, can be seen in Figure 2.8. As mentioned earlier the language model is split into two parts, one forward model (red) and one backward model (blue). The forward model gathers information from the beginning of the sentence towards the last character of the word. The backward model's responsibility is the opposite of this, gathering information from the end of the sentence until the first letter of the word. These two sources are then concatenated, creating the final embedding that contains information that has been propagated from both the beginning and end of the sentence. This results in the creation of powerful embeddings that are trained to differentiate between the semantics of a variety of words in different sentences.

2.6.10 Pooled Contextualized Embeddings

The approach of contextualized string embeddings being completely character-based has one major flaw. The issue consists of creating usable embeddings from sequences that rarely occur within contexts that are not obvious. One example of this issue can be seen in the CoNLL-03 dataset, showed in Table 2.4.

Table 2.4: Example issue with contextualized string embeddings from Akbik et al. (2019b) using BIOES annotation scheme.

Fung	Permadi	(Taiwan)	v	Indra
B-PER	E-PER	S-LOC		S-ORG

The example in Table 2.4 showcases a text sequence that lacks an obvious context, in combination with the word "Indra" assumed to not have been seen before. This results in the model wrongly classifying the word Indra into an organization (ORG) as opposed to the correct person (PER).

The solution, presented in Akbik et al. (2019b), introduce dynamic aggregation of the contextualized embeddings extracted from unique text sequences passed to the model. The aggregation works by using a pooling function allowing for previously known contexts to help and create a refined global word embedding. Using this approach enables what Akbik et al. (2019b) calls *evolving word representations*. This means that every time a known word is processed, the embeddings changes based on the context.

A pooling function is used to group values from the input matrix of the function, outputting only the min/max/mean values. A large number signals the possibility of a pattern, while a small value indicates the opposite. The reason behind using a pooling layer is to filter out unnecessary data. For example in the case of a max-pooling layer, all the data that does not indicate patterns are removed, making it possible for the model to focus more on features discovered (Chollet, 2017).

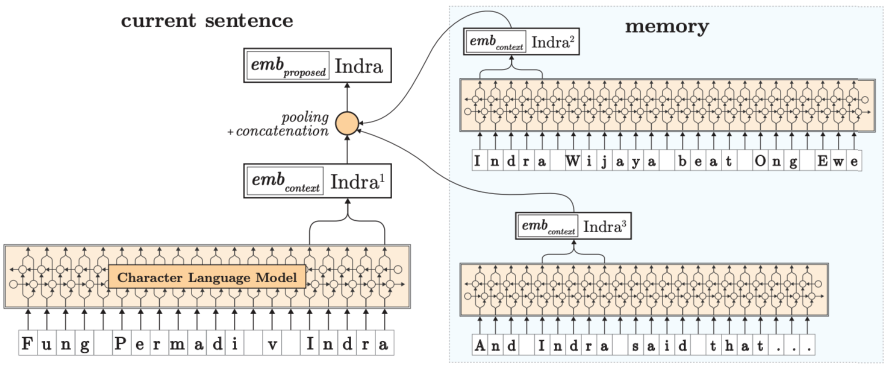


Figure 2.9: Process of creating pooled contextualized embeddings (image taken from (Akbik et al., 2019b))

The process of generating a pooled contextualized embedding can be seen in Figure 2.9. In this example the word *Indra*, mentioned previously from the sentence *Fung Permadi (Taiwan) v Indra* is processed following Akbik et al. (2019b) solution. The procedure starts by having the contextual embeddings for the word extracted. The complete collection of previously known embeddings for the specific word is then fetched from memory. These embeddings are then pooled and merged, creating the final refined embedding.

By refining embeddings through this process, the meaning of the same words will develop over time even though they are used within the same context as before. Furthermore, the embedding model will continue to learn even after the training process has finished, while the model is in prediction mode.

2.6.11 Comparing performance

To compare and properly evaluate the different models that were created, precision, recall, and F1 are computed for each of the final models. The score was determined by allowing for the created models to predict two different test sets. One test set that corresponded to the

CoNLL-2003 task, which allowed for comparison towards the state-of-the-art results available on the web. The other test set was based on Sinch's data and allowed for performance evaluation towards a real production scenario.

The performance computation was done by using the evaluation script that was supplied to the CoNLL-2003 shared task, constructed specifically to measure the performance of NER tagging.

The accuracy is a measurement of how many named entities the model was able to classify correctly when considering the total number of named entities that were predicted. This measurement was computed by dividing the number of correctly predicted entities by the total number of predictions (both true/false positives/negatives) as seen below:

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

The precision of the model is a measurement of how many correctly tagged entities concerning the total number of positive predictions made.

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

The recall is determined similarly. The computation that is made consists of dividing all of the true positives (correctly predicted entities) by all of the true positives and false negatives (falsely predicted entities). By dividing the positive predictions by the total number of true positives and false negatives combined.

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

For evaluation of the effectiveness of a model, normally accuracy together with precision and recall is calculated. However, for a named entity recognition task accuracy can be misleading due to the majority of the tagging will not belong to any predefined category. This will lead to an accuracy higher than what actually represents the performance of the model. Looking at the relationship between precision and recall, they are somewhat opposites. This means that if the recall increases in one scenario, the precision is reduced and vice versa.

Furthermore, the F_1 was computed by combination of both the precision together with recall mentioned by Tjong Kim Sang and De Meulder (2003) as can be seen below, where $\beta = 1$ since we want precision and recall to be equally important.

$$F_\beta = \frac{(\beta^2 + 1) * (Precision * Recall)}{(\beta^2 * Precision + Recall)}$$

This final measurement, the F_1, is telling us how effective our models are and is a good way to quickly compare different models against each other.

Chapter 3
Method

As mentioned previously, the task of our thesis was to develop a solution and evaluate this for the anonymization of sensitive data. To accomplish this task, we took inspiration from the work of Lange et al. (2019). This system was created as a submission to the MEDDOCAN competition where the task was de-identification of sensitive data present within medical documents. Lange et al. achieved impressive results that were the foundation for the construction of our system.

The following section will present how data anonymization was achieved in four parts. Firstly, the collection and type of data will be described. Secondly, the pre-processing of that data is presented. Thirdly, which models and a description of their architecture will be shown. Finally, it will be shown how the visualization tool for the results was created.

3.1 Datasets

A common methodology for benchmarking new model architectures is done by training the model on the same dataset as previously documented models have been trained on. This will allow for a direct and fair comparison of differing architectural decisions.

Applying the same mindset to our thesis leads to the use of two different sets of data. The first set that was used came from the CoNLL-2003 shared task described by Tjong Kim Sang and De Meulder (2003). The dataset contains 1393 news articles with a total of 301,418 tokens from the Reuters Corpus written in English. The annotation process for named entities was done manually at the University of Antwerp resulting in a corpus of excellent quality.

Annotation has been done by following the IOB-scheme mentioned earlier, containing the four different categories: Person (PER), Organization (ORG), Location (LOC), Miscellaneous (MISC) as well as Other (O). The original CoNLL-2003 dataset contains four different columns of information: the word, the part-of-speech tag, the chunk tag, and the named entity tag. As neither the part-of-speech tag or chuck tag was of importance to us, these columns were stripped, resulting in the structure seen in Table 3.1.

Table 3.1: Example structure of CoNLL-2003 dataset

EU	I-ORG
rejects	O
German	I-MISC
call	O
to	O
boycott	O
British	I-MISC
lamb	O
.	O

The second dataset was created by ourselves, containing 4500 different text messages resulting in a total of 93,447 tokens. These messages were cleaned and formatted to match the structure of the first dataset. However, the annotation scheme was extended to match the different types of information that were decided to be of importance by Sinch, seen in Table 3.2.

Table 3.2: Example of Sinch's annotation scheme

O	Other
A_S	Alphanumeric sequence
C_NBR	Card number
COM_N	Company name
CON_N	Country/Location
DATE	Date
E_M	Email Address
P_N	Person name
SCRT	Secret code
TEL	Telephone number
TIME	Time
URL	URL
USR_NM	Username

The process of annotation was done by hand through multiple iterations making sure that the corpus was of excellent quality. Having a well-labeled dataset is essential as our solution is based on supervised learning techniques. This means that having mislabeled data could result in bad performance from our model.

3.1.1 Annotation tool

To make it possible and create our own usable dataset, we carried out a manual annotation. We achieved it with the open-source software Doccano created by Nakayama et al. (2018) which allowed us all the annotation features that could be desired. The sequence labeling tool made it possible to create the structure that was wanted, as well as define our own set of tag categories. Furthermore, having the possibility of hosting the service on the network with different users made the annotation process a lot easier. Allowing us to annotate different parts of the dataset at the same time through a web browser made us save a lot of valuable time.

3.1.2 Preprocessing

When working with named entity recognition and any other classification task, data quality is very important as described earlier. To improve the quality, pre-processing of our corpus was carried out through several different steps. The process started by sanitizing the data, meaning for example the removal of unwanted text tokens and formatting anomalies. When these issues were fixed, the process continued with defining a numerical representation of the text existing within our data.

Data sanitation

To be able and achieve the best possible results for our models, sanitizing of our data had to be carried out. This was done through several iterations to guarantee a usable and high-quality dataset. The process started with separating the messages into exactly one message per row, simplifying the handling of the messages. When the separation was done, the focus shifted into filtering out all unwanted characters of each row (message). These characters were for example, those who could not be represented through our ASCII encoding and special signs in the beginning and end of a row. Furthermore, word contractions such as "you're" were extended into "you are" for consistency and clarification. Lastly, the whole dataset was converted into lowercase. One example to showcase how our preprocessing was conducted can be seen in Table 3.3.

Table 3.3: Example of how data was sanitised.

Before the messages were cleaned:
1. 43;This is a sample message.43;Don't move to close to the water.;

Result after the messages were cleaned:
1. this is a sample message.
2. do not move to close to the water.

The cleaning was done to remove irregularities of our datasets, further improving the number of known words within the vocabulary of our embeddings. Increasing this number allows our model to interpret and understand more messages, which could give better performance.

Creating textual representation

When the sanitation was concluded the preprocessing continued with the making of a numerical representation of the text usable to the model. The process started by separating the messages further, creating a list of separated tokens for each entry. This list was then accompanied by another list containing the corresponding NER tags, as seen in Table 3.4.

This allowed us to create a vocabulary containing all the unique words in the training set as well as the words within the pre-trained embeddings. Furthermore, a vocabulary for all

Table 3.4: Example of tokenized data together with NER tags.

x = [this, is, a, sample, message, containing, my, name, .]
y = [O, O, O, O, O, O, O, B-NAME, O]

the unique entity tags was also created. Even though our vocabulary covers a lot of words, there will always exist words that were not covered in the embeddings. In the case of our baseline models, that use only GloVe embeddings, an out of vocabulary (OoV) word vector with randomized values will be created. This word vector will then be trained based on its context to the correct position of the embeddings matrix. Additionally, the more complex models that made use of character-level embeddings do not suffer from the same issue of OoV word thanks to sub-word representations.

Having finished the vocabulary and association between indices and words an embedding matrix of shape (M, N) was created. The construction consisted of the unique words in the vocabulary (M) and the dimension of the embeddings (N). This matrix was then filled with the pre-trained embeddings through our previously created mapping between vocabulary and indices.

The input and labels were then converted by using the indices, encoding every word of the messages into a number interpretable by the model. To make sure our model was able to compute the input properly (with the messages being of different length), every sequence was padded. In our case, the maximum length was chosen to be 150, resulting in input having the shape of (number of sentences, 150). The last part of our preprocessing was to connect the messages passed into our model with labels. To achieve this, every label was one-hot encoded. This preprocessing prepared the dataset to be sent into the different models for development and training.

3.1.3 Creating a baseline model

A first step while creating a network for solving our task was to develop a baseline model. Having a baseline with simple architecture was important as it allowed us a foundation to compare and build from. Furthermore, developing a baseline ensured a complete understanding of our problem, sufficient data quality, and that additional development was needed. The baseline architecture together with the variations of GRU and LSTM was implemented using Keras. All these models utilize early stopping and were therefore trained for roughly 20 epochs. Keras is a high-level API written in Python with a Tensorflow backend developed by François Chollet. The models based on contextualized embeddings were implemented using the FLAIR framework created by Akbik et al. (2019a). FLAIR is a NLP framework based on Pytorch written in Python (Paszke et al., 2019). These models require significantly longer training. These also use early stopping and were trained for roughly 160 epochs.

Baseline: A simple RNN model

The architecture of our baseline was a simple recurrent neural network layer with output dimension 256, connected to a dense (prediction) layer. This dense layer outputs the predictions using a softmax activation function for the number of categories in a vector for each word. Additionally the network used GloVe word embeddings of 100 dimensions, pre-trained on Wikipedia articles. The construction can be seen in Figure 3.1.

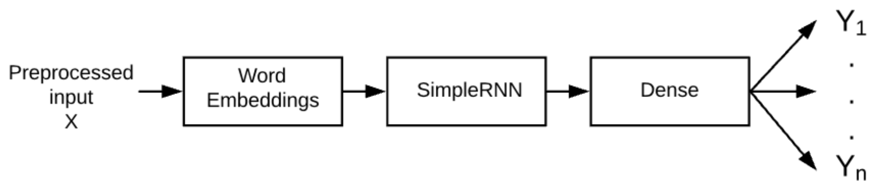


Figure 3.1: Simple RNN baseline architecture

3.1.4 Extending the baseline model

Having the baseline model allowed us to have a foundation to extend and make more accurate through several iterations. Working through an iterative process made it possible to determine what altering made the biggest impact on the performance of our models. However, through development a lot of different architectures were tested resulting in many different network types, with small differences. These networks built upon the baseline all have a similar construction, however varying results.

GRU model. The first model extended from the baseline architecture was designed to utilize a GRU network instead of the RNN. The network included a standard GRU layer with an output dimension of 256, a dropout of 0.5, and then connected to a dense (prediction) layer. This network, just as the baseline, used GloVe embeddings.

LSTM model. Moving further, a LSTM architecture was created allowing for comparison with the GRU network. This LSTM network had an output dimension of 256, a dropout of 0.5, and lastly was connected in the same manner to a prediction layer. The same GloVe embeddings were used in this construction.

Adding Bidirectionality and improving the baseline further. Having created the baseline model, as well as two additional extensions to it, we continued to extend all of the models with bidirectionality. This made the models discover more representations of the text, essentially extending the amount of data. To prevent overfitting and further improve the performance of the initial model, experimentation was carried out to create a more complex model.

This resulted in a model based on three layers as seen in Figure 3.2. Looking at the architecture, it consists of one simpleRNN layer in combination with two LSTM/GRU layers. These layers were constructed of the bidirectional, output dimension of 256, dropout, and a recurrent dropout of 0.2.

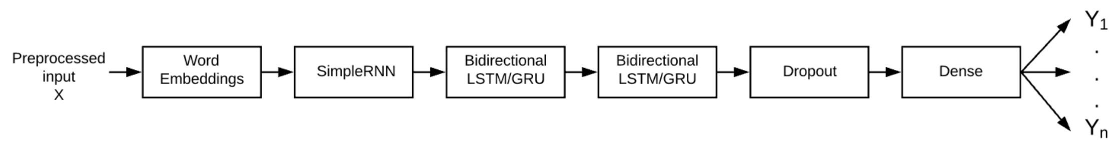


Figure 3.2: Architecture for bidirectional models

3.1.5 Contextualized String Embeddings, GloVe, BiLSTM, CRF

Taking inspiration from the architecture from Lange et al. (2019) and Akbik et al. (2018), we created a few different models. We constructed these models using the FLAIR open-source framework Akbik et al. (2019a), enabling us to create state-of-the-art results for our named entity recognition task.

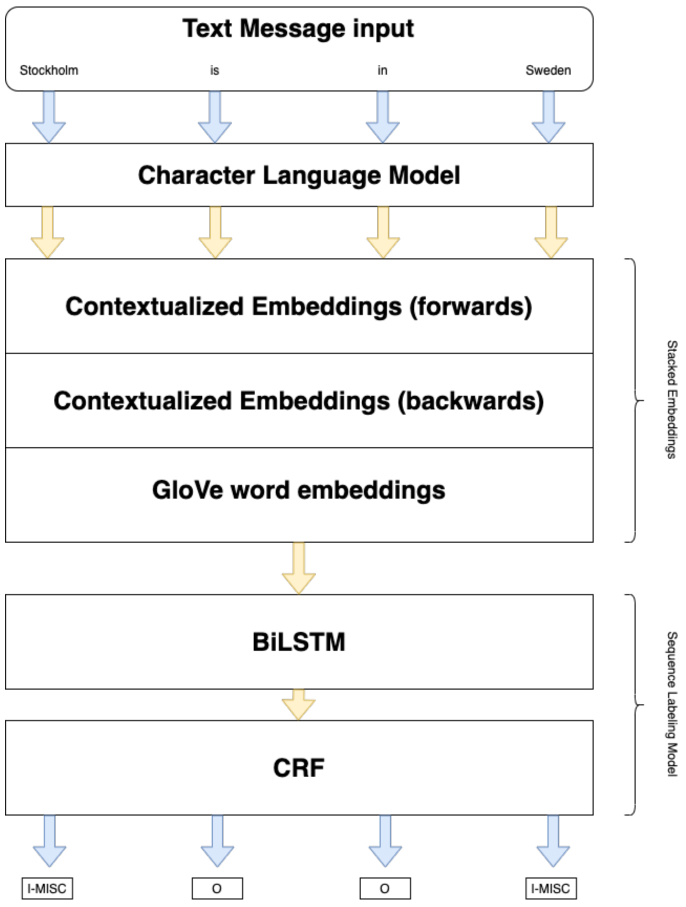


Figure 3.3: Architecture of the final system using Contextualized Embeddings, GloVe, BiLSTM and CRF.

Figure 3.3 shows our final architecture. It consisted of a character language model, followed by a stacked layer of character and word embeddings. A stacked-layer can be explained to be a concatenation of different embeddings, allowing for increased interpretability of the model.

Furthermore, these stacked embeddings were fed into our sequence labeling model that consists of a bidirectional LSTM-layer of size 256 together with a linear-chain CRF-layer. The network is trained using a learning rate of 0.1 and a batch size of 32. To determine how long our training should be carried out and avoid overfitting, early stopping was used based on the validation set. The early stopping was set to terminate training as the validation loss stagnated. The conditional random field was finally responsible for outputting the highest probability of the tag categories, based on what the neighboring tags were.

Comparing this architecture to the simpler baseline model, a lot of differences can be seen. Firstly the embeddings are created differently, where the simpler model utilized pre-computed GloVe embeddings. The FLAIR based model instead uses a character language model to create contextualized embeddings based on the input words and their context. These contextualized embeddings are then pooled and updated continuously throughout the lifetime of the model.

Furthermore, the simpler model only contains one RNN/GRU/LSTM layer that is connected to one last output layer, responsible for giving us the probabilities for each tag category. The more complex model on the other hand consists of a stacked embeddings layer, one bidirectional LSTM layer as well as a CRF layer outputting predictions based on the neighboring tags.

3.1.6 Visualization tool

To be able and showcase that the solution worked as intended a simple tool for visualization was developed. This software allowed the user to choose a model and compare the predictions made, with the correctly labeled ones. The idea was to see the performance without interpreting generated accuracy scores. An example of the tool can be seen in Figure 3.4.

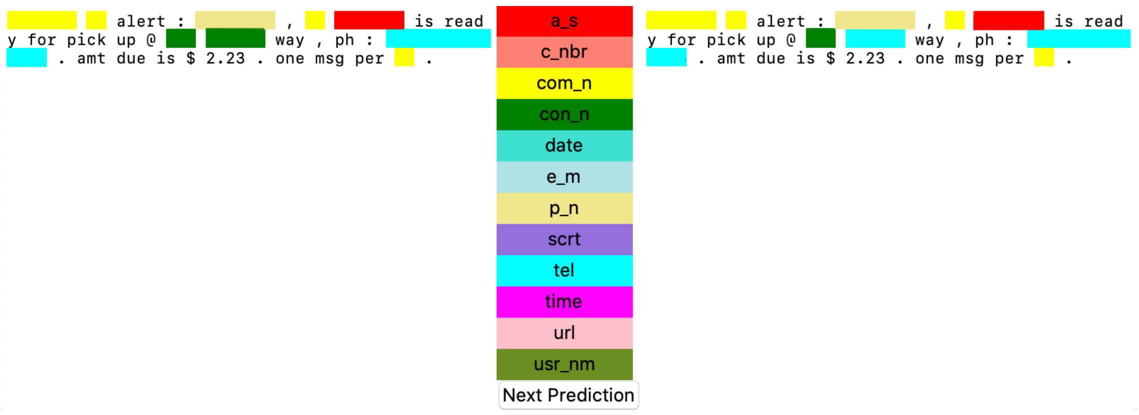

```
          alert :          ,        is read
y for pick up @             way , ph :
   . amt due is $ 2.23 . one msg per   .
```
```
          alert :          ,        is read
y for pick up @           way , ph :
   . amt due is $ 2.23 . one msg per   .
```

Figure 3.4: Visualization tool built to show the (left) correct labeling compared to (right) predicted labeling.

The visualization tool was developed using Python together with the standard GUI toolkit Tkinter.

In addition to the visualization software, a masking program was developed that utilized the model to output a masked version of the input. Table 3.5 shows an example of this.

Input that has not been masked:
input = [’eu’, ’rejects’, ’german’, ’call’, ’to’, ’boycott’, ’british’, ’lamb’, ’.’]
Output that has been masked:
output = [<ORG>, ’rejects’, <LOC>, ’call’, ’to’, ’boycott’, <LOC>, ’lamb’, ’.’]

Table 3.5: Example of masking tool.

This masking software was created to show how the model would be used in practice, removing the sensitive information of the input.

Chapter 4

Evaluation of our work

4.1 Results

Tables 4.1 and 4.2 show a complete comparison of the result for the respective datasets. We carried out the evaluation using the official evaluation script supplied with the CoNLL-2003 shared task. The script presents us with the percentage of tokens that were correctly tagged (in regards to precision, recall, and F_1). Additionally, only tag types that were present within the corpus data or in the pre-defined key sheet were included in the final overview. This evaluation script was used for controlling the performance of both datasets used in this report.

Sinch Dataset System	Precision	Recall	F_1
SimpleRNN GloVe	0.7286	0.7213	0.7250
GRU GloVe	0.7198	**0.7562**	0.7376
LSTM GloVe	**0.8381**	0.7436	**0.7849**
Bidirectional SimpleRNN GloVe	0.7783	0.7765	0.7774
Bidirectional SimpleRNN + GRU + GRU + GloVe	**0.9017**	0.8049	0.8505
Bidirectional SimpleRNN + LSTM + LSTM + GloVe	0.8760	**0.8582**	**0.8670**
BiLSTM+(GloVe+PooledFlair)	0.9290	0.9434	0.9361
BiLSTM+CRF+(GloVe+PooledFlair)	0.9395	0.9461	0.9428
BiLSTM+CRF+(Fasttext+Flair(Domain))	0.9540	0.9548	0.9544
BiLSTM+CRF+(GloVe+Flair(Domain))	**0.9552**	**0.9563**	**0.9558**

Table 4.1: Results using the Sinch dataset

Starting with our baseline model, SimpleRNN, we can see that it performed better on the Sinch dataset as compared to the CoNLL-2003 dataset, which also seems to be the recurring case overall. Furthermore, looking at the simple models, namely the first three in Table 4.1, the best model consists of an LSTM layer with a F_1 of 78.49%, ahead of the GRU with 73.76%. As excepted, the performance of simpleRNN is the worst with 72.50%.

CoNLL-2003 System	Precision	Recall	F_1
SimpleRNN GloVe	0.6235	0.7309	0.6729
GRU GloVe	**0.7352**	0.7502	**0.7426**
LSTM GloVe	0.7267	**0.7519**	0.7391
Bidirectional SimpleRNN GloVe	0.7216	0.7360	0.7302
Bidirectional SimpleRNN + GRU + GRU + GloVe	0.8113	0.8284	0.8198
Bidirectional SimpleRNN + LSTM + LSTM + GloVe	**0.8496**	**0.8322**	**0.8404**
BiLSTM+(GloVe+PooledFlair)	0.9034	0.9257	0.9095
BiLSTM+CRF+(GloVe+PooledFlair)	**0.9206**	**0.9276**	**0.9241**
BiLSTM+CRF+(FastText+Flair(Domain))	0.9054	0.9164	0.9109
BiLSTM+CRF+(GloVe+Flair(Domain))	0.9110	0.9201	0.9155

Table 4.2: Results using the CoNLL-2003 dataset.

As the capacity and complexity increases, the gap between the simpleRNN and the gate dependent layers grows. The bidirectional models give a substantial increase in performance where using bidirectional LSTM does give the best result with a F_1 score of 86.70%. Worth noting here, is the spread between the recall and the precision of the system using bidirectional GRU. This indicates that false negatives are a lot more common than false positives. However, there is a discrepancy compared to the result of the CoNLL-2003 dataset. Overall, comparing the performance of our models on the more general dataset, CoNLL-2003, the results follow a similar pattern.

Moving to the models that used biLSTM together with contextualized embeddings, the results improved markedly. The simplest model using these embeddings produced a result remarkably higher than the most complex model that did not. As we can see in Table 4.1, the best result produced using a model without contextualized embeddings has a F_1 score of 86.70%. This is to be compared to the worst result of a model using pooled contextualized embeddings with a F_1 of 93.61%, an increase close to 7 percentage units. After adding CRF to our BiLSTM system based on contextualized embeddings, the result improved even further, with the most notable improvement in precision with less false positives occurring.

The last two systems are both domain-specific, in an attempt to see the impact of training the character language model on the specific domain. As we can see the result improved once again. For both these systems the precision and recall are more similar, indicating that the false negatives and the false positives occur at a similar rate, relatively. These systems also produced a similar F_1 score of 95.44% and 95.58% respectively, with only a difference of 0.14 percentage units. This shows the marginal difference in using FastText embedding as compared to the GloVe embedding, with the latter performing slightly better.

Comparing this to the performance of the systems run on CoNLL-2003, we can see a similar pattern, showing the benefits of CRF. However, as shown having domain-specific embedding does not improve the result for CoNLL-2003.

Furthermore, the confusion matrix in Table 4.3 shows how the different words get classified and reveal which are the most common confusions. As noted, a fair amount of company names get predicted as O, producing false negatives. The other way around is also common, making false positives for company names the most common. Also, we can see that usernames and locations/country names are common misclassifications. It can be seen that our test set is underrepresented in regards to email addresses. However to those present, our model manages to identify them fairly well. It can be seen that a well defined diagonal is present within

	O	a_s	c_nbr	com_n	con_n	date	e_m	p_n	scrt	tel	time	url	usr_nm
O	14109	5	2	25	7	1	0	3	1	2	3	0	8
a_s	0	88	1	6	0	0	0	0	3	0	0	0	0
c_nbr	0	2	192	0	0	0	0	0	0	1	0	0	0
com_n	46	1	0	1355	9	0	0	2	0	0	0	1	5
con_n	10	1	0	6	307	0	0	1	0	1	0	0	1
date	0	0	0	0	0	191	0	0	0	0	0	1	0
e_m	0	0	0	0	0	0	7	0	0	0	0	0	0
p_n	2	0	0	0	2	0	0	313	0	0	0	0	0
scrt	0	1	1	0	0	0	0	3	226	2	0	0	0
tel	0	1	0	0	1	0	0	0	0	173	1	0	0
time	9	0	0	0	0	0	1	0	0	0	325	0	0
url	0	0	1	0	0	1	0	0	0	0	0	245	0
usr_nm	7	0	0	4	0	1	0	5	0	0	0	1	108

Table 4.3: Confusion matrix of the result from the best performing model using BiLSTM, CRF with GloVe, Flair(Domain) for the sinch dataset. This achieved an F_1 score of 0.9558.

our matrix, meaning that our system identifies the vast majority of the entities correctly.

4.2 Discussion

During this section, we will try to discuss and draw conclusions regarding the results obtained from our system. The developed solution presented in this report was constructed to identify sensitive entities present within messages. There will be a comparison between how the different architectures performed in addition to what the variations changed the models. We will start by discussing the output from the simpler systems and finally, look at the more complex systems. Lastly, we will mention a few extensions that could be done to the program, improving it further as well as a few closing thoughts.

4.2.1 Practical use case performance comparison

First, let us look at a fairly simple example as the sentence *c-123456 is your CompanyA verification code*. This sentence is an example of how a text message passed through the system would look like. We can identify that the words *c-123456* and *CompanyA* can be sensitive data, and thus needs to be masked.

Looking at Table 4.4, we can see that all systems manage to identify and mask the company name. However, the LSTM system is the only one that manages to successfully find both the company name as well as the secret code. One reason for this is that the LSTM model manages to retain the relevant information. This is done by using previously mentioned gating units for determining what information to keep and what to throw away. Through this the model can understand the relation between the word *verification code* and the actual code.

System	Original Sentence	Masked Sentence
SimpleRNN	c-123456 is your CompanyA verification code.	c-123456 is your <com_n>verification code.
GRU	c-123456 is your CompanyA verification code.	<con_n>is your <com_n>verification code.
LSTM	c-123456 is your CompanyA verification code.	<scrt>is your <com_n>verification code.

Table 4.4: Table displaying the masking process of the systems implemented in Keras.

Additionally the GRU based model manages to determine that *c-123456* is an entity, however not the correct one.

Comparing the models with bidirectionality with the original baseline models shows us that the previous issue is not persistent. Both of the simpler models (simpleRNN and GRU) are now able to detect the entity missed earlier.

As recurrent neural networks generally pass the data in a positive direction, meaning that the system is trained to predict based on information from the past. Then, as bidirectionality is added, the networks also pass the data in a negative direction, hence it is trained to perform predictions based on future information. Using these simultaneously will allow past and future input data to be used in calculating the same output. Therefore, this could be a result of the model finding the relation between *verification code* and the code as we process the sentence from both directions.

Moving further, comparing our extended baseline model, consisting of three bidirectional layers to the simpler single bidirectional LSTM model showed even more improvements. The idea behind this construction was to use the simpleRNN to extract broader features allowing for the two later LSTM layers to extract more detailed information of the input. This yielded even better results than through utilizing only bidirectionality, which can be seen in Table 4.5.

System	Original Sentence	Masked Sentence
SimpleRNN + BiGRU + BiGRU	CompanyA : please complete this brief survey to help improve ur service https://www.companya.com/ reply stop to end msgs	<com_n>: please complete this brief survey to help improve <com_n> service https://www.companya.com/ reply stop to end msgs
SimpleRNN + BiLSTM + BiLSTM	CompanyA : please complete this brief survey to help improve ur service https://www.companya.com/ reply stop to end msgs	<com_n>: please complete this brief survey to help improve ur service <url> reply stop to end msgs

Table 4.5: Table showing differences between using stacked bidirectional GRU compared to stacked bidirectional LSTM.

Here we can note that the architecture based on LSTM managed to classify both of the tokens correctly. In contrary, the model based on GRU is only able to determine the company entity in the sentence. This could be a result of GRU having a simpler construction, struggling with retaining long term dependencies. However, this is difficult to conclude due to the complexity of our system.

Changing the focus from the solutions developed without the use of contextualized embeddings and looking into the architectures suggested by Lange et al. (2019) we can note a clear improvement.

When comparing the extended baseline model with the overall best performing model, we can observe improvements in contextual understanding. Looking at the sentence pre-

System	Original Sentence	Masked Sentence
SimpleRNN + BiLSTM + BiLSTM	your feedback is important to us. please take a survey about your visit to Some Random Company https://randcomp.co/g3g224ty436y3	your feedback is important to us. please take a survey about your visit to <url>Random <com_n> <url>
BiLSTM+CRF+GloVe+Flair(Domain)	your feedback is important to us. please take a survey about your visit to Some Random Company https://randcomp.co/g3g224ty436y3	your feedback is important to us. please take a survey about your visit to <com_n><com_n><com_n> <url>

Table 4.6: Table showing differences between best system created in Keras and FLAIR.

sented in Table 4.6, the simpler model manages to identify the URL entity, however, it fails to fully recognize the company name to its entirety. Since the company name is split into three sub-words, resulting in the need for contextual awareness. The first part of the company names is incorrectly identified to be related to the URL, and not the other parts of the company name. Introducing our best model, that is constructed with contextual string embeddings, together with the conditional random fields sequence modeling algorithm for improved awareness of context. This architecture makes it possible for the model to understand the relation between the company name as well as the URL, masking all of the entities.

Furthermore, we can see that the models that utilized FastText embeddings did not outperform those constructed with GloVe. The use of GloVe embeddings seems to coincide with the results mentioned by Akbik et al. (2018), in his solution that produced state-of-the-art results. Worth noting is that both embeddings used were domain-independently trained on Wikipedia articles, which could explain the marginal difference.

Looking at the two models that produced the best results for the respective dataset we can see a clear difference. The best results for Sinch's dataset was produced by the model in Table 4.1 with a $F_1 = 95.58\%$. Comparing with the best results for the CoNLL-2003 dataset, created by the model seen in Table 4.2 resulting a $F_1 = 92.41\%$.

The variation of results between the architectures can be explained as differences in the datasets. Sinch's dataset consists of short texts with pattern based structure, resulting in the training data being a good foundation for the test data. This pattern-based structure allows our model to create a vocabulary corresponding to most of the words available in the test set. Having a good vocabulary together with a clear context makes the model less prone to the issues of character-level embeddings mentioned earlier.

Moving on to the CoNLL-2003 dataset, it is made up of news articles from Reuters corpus with varying lengths. The dataset contains a lot more variation between training- and test data and does not follow any general patterns. This results in a vocabulary that has not seen all of the words present in the test set. In addition to having a higher rate of unknown words, the contextual information of news articles is by nature not always obvious. These shortcomings causes the architecture to suffer from creating usable character level embeddings for rare words within unspecific contexts. This results in the use of pooled contextualized string embeddings in favor of regular contextualized embeddings.

Lastly, considering the results presented, it can be seen that the models implemented using LSTM as well as biLSTM performed significantly better than the models of simpler construction. Furthermore, the models that were created using contextualized embeddings together with CRF outperformed those that only made use of the simpler GloVe embeddings.

Looking in Table 4.1 it can be seen that the last three models have a comparable performance of roughly 95% F_1. This indicates that the architecture is in fact well suited for our dataset and task.

4.2.2 Further improvements

Although our models perform well on the dataset and the task overall, certain improvements of the system can be put in place.

Controlling the data, it is noticeable that the human factor is present, with mistakes regarding annotation and interpretation of messages. Furthermore, we can see that the dataset is rather small. Extending the collection of messages could improve the model's generalization capabilities.

Additionally, it contains an uneven distribution of entity categories (e.g email addresses), improving the diversity of these could help the model improve finding edge-case entities. Subsequently, the introduction of data augmentation could be useful for filling the number of samples in the dataset in a simple manner. For example, the current dataset could be translated from English into another language, then translated back to generate different sentences with the same meaning.

Moreover, the use of a domain-specific word embedding trained on similar text messages could help improve performance as our solutions only utilize domain-independent embeddings. This would, for example, make our model better suited for understanding abbreviations that are present within our dataset.

One more interesting addition to our system, mentioned by Lange et al. (2019) is post-processing in the form of language rules. These rules could potentially make it possible for filtering out most of the pattern-based text entities such as emails, URLs, and telephone numbers.

Lastly, to make the most out of our model a measurement for evaluating the uncertainty of our predictions would be a suitable solution. This evaluation tool would simplify human supervision of problematic data that requires additional attention.

Chapter 5
Conclusion

Throughout this thesis we have studied the possibility of creating a system for automatic removal of sensitive entities in text messages. We aimed to develop an anonymization system by using two frameworks, Tensorflow with Keras and FLAIR. In combination with these frameworks, we utilized different NLP techniques to achieve state-of-the-art results for identifying sensitive data.

This study concluded in the creation of two systems with different architectures. The first solution developed was constructed of three layers, one bidirectional simple recurrent network, connected to two bidirectional LSTM layers together with GloVe embeddings. This model produced a F_1 of 86.70% on Sinch's dataset, leaving headroom for improvements to reach state-of-the-art results. Inspired by Lange et al. (2019), we choose to create a system of similar fashion. The new system was developed to introduce more contextual awareness to the model using state-of-the-art NLP techniques. Creating a new model based on a bidirectional LSTM network in combination with contextualized input representations and CRF allowed for a F_1 of 95.58%. Achieving this, our initial objective of developing a system for automatic anonymization with state-of-the-art results was fulfilled. Even though our solution reaches impressive results, the model is not flawless, meaning that there still is a need for human supervision to reach perfection.

References

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., and Vollgraf, R. (2019a). FLAIR: An easy-to-use framework for state-of-the-art NLP. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, Minneapolis, Minnesota. Association for Computational Linguistics.

Akbik, A., Bergmann, T., and Vollgraf, R. (2019b). Pooled contextualized embeddings for named entity recognition. In *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, page 724–728.

Akbik, A., Blythe, D., and Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*, pages 1638–1649.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3(Feb):1137–1155.

Bengio, Y., Simard, P., and Frasconi, P. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE Transactions on Neural Networks*, 5(2):157–166.

Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146.

Cho, K. (2015). Natural language understanding with distributed representation.

Cho, K., van Merriënboer, B., Gulcehre, C., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014). Learning phrase representations using RNN encoder–decoder for statistical machine translation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1724–1734, Doha, Qatar. Association for Computational Linguistics.

Chollet, F. (2017). *Deep learning with Python.* Manning Publications.

Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling.

Dreyfus, G. (2005). *Neural networks. [Electronic Resource] methodology and applications.* Springer.

Gers, F. (2001). *Long Short-Term Memory in Recurrent Neural Networks.* PhD thesis, École polytechnique fédérale de Lausanne.

Gers, F. A., Schmidhuber, J., and Cummins, F. (2000). Learning to forget: Continual prediction with lstm. *Neural Computation*, 12(10):2451 – 2471.

Goldberg, Y. and Hirst, G. (2017). *Neural Network Methods in Natural Language Processing.* Morgan & Claypool Publishers.

Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning.* MIT Press. `http://www.deeplearningbook.org`.

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

Grishman, R. and Sundheim, B. (1996). Message understanding conference-6: A brief history. In *Proceedings of the 16th Conference on Computational Linguistics - Volume 1*, COLING '96, page 466–471, USA. Association for Computational Linguistics.

Géron, A. (2019). *Hands-on machine learning with Scikit-learn Keras & Tensorflow.* O'Reilly, 1005 Gravenstein Highway North, Sebastopol, CA 95472., 2nd edition.

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

Kapetanios, E., Tatar, D., and Sacarea, C. (2013). *Natural Language Processing: Semantic Aspects.* CRC Press.

Krishnan, V. and Ganapathy, V. (2005). Named entity recognition.

Lafferty, J., Mccallum, A., and Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, pages 282–289.

Lange, L., Adel, H., and Strötgen, J. (2019). Nlnde: The neither-language-nor-domain-experts' way of spanish medical document de-identification. In *IberLEF@SEPLN*.

McCulloch, W. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics*, 5(4):115.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space.

Minsky, M. L. and Papert, S. (1969). *Perceptrons : an introduction to computational geometry.* M.I.T. Press.

Mitchell, T. M. (1997). *Machine Learning.* McGraw-Hill.

Nakayama, H., Kubo, T., Kamura, J., Taniguchi, Y., and Liang, X. (2018). doccano: Text annotation tool for human. Software available from https://github.com/doccano/doccano.

Ohlsson, M. and Edén, P. (2019). Lecture notes on introduction to artificial neural networks and deep learning, fytn14/extq40/ntf005f.

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., Bai, J., and Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R., editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

Pennington, J., Socher, R., and Manning, C. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Ramshaw, L. and Marcus, M. (1995). Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.

Rau, L. F. (1991). Extracting company names from text. *[1991] Proceedings. The Seventh IEEE Conference on Artificial Intelligence Application*, i:29–32.

Rosemain, M. (2019). France fines Google $57 million for European privacy rule breach.

Rosenblatt, F. (1958). The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65(6):386 – 408.

Sutton, C. and McCallum, A. (2011). An introduction to conditional random fields. *Machine Learning*, 4(4):267–373.

Tjong Kim Sang, E. F. and De Meulder, F. (2003). Introduction to the CoNLL-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, page 142–147, USA. Association for Computational Linguistics.

Vollmer, N. (2019). Not applicable to anonymous data*. Recital 26, Intersoft Consulting.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2020-06-11

EXAMENSARBETE

STUDENTER John Helbrink, Simon Åkesson
HANDLEDARE Pierre Nugues (LTH), Jianhua Cao (Sinch), Michael Truong (Sinch)
EXAMINATOR Jacek Malec (LTH)

Dataanonymisering genom maskininlärning och språkteknologi

POPULÄRVETENSKAPLIG SAMMANFATTNING **John Helbrink, Simon Åkesson**

Att världen har blivit mer digitaliserad råder det inga tvivel om. Sett till omständigheterna vi just nu befinner oss i världen över behövs de digitala verktyg mer än någonsin. Men vilka konsekvenser för den enskilda användaren kan de digitala verktygen föra med sig?

I takt med att användandet av digitala verktyg ökar, ökar även mängden data som företag kommer över. Denna data kan sedan användas av företaget för att förbättra sina produkter och på så sätt skapa en bättre användarupplevelse. Men trots fördelarna med tillgången till mer data för företagen måste man ta personliga integriteten i beaktning. Hur får datan behandlas och vad får den innehålla?

I samband med dataskyddsförordningens utfärdande förtydligades detta. Företag runt om i Europa och övriga världen följer nu förordningen, vilket förser den europeiska befolkningen med ett skydd i hur företag sköter behandlingen av personuppgifter. Trots detta finns det ett behov av ytterligare skydd.

I vårt examensarbete undersökte vi möjligheten att anonymisera SMS. Just SMS mellan företag och kund innehåller ofta delar som kan kopplas till kundens personuppgifter. För att dölja kopplingen måste dessa ord maskeras. Väl när maskeringen är på plats blir SMS:et helt anonymt.

Först måste vi identifiera vad för problem det är vi jobbar med. Problemet som vi har framför oss kan beskrivas som ett Named Entity Recognition problem. Det innebär att vi vill kunna klassificera olika ord till fördefinerade klasser i en text automatiskt. Detta kan bland annat lösas med artificiella neuronnätverk.

Vad ett artificiellt neuronnätverk gör är att den kartlägger hur indata kan anknytas till utdata. I vårt fall är indatan ett ord och utdatan är om ordet ska maskeras eller inte. Ska ordet maskeras kommer det ersättas med en tagg. Denna tagg kan vara t.ex. "name", vilket då indikerar att det var ett namn som maskerades.

Simon och John skrev den här artikeln på LTH
<name> och <name> skrev den här artikeln <organization>

Figure 1: Exempel på hur maskeringen ser ut

Efter att ha skapat flera olika system kunde vi göra en jämförelse och se vilken av dessa som presterade bäst. De mer komplexa systemen visar att ordets sammanhang och att användandet av välanpassade representationer för orden är betydelsefulla. Med dessa teknikerna lyckades vi skapa ett system som med 95.58 % träffsäkerhet kunde maskera känsliga personuppgifter. Detta skulle innebära att stora mängder personuppgifter skulle slippa exponeras i onödan och varje människa skulle få ett ännu bättre skydd.