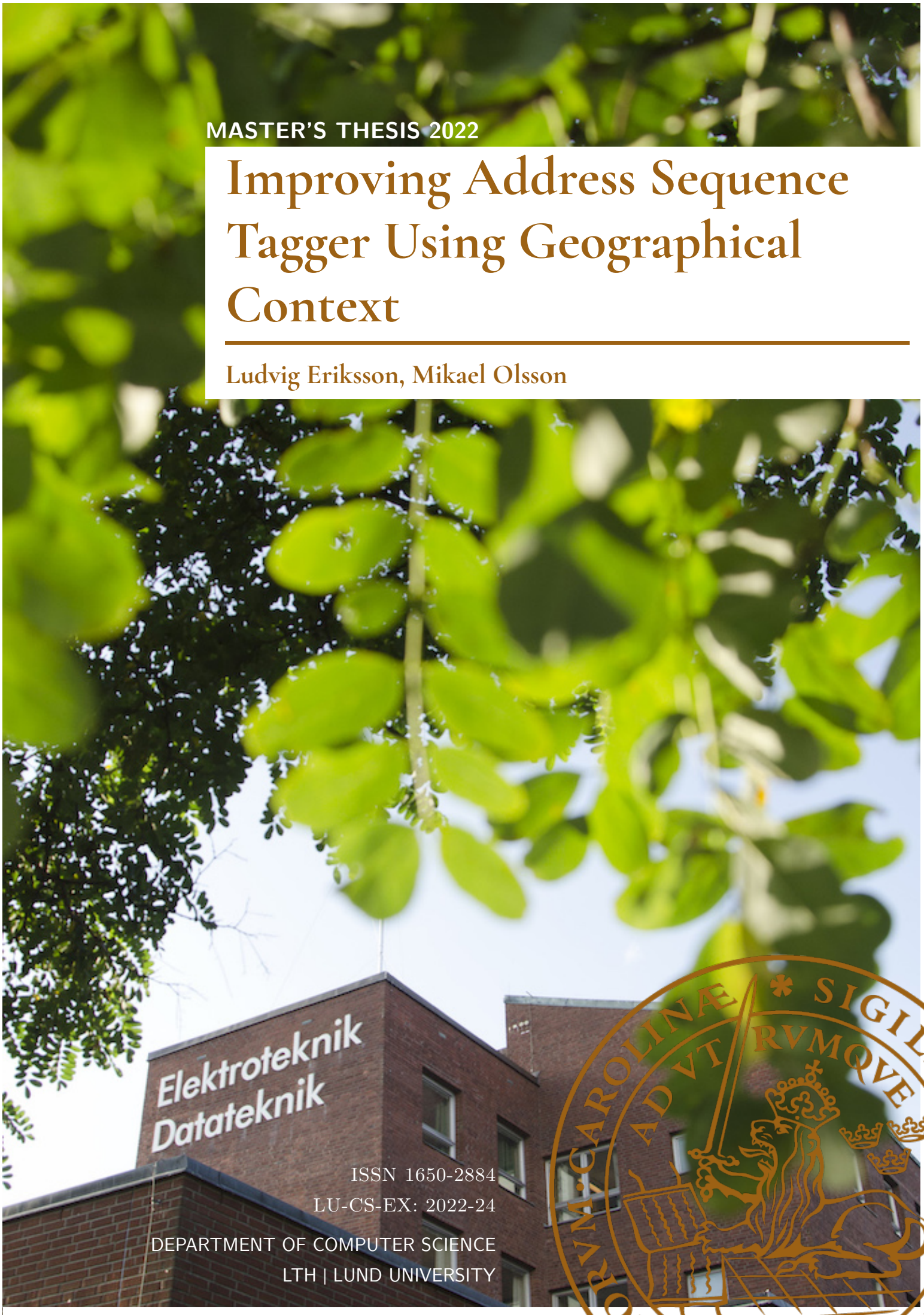
MASTER'S THESIS 2022

Improving Address Sequence Tagger Using Geographical Context

Ludvig Eriksson, Mikael Olsson



ISSN 1650-2884
LU-CS-EX: 2022-24

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2022-24

Improving Address Sequence Tagger Using Geographical Context

Förbättra en modell för sekvenstaggning av adresser med hjälp av geografisk kontext

Ludvig Eriksson, Mikael Olsson

Improving Address Sequence Tagger Using Geographical Context

Ludvig Eriksson
dat13ler@student.lu.se

Mikael Olsson
dic15mol@student.lu.se

May 11, 2022

Master's thesis work carried out at AFRY Digital Solutions.

Supervisors: Marcus Klang, marcus.klang@cs.lth.se
Frank Camara, frank.camara@afry.com

Examiner: Martin Höst, martin.host@cs.lth.se

Abstract

For humans, parsing an address into its components such as street and city is easy but time-consuming. Computers can have a harder time with such tasks. There are existing tools that use machine learning which has proven to be effective but still leaves room for improvement in some aspects. In this thesis, we investigate how deep learning models perform compared to an existing tool that uses a statistical model. On top of this, we explore how using geographical context can improve the accuracy of a machine learning model. After deciding on using Danish address data and having analyzed the data, we started by setting up baselines using existing tools and frameworks such as Libpostal, FLAIR, and Transformers by Hugging Face. Analyzing the data and baseline results led to setting up certain tests, such as trying different embeddings and oversampling underrepresented data fields. Using coordinates of an address, a nearby address was pulled from a spatial database and added as geographical context into a BERT transformer model to see how this would affect the accuracy. Our results showed that adding geographical context did improve the accuracy of a number of fields, though some fields likely suffered from the tokenization used for the transformer models. Compared to Libpostal, even the models without context performed better on most individual labels. Adding context to a BERT model did seem to offer an increase in accuracy, showing that geographical context is a viable method to improve parse accuracy.

Keywords: machine-learning, NLP, maps, address, parsing, geocoding

Acknowledgements

We would like to thank the following people for their help during our thesis:
Mikael Nelson for giving us an early introduction to machine learning.
Hampus Londögård and Marcus Klang for continuously answering all our questions regarding machine learning and thesis writing in general.
Dick Max-Hansen and Frank Camara for giving us the opportunity to write our thesis at AFRY and supporting us during our time at AFRY.
Daniel Palmqvist and Muhammad Ehsan ul Haque for our weekly meetings.

Contents

Chapter 1
Introduction

There are several different map tools such as Apple Maps, Google Maps and the open-source variant OpenStreetMap. For them to correctly interpret user input they need to correctly parse the address provided into its parts, for example, street name, street number, city and postcode. This in an important pre-processing step for *record linkage*.

Record linkage refers to the process of joining records that refer to the same entity in different data collections. This process is trivial when the records share a common, unique key or identifier (Li et al., 2014) although in the case of addresses there is often no unique key. Instead, we rely on element-wise comparison between pairs of records and this is why it is so important to split the address correctly into its elements so that you can match it to the correct record. This can be done with two different solutions: rule-based and probabilistic techniques. In this thesis, we will use machine learning based on probabilistic modelling that uses the codes of statistics for data examination.

Currently, there are tools such as Libpostal, which is based on a statistical model, and a rather large one at that. Libpostal claims a high accuracy, however, that is on a withheld data set which means that we are unable to confirm the results.

This masters' thesis project was done for a client of AFRY. The client handles an immense amount of address parsing and is constantly looking for new and improved methods. As in every other industry, the potential for machine learning grows and while the interest grows so does the knowledge and applications. One application that the client is interested in is the idea of using geographical context to get even better performance.

1.1 Problem Formulation

We aim to use geographical context to improve the predictions. Using the coordinates for an address should allow us to find the closest addresses and feed them into the machine learning

models to increase the accuracy of the predictions. The objective is to identify the labels available and then correctly label the input given. Using the input **Danskevej 23B, 11 4, 2300 Odense** should yield the result in Figure 1.1.

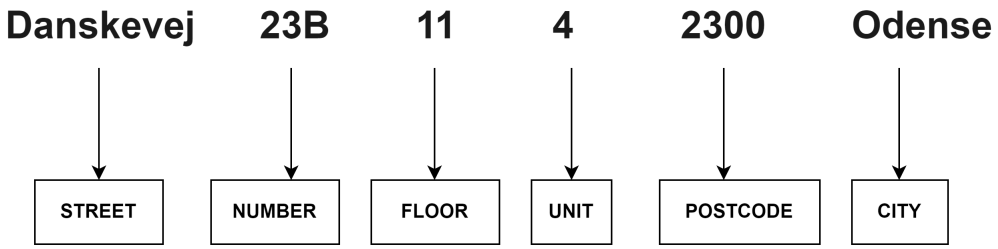


Figure 1.1: The expected result from parsing the address.

1.1.1 Scope

We decided to use Danish addresses for this project. The reason behind this is that they were readily available as the Danish authorities keep all addresses documented and available through an open API. We considered Swedish and French addresses but Swedish addresses were not available in larger quantities nor in good quality and French addresses were much more complex than the Danish ones.

An approach that was considered was to modify Libpostal, presented in Section 2.1, however, it was established early on to be out of scope for this thesis.

1.2 Research Questions

In this thesis, we aim to improve the accuracy in address parsing using geographical context in the form of nearby addresses. Using other tokens in an address as context while parsing addresses is nothing new and is used in Libpostal. Our goal is instead to use other addresses as context to improve accuracy, as nearby addresses are likely to share common traits which we hope will help improve predictions. In this thesis, we will aim to answer the following questions:

- **RQ1**: How well do different deep learning models perform on the address parsing task?
- **RQ2**: How do deep learning models compare with a statistical model such as Libpostal?
- **RQ3**: Can geographical context improve the accuracy of address parsing?

1.3 Contribution

Though context, in general, has already been widely used in natural language processing tasks, the use of geographical context remains relatively unexplored. With this thesis, we hope to give insight into the benefits of using geographical context in address parsing.

1.4 Work Distribution

Large parts of the thesis work was done in collaboration between the authors. Even parts that were divided, such as some of the programming, was often done with help and input from each other. Writing of the thesis was split between the authors, with Mikael writing large parts of the Technical Background and Ludvig writing much of the Results and Methodology. The Discussion and Conclusions were written together.

1.5 Outline

To begin, in Section 2, we will go through technical information and terms mentioned in the thesis and related work that we consider important to understand our thesis. We will introduce you to the basics of machine learning, natural language processing and we will give a short introduction to spatial databases.

Following this, we will introduce you to the data we have used in Section 3, where we got it and how we formatted it. We will also introduce our test, train and validation data sets. in Section 4.1 we will introduce how we approached this thesis, starting with planning and ending with the tests we performed. In Section 5 we will present the results of all our tests with an easy overview showing the main point of our thesis, after which we will go into detail for all the tests mentioned in Section 4.1. Next, in Section 6, we will discuss the different results and if they were as expected. Finally, in Section 7, we will present our conclusions and how we believe the work can be improved in the future through future work.

Chapter 2
Technical background

In this section, we introduce the theory that is necessary to know to understand our work. This includes previous related work, an introduction to machine learning, models used and spatial databases.

2.1 Related Work

Li et al. (2014) present a probabilistic address parsing system based on a Hidden Markov Model. They also introduce several other new approaches to create models yielding high accuracy. Just as in our thesis they define a set of semantic fields that they parse addresses into. They also highlight the importance of disarrangement, i.e. to reorder the address field. This is done to simulate how a person would actually input an address in a browser or message. Their result shows that *disarrangement*[1], in combination with *denormalization*[2] and *incompletion*[3], yields a higher result compared to unprocessed data. They used 100 million synthetic training instances and it yielded them a high F1-score of 95.6%. The paper shares many common variables with our thesis such as using a probabilistic system and investigating hyperparameters such as disarrangement.

Sharma et al. (2018) introduce a machine learning approach to address parsing using a neural network. Their approach to the issue at hand is interesting and differs from Li et al. by instead of introducing randomness into their data they instead correct the data and put it in the right order before feeding it into the model. They used a Stochastic Gradient Descent (SGD) model with backpropagation and trained on 1.1 million addresses achieving an accuracy of 97% on a test set of 10 000 addresses. The paper, just as our thesis, uses machine

[1] Rearranging the order of the words in the input.

[2] Replacing the input with an alternative form, such as replacing "California" with "CA" or "Cal.", and "NE" for "north east".

[3] Removing some of the words in the input.

learning to parse addresses but as it differs from Li et al by correcting data it also differs from our thesis.

Craig et al. (2019) investigate how active learning can be used to improve address parsing models. Active learning means that the learner is trained on strategically chosen examples so that it learns the most from them. It holds a big advantage in that it requires only a small set but the downside is that this set needs to be highly informative and labeled by a human most of the time. Craig et al. show that just adding a small amount of human-annotated real queries results in visible improvements compared to just using synthetic data. This was done on multiple models, including Libpostal, and showed that just adding 2000 human-annotated real queries improved the synthetic models by over 10%. While we do not use synthetic data in our thesis it is interesting to see how synthetic data could be used in conjunction with human-annotated data.

In Barrentine (2016) the address parser Libpostal is introduced, an open-source address parser written in C. It is a custom statistical Conditional Random Field (CRF) model trained on large, global data sets. It claims an accuracy of 99.45% on full parses, i.e it labels all the labels in an entry correct. Libpostal will be used as a baseline in our thesis.

2.2 Machine learning

The field of machine learning is generally considered to be a subset of artificial intelligence. The creation of the term machine learning is widely attributed to Arthur Samuel in 1959, from his work researching machine learning using the game of checkers (Samuel, 1959).

Machine learning is a method for a program to "learn", meaning to improve the performance of a task, by being fed data instead of being explicitly told how to do it (Goodfellow et al., 2016). In other words, the idea is that the program will perform better with experience. A more formal definition of machine learning was provided by Mitchell (1997):

"A computer program is said to learn from experience E with respect to some class of tasks T and performance measure P if its performance at tasks in T, as measured by P, improves with experience E."

In Figure 2.1 we can see a simplified representation of a machine learning algorithm. The input to the model needs to be a tensor, which is a numerical representation of the data. The goal of the model is to find the parameters θ that give the best estimation of the function that outputs Y. The output is also a tensor, which can represent a probability distribution for the output.

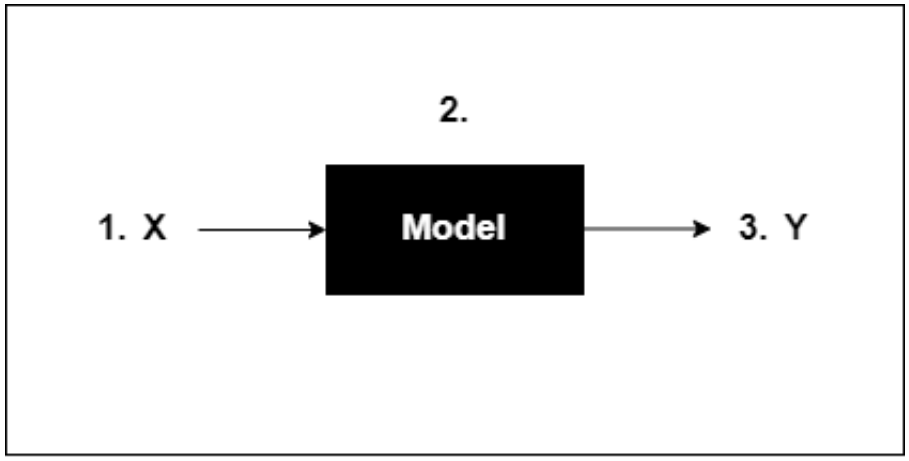


Figure 2.1:

1. Numerical input X (tensor) goes into the model along with starting values for parameters θ.
2. Model, $y = f(x; \theta)$
3. Numerical output Y (tensor) comes out of the model

Machine learning is comprised of a few different types of learning, some of the main ones being supervised learning and unsupervised learning. There are also some types of machine learning that are not considered among the main ones, but instead can be found somewhere in between, such as self-supervised learning.

2.2.1 Supervised learning

In supervised learning, the model will be trained on labeled data, meaning that an input will result in a specified output. Using the labeled data, the model tries to find the parameters for a function that minimizes the error between the output and the desired output.

2.2.2 Unsupervised learning

Unsupervised learning means that the data is unlabeled. Without labels, the model will try to find patterns in the input data and group similar data in clusters. Since there are no labels, it becomes slightly more complicated to evaluate the performance of the model compared with supervised learning (Cord & Cunningham, 2008).

2.2.3 Self-supervised learning

Somewhere in between supervised and unsupervised learning, there is a newer type of learning called self-supervised learning. Just like in supervised learning, self-supervised learning has the goal to learn a function that maps an input to an output. Unlike supervised learning, self-supervised learning does not use labels in the same way but instead tries to find some

structure or correlation in the data (which is why it is also similar to unsupervised learning). This is then used as a sort of label (LeCun et al., 2015).

2.3 Deep Learning

Just like machine learning can be thought of as a subset of AI, deep learning can be seen as a subset of machine learning. While deep learning took off in the last decade, its origins go back to the 1960s when work was done with neural network models (Schmidhuber, 2015). The idea with deep learning was to give a program the capability to work like the human brain and imitate brain patterns. (At least this was part of the original inspiration, though modern deep learning does not resemble the human brain in the same way) (Chollet, 2021). This is done using artificial neural networks.

2.3.1 Artificial neural networks

Haykins (Haykin, 2004) defines a neural network as:

"A neural network is a massively parallel distributed processor made up of simple processing units, which has a natural propensity for storing experiential knowledge and making it available for use."

At the core of deep learning, we have artificial neural networks (ANNs) that are modeled after biological neural networks that are present in the human brain. While at its inception it was inspired by the human brain, some researchers have pushed to move from this comparison, as it has developed since then and does not necessarily resemble the human biological neural network as much anymore.

A neural network consists of many so-called neurons, sometimes also referred to as units. The units are arranged in layers, where each layer typically does something different to the input. (The amount of layers used in the network is the depth, hence the name deep learning). The input units get activated from the environment, and then feed into the rest of the network, where other units get fed weights from previous units. Learning happens through finding weights that make the network perform the desired behaviour.

For the model to know if its modifications of the weights make it perform better or worse, some type of evaluation needs to happen. A loss function is a way for the model to find how far away from the desired outcome its prediction is, by calculating the difference. The difference is used to readjust the weights again, in whatever direction is needed to get a lower difference. An optimizer is responsible for this adjustment. This process is depicted in Figure 2.2.

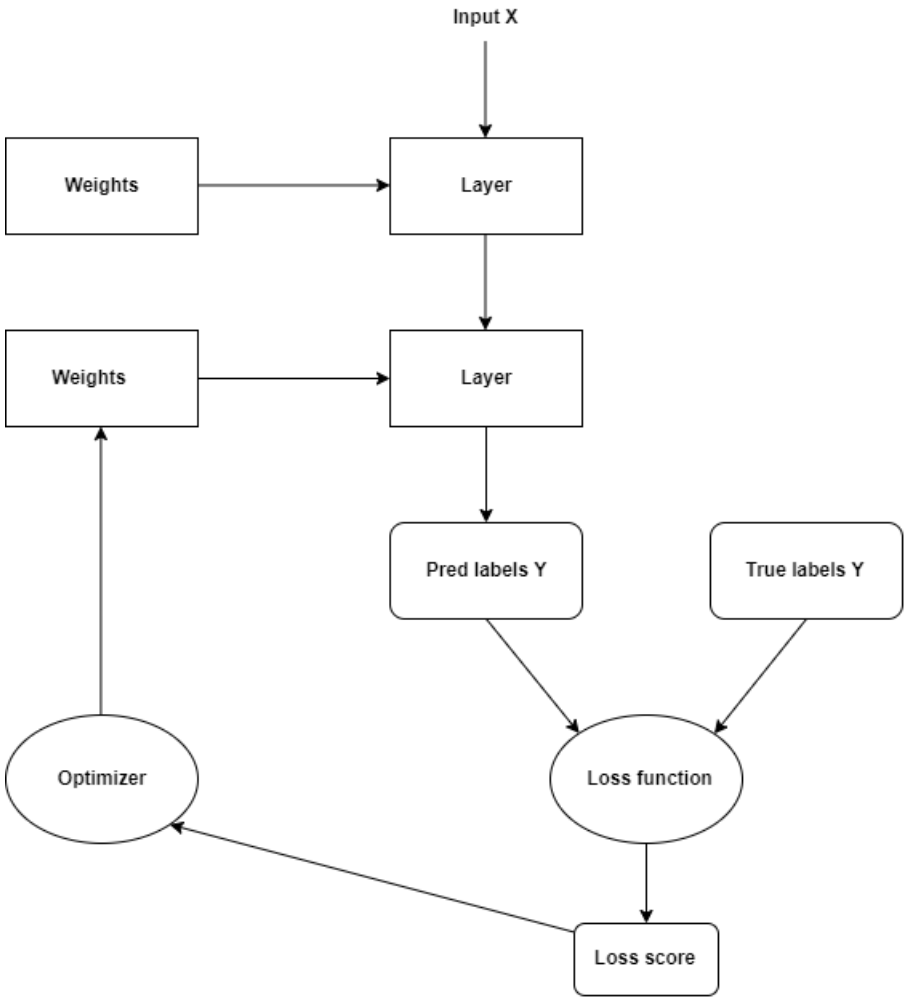


Figure 2.2: Learning process

Optimizers

A model learns by minimizing the loss function. The job of the optimizer is to update the parameters in the model to achieve a lower loss. The ideal is to find the global minima in the loss function curve. The most common way to optimize neural networks is to use **gradient descent**.

For each iteration in the training of a model, the loss is calculated. The gradient is calculated to move a step in the direction where the gradient is steepest, to move further down in the function, towards a minima, where the loss is smaller. The idea is to move more in the direction of smaller loss values until hopefully a minimum loss is found. Gradient descent is illustrated in Figure 2.3. There are a number of different techniques for doing gradient descent.

Batch gradient descent calculates the gradient over the entire training set at each iteration, which makes it quite slow for larger data. **Stochastic gradient descent (SGD)** picks a random selection of samples from the training data at each iteration and calculates the gradi-

ent using this, instead of the entire data set, making it significantly faster than batch gradient descent. This comes at a cost, as SGD can have a hard time converging to the minima. So we get a solution quickly, but it will not necessarily be optimal. Batch gradient descent will take longer to converge but can result in a more optimal solution. **Mini-batch gradient** descent calculates the gradient for a smaller subset of the data set, making it quicker than regular batch gradient descent.

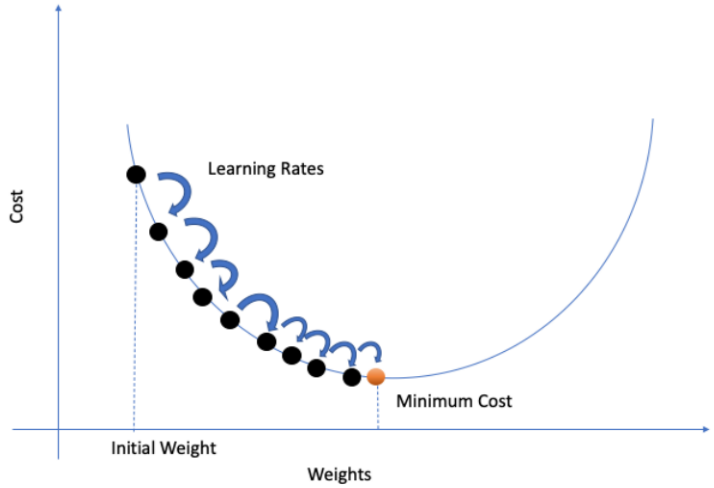


Figure 2.3: Gradient descent (Ghosh et al., 2020)

Momentum is a technique that can be used to avoid getting stuck in local minima, as it is the global minima that we want to find. It speeds up the time it takes to converge and the parameter update will depend on the current gradient as well as the previous parameter update. This results in a straighter path towards the minima, as seen in Figure 2.4

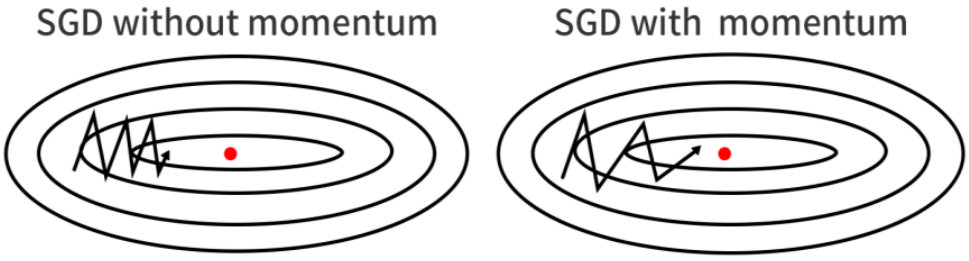


Figure 2.4: Stochastic gradient descent with and without momentum. Instead of oscillating across the slope, using momentum, a straighter path will be taken towards the minima (Du, 2019).

Adam (adaptive moment estimation) is an optimization algorithm, that uses adaptive learning rates to find individual learning rates for the model's parameters. Similarly to SGD with momentum, it also uses momentum. **AdamW** was developed to combat issues Adam had with generalization by decoupling the weight decay from the gradient update, meaning that changing the weight decay or changing the learning rate, does not affect the other.

Parameters

Before initializing the training of a model, several parameters have to be set. The learning rate decides how big the optimizing steps should be. If the learning rate is very high, the adjustment of weights might not be enough to properly influence the system. On the other hand, if the learning rate is too low, the system will learn much more slowly. An epoch decides how many times the training data should pass through the model. This is essentially the same as how many times the weights will be updated before the training is stopped unless it is stopped early. The batch size decides how many training samples are used in each iteration.

Overfitting and Underfitting

A common problem in learning is the issue of *overfitting* and *underfitting*. Overfitting is when the learning algorithm fits the training data too well by finding overly complex patterns that works well with the training data. The patterns will be too specific and won't work well with other data (Jabbar & Khan, 2014).

The opposite of overfitting is underfitting. This happens when the model is unable to fit the variability of the data and overgeneralizes. The result will be a model that is "too simple" for what it is trying to describe (Jabbar & Khan, 2014).

Early stopping

There are different techniques to avoid over- and under-fitting, a common one being *early stopping* (Jabbar & Khan, 2014). Early stopping means that the training of the model will be stopped once the error score of the validation data starts to increase, as seen in Figure 2.5.

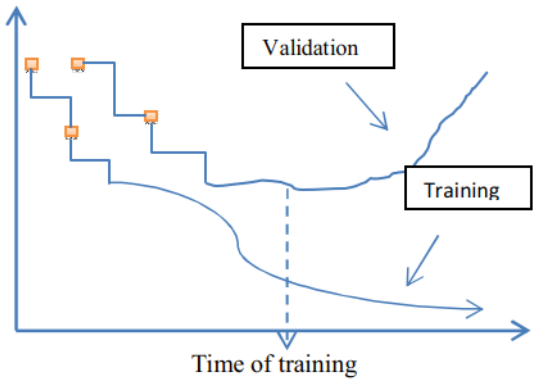


Figure 2.5: The dashed arrow shows at what time of training early stopping should occur, i.e. when the validation error starts to increase. (Jabbar & Khan, 2014)

Data

The data is split into three sets. These three are the training set, the validation set and the test set. A common split of these will be 60% of the data for the training set, and then the

remaining split evenly between the validation set and the test set.

The different data sets are used for different things. The training data is used to train the network. The validation data is used to tune the parameters of the models during the training, and the test set is used to evaluate the model after training has finished.

Architecture

There are two major groups in neural network architecture. The first one is a *feed-forward network*, which has no closed loops, and sends values forward in the network through different layers until it reaches the output layers. The second one is a *feed-back network*, also called a *recurrent neural network* (RNN).The RNN is often used as a sequence model since it can take into account word order and understand context (Chollet, 2021). One of the main objectives of RNNs is to be able to keep a memory of sorts. This is done using recurrent cells, as seen in Figure 2.6. A recurrent cell stores its previous value, though over time the information from previous iterations is lost.

To prevent this issue, long short-term memory (LSTM) was introduced in the 90s (Hochreiter & Schmidhuber, 1997). The LSTM acts as a type of memory for the RNN so that it retains information that might be useful in the future (LeCun et al., 2015). In Figure 2.6 we see that the LSTM has a memory cell. This cell has the same properties as RNN's recurrent cell, but also stores the current and last values of the cells state. The cell also contains three gates; input, output and forget. The gates control the flow of the information, how much of the input is let in, how much of the output is sent to the next cell and what should be forgotten (Leijnen & Veen, 2020).

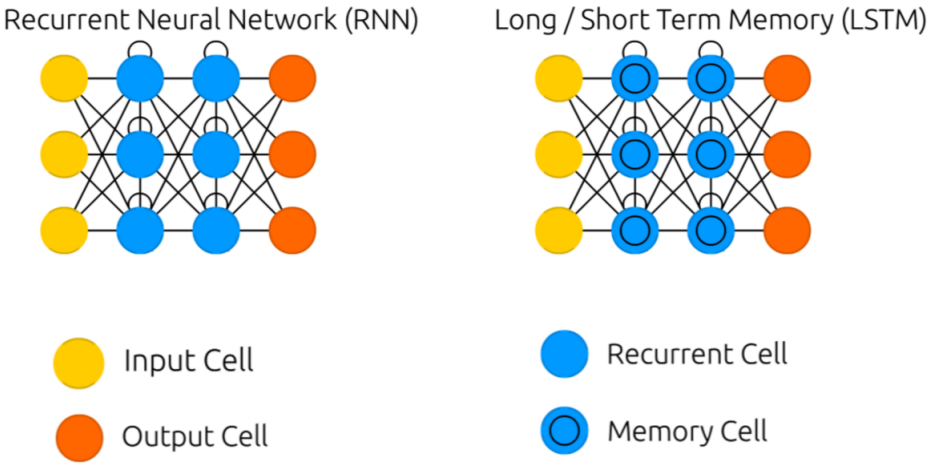


Figure 2.6: RNN and LSTM neural networks (Leijnen & Veen, 2020)

2.3.2 Transformer

The transformer architecture was introduced by a team at Google in 2017 (Vaswani et al., 2017) and has since then revolutionized the machine learning field. Up until now the use

of attention mechanisms have been, in all of but a few cases, only used together with a recurrent neural network. The transformer architecture on the other hand completely avoids recurrence and instead relies entirely on an attention mechanism that draws the global dependencies between the input and the output.

The transformer is divided into two main components, the encoder and the decoder. This is no different from other competitive neural sequence transduction models. The difference is that the encoder maps an input sequence of symbol representations to a sequence of continuous representations. It then generates an output sequence of symbols, one element at a time. The model is also auto-regressive which means that it consumes the previously generated symbols and uses them as additional input when generating the next. This is all done entirely using self-attention. Both the encoder and decoder uses fully connected and stacked self-attention layers.

One of the major benefits of the transformer is its multi-headed attention. Instead of performing a single attention function with its keys, values and queries it calculates it multiple times with different sets of keys, values and queries and then takes the average of the output. This allows it to run parallel computations which results in it requiring less time for training (compared to other models using attention).

Attention

Vaswani et al. (2017) describes the attention mechanism as mapping a query and a set of key-value pairs to an output where the query, keys, values, and output are all vectors. The output is then computed as a weighted sum of the values. What this means is that it weighs the relevance of the input elements and then takes them into consideration in the model's predictions. An example of this would be the sentence "The chicken crossed the road because it felt it wanted to get to the other side". A model would have some issues determining whether "it" referred to the chicken or the road. This is where the attention mechanism stands out by putting larger weights on the word "chicken" than the word "road", helping in the prediction process. The type of attention used in the transformer architecture is called Scaled Dot-Product Attention and is calculated using the function below:

$$Attention(Q, K, V) = softmax(\frac{QK^T}{\sqrt{(d_k)}})V$$

2.4 Natural Language Processing

In the domain of computer science, human languages are often referred to as natural languages. Natural languages are different from computer languages in several ways, the most distinct being that computer languages were created rule-first while human, natural languages were created usage-first (Chollet, 2021).

The field of natural language processing, commonly shortened to NLP, has grown at a rapid pace the last couple of decades together with the rise of the internet and access to information. A big usage area is information retrieval and information extraction (Nugues,

2014).

Increasingly, machine learning has been used in NLP tasks, starting with decision trees in the 1980s, going through statistical models, and around the mid 2010s, deep learning models started being used, especially RNNs such as LSTM. Since then, Transformers have started to replace RNNs (Chollet, 2021).

2.4.1 Named entity recognition

A type of task in NLP is named entity recognition. The idea is that given a sentence, we want to locate and identify entities within the sentence and classify them into predefined categories. For instance, if we have the sentence "Mikael is travelling to Rio de Janeiro", "Mikael" is labeled as "PERSON" and "Rio de Janeiro" is labeled "LOCATION".

2.4.2 Tagging

Named entity recognition uses something called tagging. Tagging also requires that sentences are *chunked*. This means that sentences are split into chunks of related words. If we use the address "10. Juli Vej 12, 6070 Christiansfeld, Danmark" as an example, we get the following chunks:

$$\{10.\ Juli\ Vej\},\ \{12\},\ \{6070\},\ \{Christiansfeld\},\ \{Denmark\}$$

The chunks are then tagged to extract some meaning. There are different tagging schemes, including BIO and BIOES. The letters mean the following:

- B - beginning of a chunk
- I - inside of a chunk
- O - outside of all chunks, i.e. irrelevant in the context
- E - ending of a chunk
- S - single part of a chunk

If we apply the BIOES scheme to the chunks above we get:

$$\{10.\ B-STREET,\ Juli\ I-STREET,\ Vej\ E-STREET\},\ \{12\ S-NUMBER\},\ \{6070\ S-POSTCODE\},\ \{Christiansfeld\ S-CITY\},\ \{Denmark\ S-COUNTRY\}$$

2.4.3 Tokenization

Before any meaningful work can be done with the text, it needs to be tokenized. A tokenizer essentially prepares the input for a model. Tokenization breaks down a text into smaller elements called tokens. There are different methods for tokenizing, common methods being splitting on whitespace or splitting on whitespace and punctuation. Using the string "**Mælkevejen 69D, 1440 København, Danmark**" as an example to demonstrate tokenizing

methods. If we were to split only on whitespace we would get 5 tokens but if we were to use whitespace and punctuation to tokenize the same sentence, we end up with 7 tokens. The difference can be seen in Figure 2.7. These two methods are fairly straightforward. There are other ways to tokenize text that are slightly more complicated, but more beneficial.

WordPiece tokenization is a technique initially developed to deal with Japanese and Korean voice recognition issues at Google (Schuster & Nakajima, 2012), which is now used in BERT models. The first step is simple whitespace and punctuation tokenization, after which WordPiece does subword tokenization. Subword tokenization can be thought of as a combination of word- and character-level tokenization, where rare words can be split into smaller subwords. This way, the root of a word can be kept which can help understanding other words using the same root. It is similar to Byte-Pair Encoding which is explained in Section 2.4.4. An example of WordPiece tokenization can be seen in Figure 2.7.

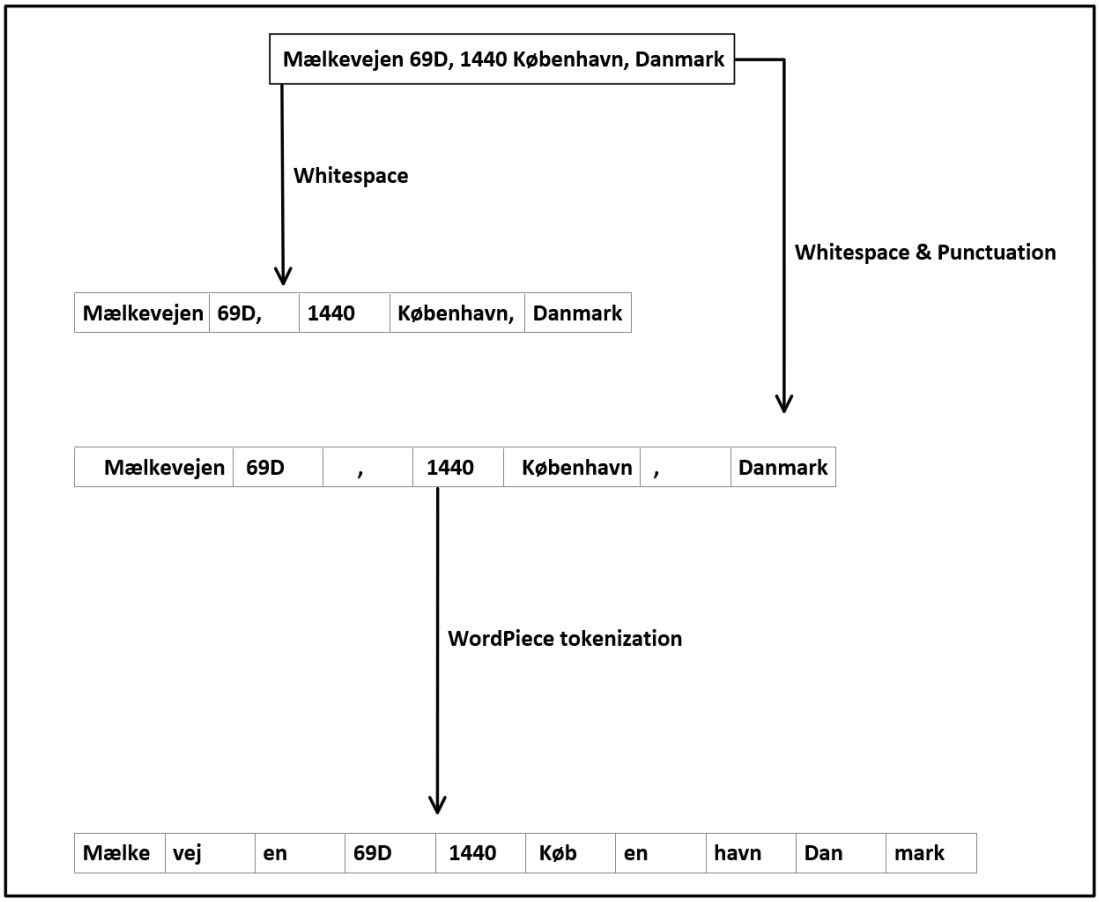


Figure 2.7: Different tokenization methods.

2.4.4 Embeddings

Splitting your input into separate tokens is just one step towards feeding the input into the model but since the computer only understands numbers the generated tokens need to be translated into numeric vectors that in turn can be fed into the model.

One Hot Encoding

One hot encoding is the most basic way of encoding the tokens. An integer, i, is assigned to each token. The token is then represented by a vector of size N where N equals the size of the vocabulary with a 1 on the i:th position and zeroes on the rest. This makes them sparse, hard-coded and high-dimensional which takes up a lot of space.

Word Embedding

To reduce the high-dimensionality of one hot encoding word embeddings were made. Word embedding is a classic embedding type that is static and distinct word gets one pre-computed embedding (vector). Most embeddings are of this type and include popular types such as GloVe, Word2Vec and FastText. The attempt to encode the meaning of the words, to capture the semantic meaning of the word which in turn means that words with similar semantic meanings will have similar vector representations. The vectors often have tens or hundreds of dimensions which can be compared to the thousands or millions of dimensions that are required for sparse word representations such as one-hot encoding.

Word embedding uses two different ways to learn the embedding representation: CBOW and skip-gram. CBOW stands for *continuous bag-of-words* and uses a certain window size to predict a word using its surrounding words. Skip-gram on the other hand works the entire other way and tries to guess the context given a word.

A fascinating feature of word embeddings is that the semantic relationship between different words can be represented as a geometric relationship between the numeric vectors. This means that you can do simple vector operations, such as addition and subtraction on them.

King - man + woman = Queen

It is therefore possible to find vectors representing these operations. For example, the vector that is used to go from "Paris" to "France" should be able to be used to go from "Stockholm" to "Sweden".

Flairembeddings (Contextual String Embeddings)

Contextual String Embeddings passes sentences as sequences of characters into a character-level language model. They are trained without any explicit notion of words and as a result of that model words as sequences of characters. They are also contextualized by their surrounding text which means that the same word will end up having different embeddings depending on its context (Akbik et al., 2018). All this is done by utilising the hidden states of a forward-backward recurrent neural network (RNN). This makes the model not only able to understand word semantics like an ordinary word embedding but the actual contextual semantics too. In FLAIR, these embeddings go under the category 'FlairEmbeddings'.

Byte-Pair Embedding

Byte Pair Encoding (BPE) is a variable-length encoding that views text as a sequence of symbols and iteratively merges the most frequent symbol pair into a new symbol. E.g., encoding an English text might consist of first merging the most frequent symbol pair {**t, h**} into a new symbol {**th**}, then merging the pair {**th, e**} into the in the next iteration, and so on (Heinzerling & Strube, 2018). BPE was applied to all Wikipedia articles that were large enough and embeddings were pre-trained for the resulting BPE symbols using GloVe. What this means is that the word embeddings are precomputed on the subword level and that they can embed any word by splitting the words into subwords and then looking up their embeddings.

Heinzerling and Strube (Heinzerling & Strube, 2018) also found that while offering nearly the same accuracy as word embeddings they only take up a fraction of the model size which makes them great for small models.

Transformer Embedding

Transformer Embeddings is a type of embedding used in the FLAIR framework to use the embedding layer of popular transformers-based models such as BERT, RoBERTa and XLM-R. It allows you to pair the powerful transformer model embedding layer with a traditional LSTM model. The transformer is a model type created by Google 2017 that is based solely on attention mechanisms. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train (Vaswani et al., 2017) compared to current state-of-the-art models at that time.

2.5 Spatial data

Spatial data, or geospatial data, is a type of data that represents some geographical area or location. Coordinates, for example, are a type of spatial data, and they can be represented in different ways. Regular coordinates, latitude and longitude, are geographics. These are called spherical coordinates. Working with spatial data, you can work with the geometry type or geography type. Geometric data is a data type that is mapped on a two-dimensional flat surface, while geographic data is mapped around a sphere. The geography data type supports spatial features for so-called geodetic coordinates, meaning latitude/longitude. Depending on your use case, you want to pick one of these. The geography type is recommended when working with coordinates that cover larger areas (Hsu & Obe, 2021).

2.5.1 GIS

Geographic Information Systems, shortened as GIS, does not have a definite definition. Chang (2008) defines it as "a computer system for capturing, storing, querying, analyzing, and displaying geospatial data.". GIS can make working with data from different sources and locations much easier. Using GIS, geographic features can easily be stored in a type of

database, specially created for this type of data, which allows for spatial functions to be used on the data.

2.5.2 Spatial databases

A spatial database gives you the possibility to do spatial analysis on top of a regular relational database. Just like regular database systems, indexing can be used to speed up operations, in this case, spatial indexing. Spatial indexes speed up the use of a spatial database, by organizing the data into an efficiently traversable structure instead of being required to do a sequential scan of the database every time a query is done.

Some general requirements for a spatial database system are that it is a database system, it offers the use of spatial data types and it offers the use of spatial indexing (Güting, 1994). A spatial database system is essentially a regular database system with added-on capabilities for spatial types, spatial functions and spatial indexes. This can be done either as some standalone solution or as an extension of existing database systems, such as PostGIS for Postgresql, Spatialite for SQLite.

Another important attribute of spatial databases is the support for spatial functions or spatial queries. These are queries that examine the relationship between spatial data, for example, the distance between two points.

2.6 Models

In this section we will present background of some of the models used in this thesis.

2.6.1 BERT

BERT (*Bidirectional Encoder Representations from Transformers*) is a language representation model developed by Devlin et al. (2019) in 2018 and based on the original Transformer implementation by Vaswani et al. It is designed to pre-train deep bidirectional representations from unlabeled text and does this by using joint conditions on both the left and the right context in all layers as opposed to previous models that could only look from left-to-right or vice-versa. It uses two different unsupervised pre-training methods. The first one is known as Masked Language Modelling, often referred to as the *Cloze-task* (Taylor, 1953) where you take a percentage of the input words at random and mask them with a special token. You then try to predict the word using only the surrounding words in the input. In the original paper, they masked 15% of the data.

The second pre-training method is known as Next Sentence Prediction (NSP) and is used to give the model some understanding of sentence relationships. It takes two sentences, **A** and **B** to try to decide whether the sentence following **A** is **B** or if its a random sentence from the corpus. BERT uses the BooksCorpus and the English Wikipedia containing 800 and 2500 million words respectively for its pre-training.

2.6.2 Danish-Bert

The Danish BERT model is basically the normal BERT-model pre-trained on a Danish corpus consisting of data from the Danish Wikipedia, Common Crawl, debate forums and the Danish OpenSubtitles giving it close to 93 million sentences. It uses the original BERT model to create an ecosystem around standardized models instead of newer models (such as AlBERT and RoBERTa) that could have given minor performance improvements.

2.6.3 Libpostal

Libpostal is a C library for parsing and normalizing street addresses around the world using statistical NLP and open data. It is an open-source project implemented using Conditional Random Field, commonly known as CRF. The CRF model is trained using an averaged perceptron. The big difference to other publicly available models at the time of creation was its ability to train without storing the entire training set in memory which was common for CRF models in 2017 and put a hard limit on the amount of data that could be used. It claims an accuracy of 99.45% but since the evaluation data is not publicly available the claim cannot be verified.

As Libpostal uses its own normalization process it currently supports up to 60 languages and can parse addresses in more than 100 countries.

2.7 Libraries

In this section we present the libraries and frameworks used for the models used in the thesis.

2.7.1 PyTorch

With the increased interest in machine learning, there has been an explosion of different new machine learning tools. Many of them, such as TensorFlow, construct a static dataflow graph to represent the computation and which you then can apply to batches of data. This provides visibility into the computation ahead of time albeit comes at the cost of ease of use and debugging. Prior work has also seen the value in dynamic eager execution but has implemented it either at the cost of performance or its applicability.

PyTorch is a tool introduced in 2019 by a team at Facebook AI Research lab (FAIR). Its success stems from utilising old ideas to create a design that balances speed and ease of use (Paszke et al., 2019). It strives to make writing and using models and data loaders as easy as possible and to let the complexity of machine learning be handled internally by the tool while keeping it accessible through intuitive APIs. It provides tools that allow its users to manually control the execution of their code and thus allowing them to improve their performance themselves.

2.7.2 FLAIR

FLAIR is a powerful NLP framework developed by Zalando Research (Akbik et al., 2019). It is built on PyTorch and allows you to apply state-of-the-art NLP models such as named entity recognition (NER) and part-of-speech tagging (PoS). The core idea of the framework is to present a simple and unified interface for very different embeddings and thus hide all embedding-specific engineering complexity. This allows users to mix and match various embeddings with little effort. A recommended setup is to stack WordEmbeddings with FlairEmbeddings (a contextualized embedding) for state-of-the-art performance. Another thing with FLAIR is how it addresses the learning rate. For each epoch, it evaluates the model and if it does not see an increase for four consecutive epochs it halves the learning rate. It then stops when it considers the learning rate to be too low.

2.7.3 Transformers

With the success of the transformer (Vaswani et al., 2017) structure and model pre-training such as the BERT model came some practical challenges that needed to be addressed in order for these models to be widely utilized. A system that allowed you to train, analyze, scale and augment models easily was required.

Transformers is a library created by a team at Hugging Face, an AI community, dedicated to supporting Transformer-based architectures and facilitating the distribution of pre-trained models and data sets. It is maintained and updated by a team of engineers and researchers backed by a strong community. It mimics the standard NLP pipeline, all the way from processing data to making predictions. Every model in the library consists of 3 building blocks: a tokenizer, the transformer and the head that makes the predictions. The library is available both in PyTorch and TensorFlow. It can be transferred between these frameworks seamlessly.

2.8 Evaluation Metrics

The task in this thesis is a type of classification task. In a classification task, a model is meant to infer which category an input should belong to. For example, if we have pictures of cats and dogs, we want to figure out which of two classes, cats or dags, each picture belongs to.

Depending on the type of machine learning model used, certain methods of evaluation will be preferred. For a classification task, we will generally use performance metrics that are based on the following four outcomes:

- True positive (TP) means an example is correctly labeled as positive.
- False positive (FP) means an example is incorrectly labeled as positive.
- True negative (TN) means an example is correctly labeled as negative.
- False negative (FN) means an example is incorrectly labeled as negative.

Using these four classifications we can get a number of metrics.

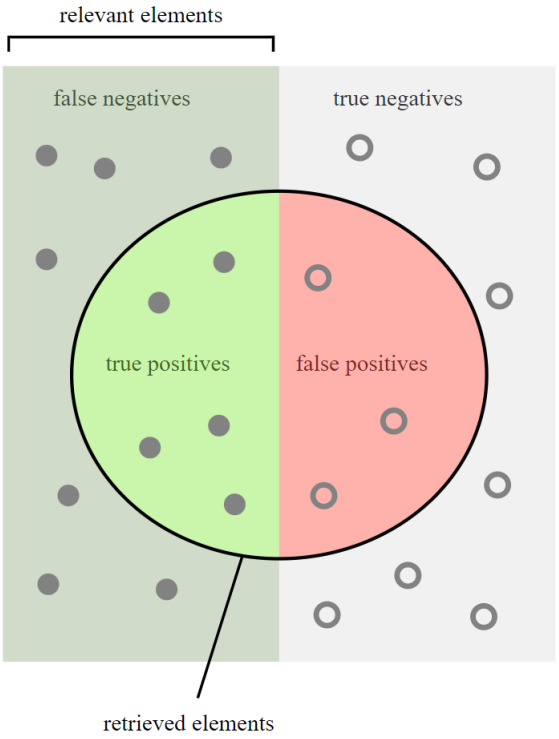


Figure 2.8: Visual explanation of TP, FP, TN, FN (Walber, 2014)

Accuracy

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

Accuracy does not always give a fair view of the performance, as it can be sensitive to imbalanced data, which is why some other metrics are often used (Goodfellow et al., 2016) (Tharwat, 2020).

Precision

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

Precision, also known as positive predictive value corresponds to the fraction of correctly classified positive samples out of all positively classified samples.

Recall

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

Also known as true positive rate, hit rate, sensitivity. This corresponds to the fraction of correctly classified positive samples out of the total number of positive samples in the set.

F1-score

$$F1 - score = \frac{2 * TP}{2 * TP + FP + FN}$$

F-measure, commonly called F1-score, is the harmonic mean of precision and recall.

Multi-classification

These types of metrics have been explained in the context of binary classification, but they can easily be extended to multi-classification tasks. In that case, a prediction will be a true positive if it is classified as the correct class, or a false negative if it is classified as anything else.

Averages

Evaluating a classifier on all the classes is important and gives a good idea of what classes the classifier has an easier or harder time with. To get a sense of how the classifier performs overall, an average can be calculated. There are a few different ways to do this, two of which will be presented here.

Micro average is an average over all samples, i.e equal weight will be given to each sample. Doing this can skew the result towards the larger classes (Manning et al., 2010).

An alternative is to use ***Macro average*** which is an average over all the classes, i.e equal weight will be given to each class. Calculating the macro average can give a better sense of the result when some of the classes are smaller (Manning et al., 2010). When presenting our results, we will use a macro average of the f1-score, for which the formula is:

$$Macro\ avg.\ F1 - score = \frac{1}{N} \sum_{i=1}^{N} F1 - score_i,$$

where N is the number of classes.

Confusion matrix

A good way to visualize the predictive performance of a multi-class model is to create a confusion matrix. The matrix shows the relation between the predicted classes and true classes, where the ideal is a perfect diagonal score, meaning that every sample was correctly

classified. This is a useful tool to see what classes often get mixed up, which can indicate how to make improvements to the model. In Figure 2.9 we see an example of a multi-class confusion matrix. In this case, six samples of each class were predicted. We can see that of the six "STREET" samples, five were classified correctly, and one was incorrectly classified as "CITY". Similarly, of the six "CITY" samples, one was incorrectly classified as "STREET". All "POSTCODE" samples were correctly classified. When dealing with a big amount of samples, it can be a good idea to normalize the matrix to make sense of the numbers.

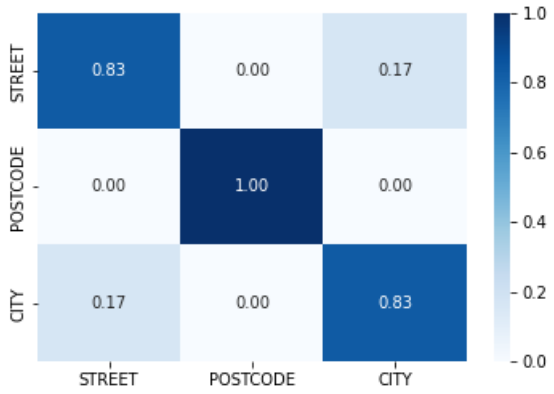


Figure 2.9: Normalized confusion matrix with three classes.

Chapter 3

Data

In this chapter, we will introduce the data used in our thesis.

3.1 Data set

As earlier mentioned the data set used in this thesis consists of Danish addresses. As the client was unable to share their data we had to acquire it elsewhere. It was acquired from the Danish Data Authority (Dataforsyningen) using their web API. It consists of 3,833,076 entries, covering all Danish addresses. This leaves us with the following fields used in our thesis:

- Full address
- **Street name**
- **House number**
- Floor
- Unit
- Supplementary city name
- **Postcode**
- **Postal district**
- Coordinates (Longitude/Latitude)

3.2 Danish addresses

The Danish address has four required fields (marked in bold above) but can, as seen in the previous section, include 7 fields. This makes the Danish addresses vary in difficulty ranging from very easy to parse up to the example further down with equal fields. An example containing all fields can be seen below:

Kronprinsessegade 7 st. th, 1306 København K

3.2.1 Level and Unit

Levels in Danish buildings are written using numbers except the ground floor and basement. In addition, the unit is often designated by its location and not with a number but can be done with both or a combination of the two.

Abbreviation	Danish	English
st.	Stuen	Ground floor
kl.	Kaelder	The basement
tv	Til venstre	To the left
th	Til hoejre	To the right
mf	Midtfor	In the center of the floor

3.2.2 Postal district and supplementary city

Most fields are common around the world but Danish addresses have two fields that are a bit uncommon: postal district and supplementary city name.

A postal district is usually associated with a city or municipality. In Denmark, this is not always the case. Smaller areas may share the same postal district while larger cities, such as Copenhagen, are divided into several (København S, V, N, K and Ø).

The supplementary city name, in Danish known as *supplerende bynavn*, is an supplementary city name linked to a group of addresses when it is appropriate to specify their location within the municipality or postal district. Approximately 30% of Danish addresses are associated with a *supplerende bynavn*. Any local place name can be used as a supplementary city name, e.g. the name of a district or a larger building in the area. In other cases, the name of a land area or e.g. an island can be used. In most cases, however, it is the name of the town or village which means that sometimes the supplementary city name and the postal district can be the same. In our complete data set there are 36827 such entries which correspond to approximately 1% of the data. An example of this can be seen below:

Hyldeblomsthaven 18, Havdrup, 4622 Havdrup

3.2.3 Test set

Due to limitations in our hardware, we chose to run the tests with a subset of the full test set instead of using our full test set containing more than 500,000 entries. The division of data can be seen in Figure 3.1. The division of data was originally done geographically on the full data set. We used the regions of Aalborg, Aarhus and Odense for our validation data and the region of Copenhagen as our test data. This gave us a rough split of 70-15-15. When creating our smaller subset we took data from these larger sets but with a 75-12.5-12.5 split. It gave us a somewhat fairly represented data set which was later oversampled, see Chapter 5.

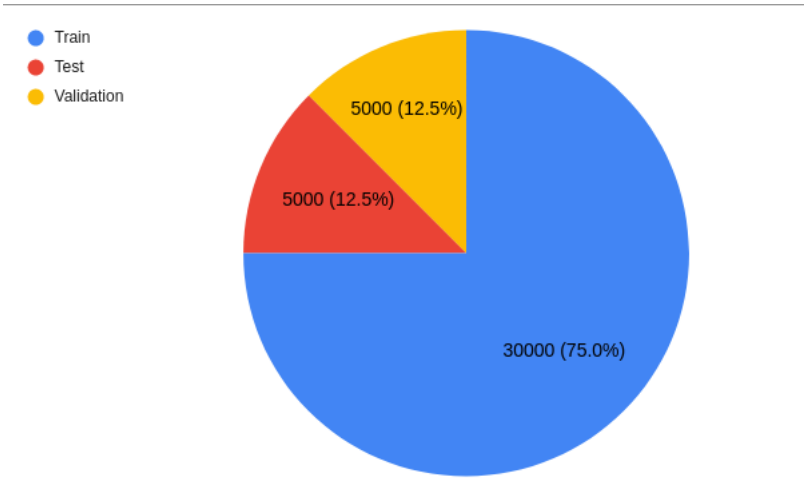


Figure 3.1: Division of data between test-, train- and validation data.

Chapter 4
Methodology

In this section, we will give an overview of the methodology used in this thesis, along with our approach to answer the research questions. We will also describe our experimental setup as well as the technology and libraries used in the implementation.

4.1 Research Approach

To answer the research questions, an experimental approach was taken. The experiment design looked something like Figure 4.1, where the last four steps were performed for each model.

The approach for RQ1 was to train and evaluate a couple of deep learning models with different architectures and then directly compare their F1-scores. For this part we also did a number of experiments with regard to hyperparameters, to see how these would affect the result. These tests are presented in Section 4.4.4. The goal for this question is to find the combination of hyperparameters that returns the highest score for the non-context model.

RQ2 required us to evaluate libpostal, one of our baselines. While we could not train the model from libpostal ourselves, we ran the same test set that we used fort the previous models through the models prediction. For this question, the goal is to beat the libpostal baseline. A good result for this question is simply one where we score higher than the baseline, when evaluation our model. The evaluation for RQ2 had to be done differently than for RQ1, which is explained later in Section 4.2.

For RQ3, we had to find a way to inject context into the model. We have previously defined context as nearby addresses. For each address in the data set, we queried the spatial database to find its closest neighbouring address. This was used as context. This will be further expanded upon in Section 4.4.5. For this question, the goal is to see an improvement from our models that did not use context. A good result would then be one where the context model scores higher than the non-context models.

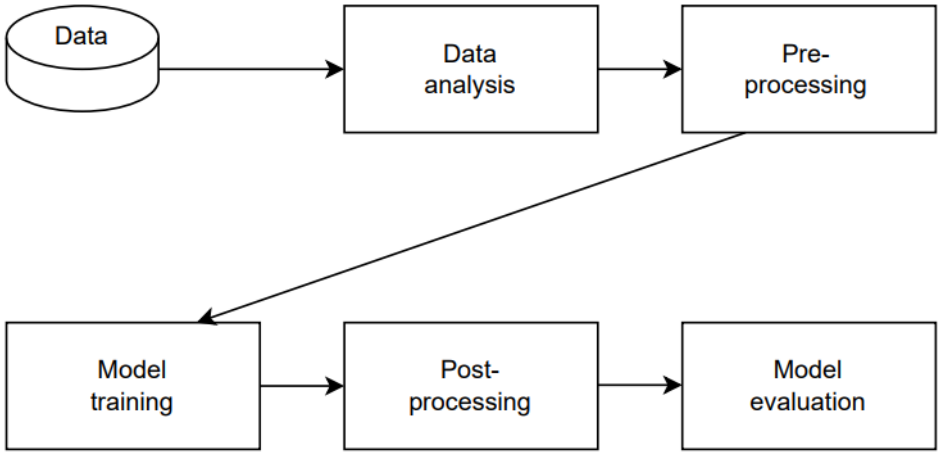


Figure 4.1: The pipeline of our project

4.1.1 Data analysis

One of the primary goals was to acquire and understand the data, as the structure of the data played a large role in the project. Naturally, this was the next step to take after setting the goals for the thesis. This was done by reading theory, contacting Danish authorities with questions about the address format and looking at the data using exploratory analysis. We then divided the data into geographical regions, see Section 3.2.3.

4.1.2 Pre-processing

The only pre-processing done was the tokenization. *FLAIR* uses segtok to tokenize on whitespaces for its models and for our BERT-models using *transformers* we used a BERT tokenizer that is based on Wordpiece. The main difference between these two in this use-case is that the dot ('.') belonging to the *level*-tag becomes an individual token with BERT as opposed to belonging to the *level*-token. This will be discussed further down.

4.1.3 Post-processing

After training the model and before evaluating it, we need to do some post-processing. We extracted all our fields from the database and created tuples such as **('Street', 'Danskevej')**. We then fed the entire address into our machine and the predictions were presented as tuples such as the ones in our comparative data. The final step was to go through the comparative list that we knew were true and make sure that every token existed in the right place in our prediction.

4.2 Evaluation

In section we will describe how the evaluation was performed for the different experiments, and adaptations that had to be taken to account for the different frameworks. We will also define what we consider to be a good result.

4.2.1 Full comparison

As Libpostal does not produce the same labels as the other models we had to compare it differently as no tool could produce the normal measurements (such as f1-score) easily. We, therefore, used the following measurements:

- **Full Parse Accuracy**: We counted the number of entries that had every label in them correctly and divided it by the total amount of entries.
- **Accuracy:** The normal measurement. We counted the number of labels that were correct and divided by the total amount. We also did this for every label type providing us with the accuracy of each label.

This was only used for the comparison between all models in Table 5.1. An example of how this looks can be seen below in Figure 4.2:

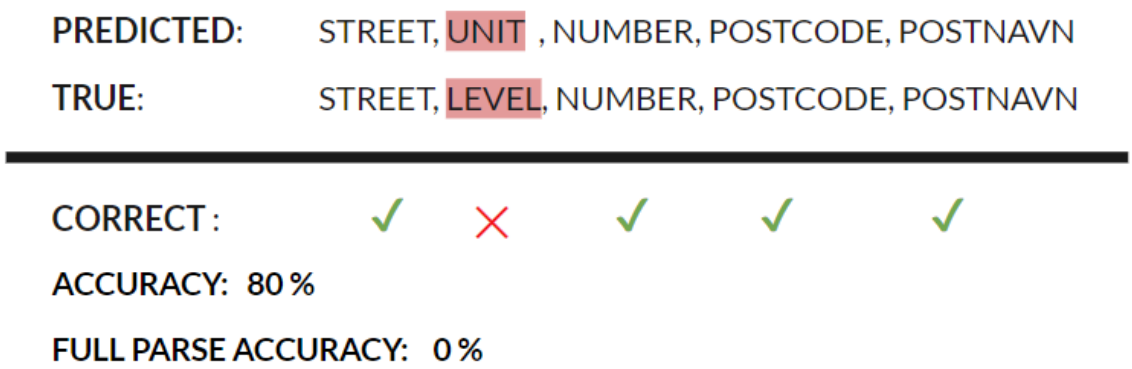


Figure 4.2: The way accuracy and FPA is calculated

4.2.2 FLAIR

After testing the model, the FLAIR framework used for the no context models, return a classification report containing performance metrics such as accuracy, f1-score and macro average by default. This can also be called using the method *model.evaluation* which lets you easily evaluate different test sets. Using the information returned after testing, confusion matrices were constructed to get a better understanding regarding the misclassification of classes that were returning lower scores. These confusion matrices can be seen in Figures 4.3, 4.4, and 4.5.

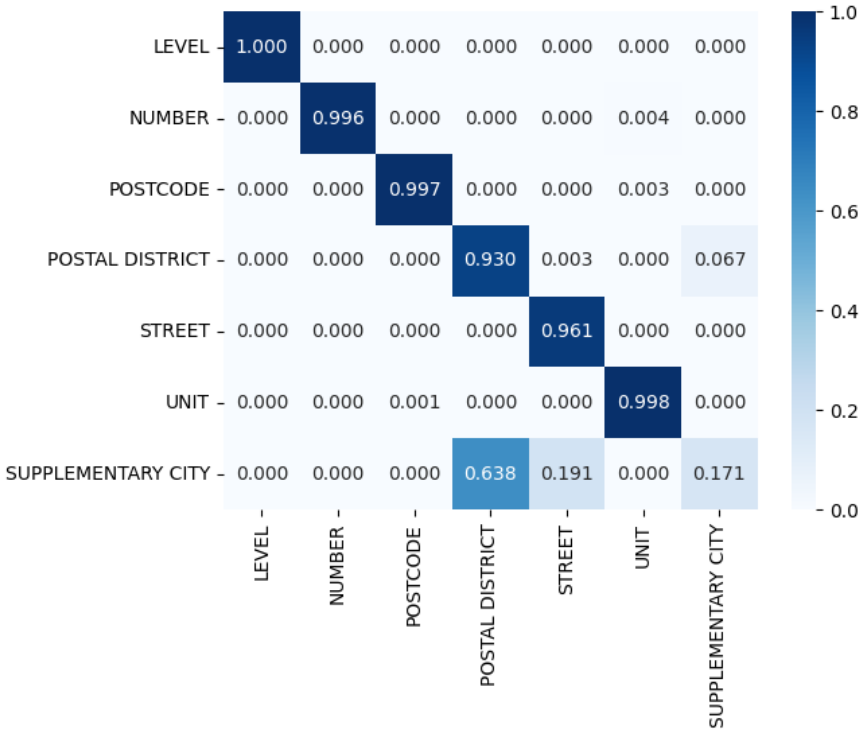


Figure 4.3: Confusion matrix depicting the test results with Flairembeddings

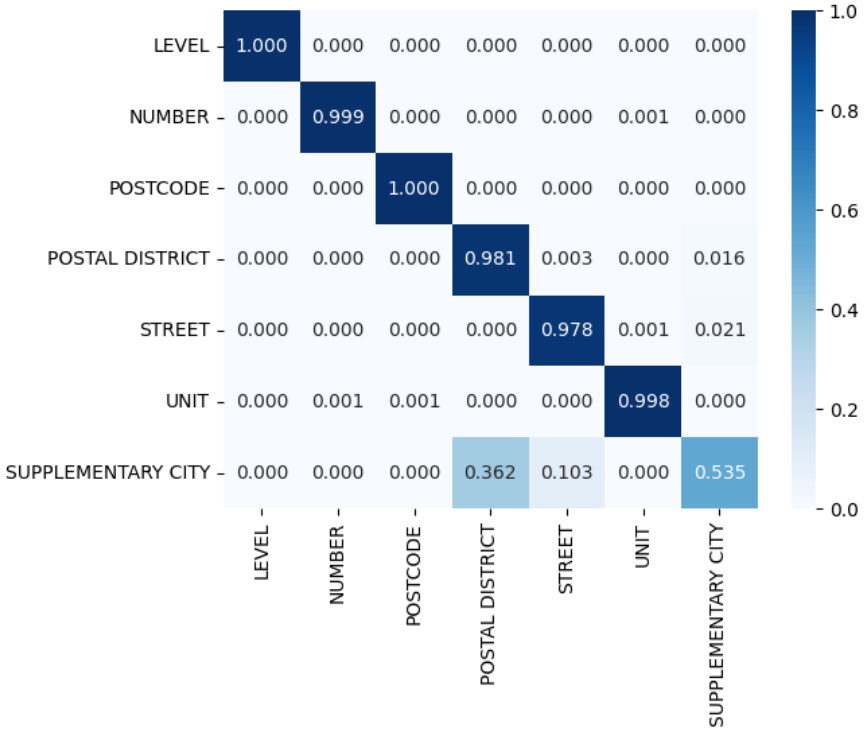


Figure 4.4: Confusion matrix depicting the test results with BPEemb.

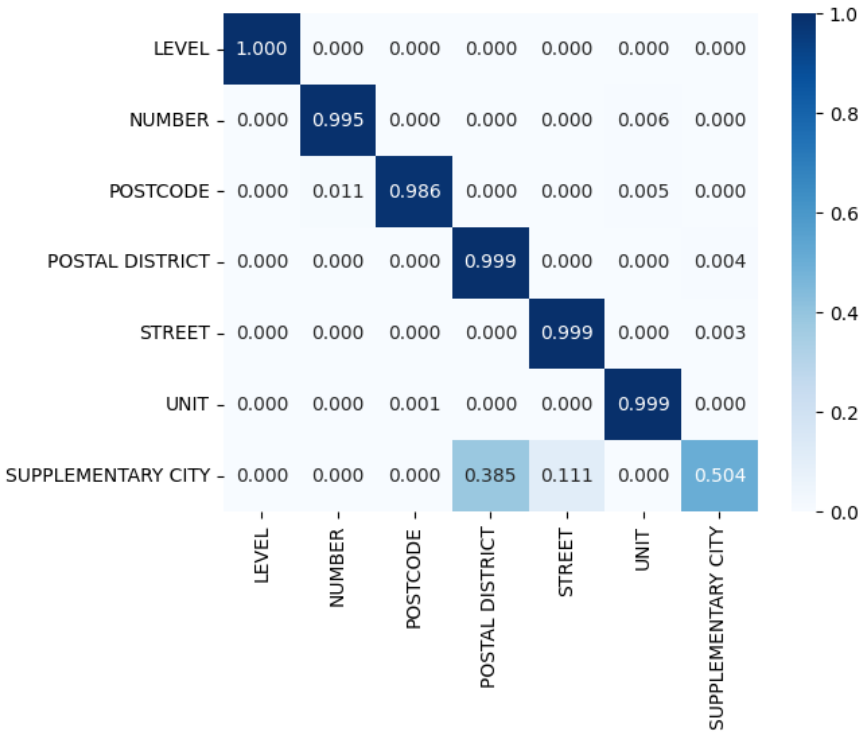


Figure 4.5: Confusion matrix depicting the test results with Transformer-embedding BERT.

4.2.3 Transformer

For the Transformer models, evaluation was done using the Python framework seqeval (Nakayama, 2018). Seqeval is used for the evaluation of sequence labeling and returns a classification report like the one returned by FLAIR. We confirmed that seqeval produces the same result as the evaluation from FLAIR so that they can be compared on equal terms.

4.2.4 Libpostal

As Libpostal does not use entirely the same labels as we do when using our models, we mapped their predicted labels to match ours, see Table 4.1.

4.3 Validity threats

As mentioned previously in Section 4.2, Libpostal does not return the same type of labels as our other models. We tried to fix this by introducing accuracy and full parse accuracy as metrics. It would have been preferable to use F1-score for the libpostal model as it usually a preferable metric to accuracy in NLP tasks.

Throughout the process of the thesis we were given a lot of suggestions and guidance from our different supervisors, all more experienced than us in the subject of our thesis. It

is not unthinkable that had we not had our supervisors, some things might have been done differently, such as the choice of models and what libraries to use.

One type of threat is how generalizable the experiment is. As we will only be using Danish addresses in this thesis, an external threat that immediately stands out is how generalizble the results can be considered to be. While it might be applicable to some countries with similar address structures, e.g. Sweden, this might not be the case where a different address structure exists, e.g. Nicaragua where street names in the traditional sense do not exist but instead are a sort of description relative to a central point (Rhind, 2020).

4.4 Models

In this section we will present the different models trained and evaluated in this thesis, along with the hyperparameter tests performed for the model without context.

4.4.1 Libpostal baseline

We used Libpostal for our baseline. Libpostal uses a special metric called *full parse accuracy* which counts a prediction to be correct only if all individual fields have been correctly assigned. In addition to normal accuracy we will therefore present *full parse accuracy* for the comparative results.

Table 4.1: Labels of Libpostal compared to the ones used in this thesis.

Libpostal	Ours
Road	Street
House_number	Number
Level	Level
Unit	Unit
N/A	Supplementary city
Postcode	Postcode
City	Postal district

4.4.2 BERT model without context

To make a full comparison between context and no context we also made a baseline using BERT with PyTorch without context. The BERT model that we used was a Danish model called Danish-Bert-BotxoWe used the default parameters which were the following:

Loss function	Batchsize	Finetuning
BERT	32	Yes

For the BERT model default settings were used. The default optimizer used for BERT is AdamW.

4.4.3 FLAIR model without context

The *FLAIR* framework is easy to use for sequence labeling tasks. This made it a perfect starting point for us. By just changing one line of code we could change what embedding to use which completely changed the model. The data was entered from a simple text file with an easy-to-understand structure as can be seen in Table 4.2. We used the default[1] parameters given by *FLAIR*, with the optimizer being SGD, along with BPE embedding.

Table 4.2: An example of the data used in training and evaluation with randomized order and fields. "VEJBY" is equivalent to supplementary city.

Token	Label
Hf.	B-VEJBY
Venners	I-VEJBY
LYST	E-VEJBY
Bøllemosegårdsvej	S-STREET
2100	S-POSTCODE

4.4.4 Tests

We used default parameters when training but did a number of tests to see how changing the hyperparameters affected the result, listed below. All tests were done using an LSTM-model trained through FLAIR with a Byte-Pair Embedding except for the test where we compared different embeddings to each other.

1. **Embeddings**

   Every embedding has its advantages and for us to figure out what would suit the project best we would have to try different ones. Since FLAIR allows you to change the embedding used (or even combine multiple) with just one line testing multiple was easy. We decided to start with the recommended approach by FLAIR: stacking backwards and forwards flair embeddings with GloVe word embeddings. We then removed the GloVe embedding and later changed the flair embeddings for BPE and then later transformerembedding (using Danish-BERT-Botxo). This means that a comparison was made in FLAIR between the following three embeddings:

   - **Transformerembedding** using Maltehb/Danish-bert-botxo
   - **Flairembeddings**, stacked multi-language backwards and forwards
   - **Byte-Pair Embedding (BPEmb)**

[1] https://github.com/flairNLP/flair/blob/master/flair/models/sequence_tagger_model.py#L27

2. Learning rate

Learning rate is something that greatly affects the accuracy of the model. When training an LSTM model using FLAIR the team behind FLAIR suggested using a learning rate of 0.1 however since LSTM models have a wide range of good learning rates we wanted to see how it affected our model. With the transformer model, it was more straightforward. Generally, a low learning rate is preferred, and when creating BERT Devlin et al. (2019) used a learning rate of $2e^{-5}$ which is the same learning rate recommended with the *transformers* library.

The tests were performed on the LSTM-model created with *FLAIR* using BPEmp.

3. Incompletion & disarrangement

Inputting the full address in correct order results in a very easy problem that could be resolved without machine learning by splitting on ',' or looking for certain keywords. But when someone looks for an address they rarely enter the full address and sometimes they even enter it in the wrong order. Both Li et al. (2014) and Sharma et al. (2018) suggest that the order of the data matters and it can be accounted for by either training the model on randomized data or by reordering it before feeding it into the machine. In this paper, we decided to train the model on randomized data and see how it affects the results. In addition to swapping the order of the fields, we also introduced randomness to whether certain fields would appear or not. Street, postcode and postal district all have a 50% chance of appearing in the entry regardless of the other ones, creating blanks.

We do not address blank addresses as it is unlikely that anyone would search for a blank address.

4. Size of training set

With many research-orientated machine learning programs, it is assumed that the entire data set fits in your RAM. This essentially means that without time-consuming optimizations that performance scales with your hardware (in addition to speed). As we were very restricted in our hardware we trained the model on a smaller set. To see how the size of the training data set affects the accuracy we did run one test comparing different training sizes presented in Figure 5.2.

5. Oversampling

As seen in our exploratory analysis of the data, the supplementary city name, appears in few entries. In our initial test set, there were only 24 entries out of 5000 containing this field and our training set contained 9139 out of 30 000. We, therefore, decided to oversample so that 50% of the entries in training and test contained this field to see how it affected the accuracy of the model on this field.

As FLAIR was developed to be used for quite specific tasks, we were unable to find an easy way to modify the code to use context. Due to the complexity, we deemed that it would take too long to modify it, which meant that we had to switch framework.

4.4.5 BERT model with context

We chose to work with transformer as it is considered to be state of the art (Vaswani et al., 2017), despite being developed five years ago, and it was also suggested from multiple parties that we used it. The most common transformer is BERT and we chose to use the basic version that had been trained on Danish data instead of any of the other versions available such as RoBERTa due to BERT being the most common.

The transformer was implemented using one of the tutorials of the *transformers* library called *run_ner_no_trainer.py*. This tutorial was chosen as it provided every part of the process exposed and modifiable. This allowed us to easily understand how you used PyTorch to train a model even though we had never used it before.

To be able to use our own data, we needed to modify it to work together with the *transformers* data loader. This was done by copying and modifying the data loading script of another data set available through *Hugging Face*, allowing us to use the same format on our data files as with FLAIR.

The next step was to add context. We already knew that the BERT tokenizer could take two sentences, sentence A and sentence B, as input so by adding our context to our data file with O-labels (*outside*) we could later separate it from the actual data and feed both sentences into the tokenizer. One issue with this was that almost every label came out with an O-label. By modifying the loss function, we managed to remove every token with an O-label so that it was ignored by the loss function, which rectified this issue.

4.5 Implementation

In this Section, we present some of the technological features we used during the thesis along with some parameters.

4.5.1 Environment

The department of computer science has supplied us with a lab computer with a lot of RAM and a high-end CPU to run the training of the models on but since a high-end GPU decreases the run time enormously and none was available in the department we decided to run it on one of our private machines. This caused some problems when trying to run the whole set due to lack of RAM however downsampling the data set removed this issue. Hardware and software versions used can be seen in Table 4.3 and Table 4.4.

Table 4.3: Hardware used in this project.

CPU	GPU	RAM
AMD Ryzen 5600x	Geforce GTX 1080 Ti	32GB DDR4 3600MHz

Table 4.4: Software versions used in the paper.

	Version
FLAIR	0.10
Transformers	4.15.0
Libpostal	1.1
Ubuntu	20.04 LTS
Python	3.8.10

4.5.2 Postgres/PostGis

We decided to use PostgreSQL 12.9 along with the extension PostGIS 3.0 after an unsuccessful attempt to use Spatialite for SQLite. We used the geography data type. PostgreSQL allows for various spatial indexing methods, for example, Generalized Search Tree (GiST) which is the most commonly used method and is what we ended up using.

Our main table named "addresses" consists of 3 832 933 rows with 11 columns. Three of the columns (level, unit, supp. city) may have a null value. All fields except for the *geog* field were entered as-is from the data. The "geog" value was calculated from the latitude and longitude coordinates of the address, using the built-in geography type constructor in PostgreSQL. An example can be seen in Table 4.5.

After determining how to split the data into our training, validation and test sets, we created three new tables in our database depending on the municipality.

Table 4.5: Example entry in our database.

address	Mælkevejen 69D, 1440 København K
street	Mælkevejen
number	69D
level	null
unit	null
supp. city	null
postcode	1440
postal district	København K
municipality	København
geog	0101000020E61000007E4EC5B6DA332940AE0E3E9C59D64B40

4.5.3 Overview of data sets used in the tests

- **Normal**: The normal data set consisting of 30 000 entries for training, 5000 for testing and 5000 for validation.
- **Randomized**: The normal set that has been randomized as described in Section 4.4.4.
- **Oversampled**: A data set that is almost the same as the randomized one however we have oversampled as described in Section 4.4.4.

Chapter 5
Results

5.1 Comparison all models

In Table 5.1 below we present a comparison of the accuracy of the models presented in this paper. Total label count means the number of entries where it has all labels correctly labeled. We can see that Libpostal is easily beaten by our BPEmb model however that our final model, BERT with context, falls a bit short being held back by its accuracy on levels.

Table 5.1: Comparison of all models.

Label	Libpostal	BPEmb	BERT	BERT with context
Street	71.46%	76.02%	93.95%	**95.64%**
Number	78.74%	**98.85%**	83.88%	93.52%
Level	1.38%	**57.97%**	1.85%	2.14%
Unit	0.33%	65.58%	83.56%	**88.76%**
Supplementary city	0.00%	**55.52%**	34.78%	40.58%
Postcode	62.06%	**99.34%**	84.04%	90.28%
Postal district	67.54%	43.91%	85.71%	**99.86%**
Accuracy	59.52%	71.86%	75.95%	**83.23%**
Full Parse Accuracy	16.12%	**37.88%**	11.68%	15.28%

5.2 Baselines

5.2.1 Libpostal

As we can see in Table 5.2, Libpostal performs worse than expected on normal data and with a terrible full-parse accuracy when we introduce disarrangement and incompletion.

Table 5.2: Libpostal baseline. Random means that the data it has been tested on has been randomized as per section 4.4.4

Testdata	Full Parse Accuracy	Accuracy
Normal	**56.91%**	**87.23%**
Random	16.12%	59.52%

5.2.2 BERT model without context

Before modifying the transformer to include context we had to establish a baseline using the transformers library. The results can be seen in Table 5.3.

Table 5.3: BERT baseline with randomized data.

Label	Precision	Recall	f1-score
Level	0.9000	0.0124	0.0245
Number	0.9626	0.9690	0.9658
Postcode	0.9994	0.9994	0.9994
Postal district	0.9473	0.7523	0.8386
Street	0.8728	0.9368	0.9037
Unit	0.3351	0.8947	0.4877
Supplementary city	0.5462	0.6005	0.5721
micro avg	0.8564	0.8535	0.8550
macro avg	0.7948	0.7379	0.6845
weighted avg	0.8867	0.6005	0.5720
F1 Score	0.8524		
Accuracy Score	0.8413		
Precision Score	0.8644		
Recall Score	0.8528		

5.3 Hyperparameter tests

In the first experiment, we experimented with different embeddings for FLAIR to evaluate which one would best suit our needs. We also tested how different learning rates affected the result and oversampling of the supplementary city name as it was underrepresented compared to other labels.

5.3.1 Embedding comparison

As we can see in Table 5.4 BPEmb wins in every category except for accuracy which instead is won by the flairembedding. BPEmb claims a much better prediction speed and size compared to the other embeddings while also claiming a better f1-score which is impressive.

Table 5.4: Comparison done with 0.1 LR, 20 epochs, randomized data.

Embedding	Predictions/s	Size	F1-score (macro)	Time to train	Accuracy
Transformer(BERT)	137	453.4 MB	0.8253	18 min	81.05%
BPEmb	**445**	**28 MB**	**0.8601**	**8 min**	71.86%
FlairEmbeddings	47	448 MB	0.8588	150 min	**81.97%**

5.3.2 Learning Rate comparison

Figure 5.1 shows the results from the comparison of different learning rates when training a model in FLAIR using Byte-Pair embedding. We can see that a higher learning rate of 0.1 performs best and requires the least amount of epochs while the lowest learning rate of 0.0001 does not get close even though it runs for 200 epochs.

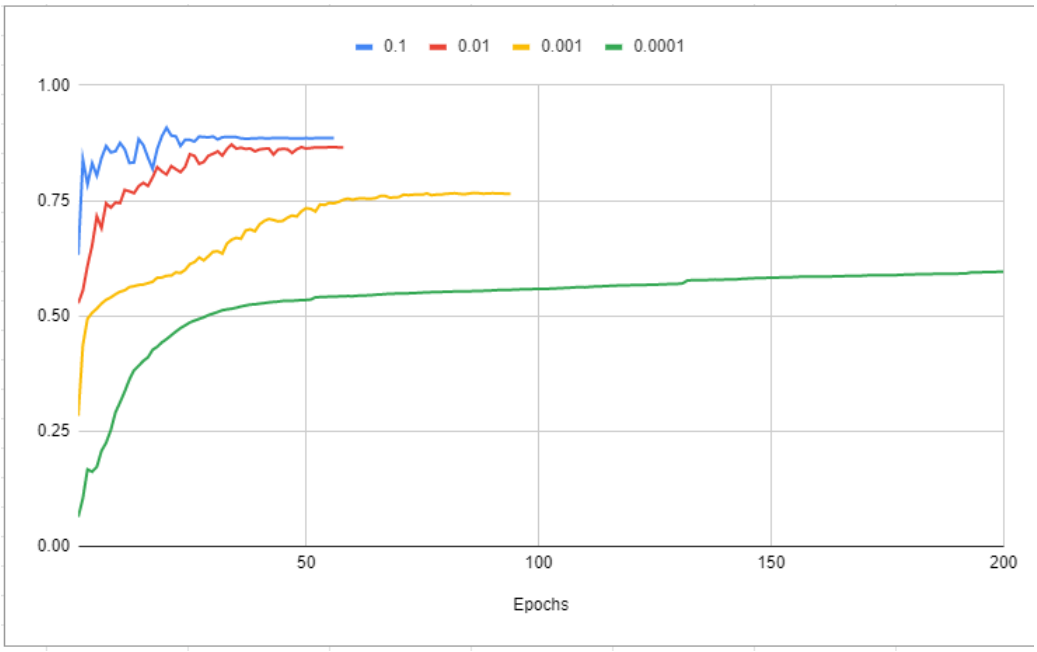


Figure 5.1: Test with different learning rates

5.3.3 Effects of incompletion & disarrangement

Using incompletion and disarrangement to create a random data set we trained models on both random and normal data and tested both on random and normal data. The results can be found in Table 5.5. We can see that a model trained on random data performed better on the random data than the model that was only trained on normal data while still not losing too much performance on normal data.

Table 5.5: Testing normal and randomized model on randomized data. Test on normal data for reference.

Trained on	F1-score (macro)	Test-set
Randomized	0.9229	Normal
Normal	0.9720	Normal
Randomized	**0.8542**	Randomized
Normal	0.5219	Randomized

5.3.4 Size of training set

As we can see Figure 5.2 a model performs better the more data is has been trained on. Training was done with eight different training set sizes, with the largest containing 100 000 entries.

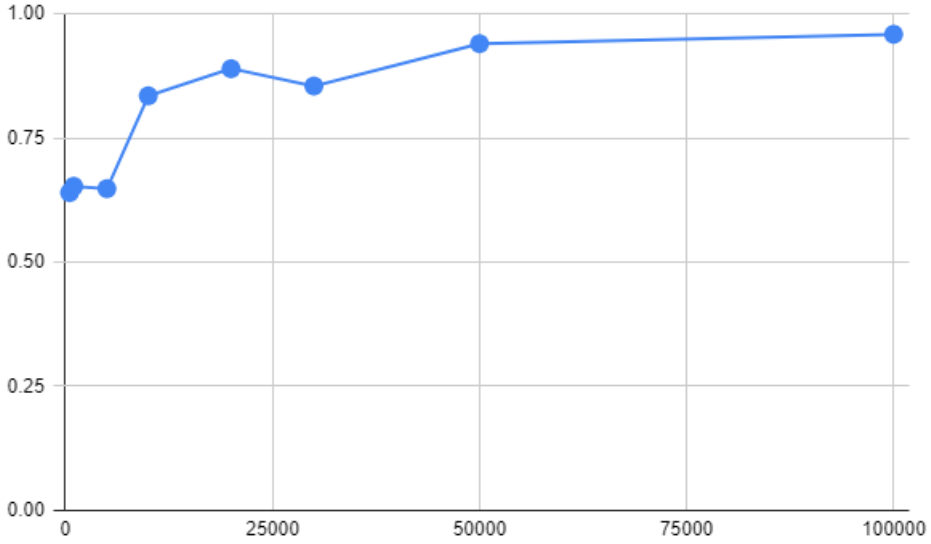


Figure 5.2: F1-score depending on size of training set.

5.3.5 Oversampling

In Table 5.6 we present these results. We can see that the model trained on oversampled data actually performs worse than the one that was trained on normal data.

Table 5.6: Comparison done between models trained on normal and oversampled amounts of supplementary city name entries, oversampled from 0.7% to 50%. Tests were done both on normal and oversampled test data.

Training data	F1-score (macro)	Test data	F1-score (Supplementary city name)
Normal	**0.8542**	Normal	**0.3129**
Oversampled	0.7583	Normal	0.0478

5.4 BERT model with context

In this section, we present the results of running BERT with context. We ran it both on non-randomized 5.7 and randomized data 5.8. We see here that training on a random data set only offers a slight increase in performance compared to a normal one. While resulting in a good overall performance the label-wise performance for *level* is very low.

Table 5.7: Non-randomized data with context.

Label	Precision	Recall	f1-score
Level	1.0000	0.0124	0.2449
Number	0.9779	0.9908	0.9843
Postcode	1.0000	1.0000	1.0000
Postal district	0.9587	0.9614	0.9600
Street	0.9692	0.9894	0.9792
Unit	0.3221	0.9424	0.4801
Supplementary city	0.8203	0.6910	0.7501
micro avg	0.9144	0.9238	0.9191
macro avg	0.8640	0.7982	0.7398
weighted avg	0.9441	0.9238	0.9148
F1 Score	0.9237		
Accuracy Score	0.9146		
Precision Score	0.9101		
Recall Score	0.9212		

Table 5.8: Randomized data with context.

Label	Precision	Recall	f1-score
Level	0.3333	0.0069	0.0135
Number	0.9441	0.9872	0.9652
Postcode	0.9968	0.9974	0.9971
Postal district	0.9756	0.9776	0.9766
Street	0.9587	0.9834	0.9709
Unit	0.3499	0.8898	0.50233
Supplementary city	0.8548	0.7354	0.7906
micro avg	0.9182	0.9278	0.9230
macro avg	0.7733	0.7968	0.7452
weighted avg	0.9217	0.9278	0.9164
F1 Score	0.9216		
Accuracy Score	0.9238		
Precision Score	0.9249		
Recall Score	0.9323		

Chapter 6

Discussion

6.1 Comparison with baselines

6.1.1 Libpostal

Our first objective of this thesis was to see if we could beat the score of Libpostal. As can be seen in Table 5.2 Libpostal run on randomized data resulted in an accuracy of 59.52% which was easily beaten by all of our models trained in FLAIR as can be seen in Table 5.4. It also had a full-parse accuracy of 16.12% which is far from the claimed 99.45%. What is clear is that Libpostal could easily be beaten by a model in an area where the model is only trained on that area however as Libpostal is claimed on data from all over the world it does not entirely make it a fair comparison.

6.1.2 BERT model without context

Our second baseline to beat was the transformer-based model that was not trained using geographical context. The result from the baseline can be seen in Table 5.3. We ran the context-based model both on randomized and standard data and the results can be seen in Table 5.7 and 5.8. As we can see the model using context shows a slight improvement over the baseline with a macro average f1-score of 0.75 compared to the baseline of 0.68. What is more interesting is how the supplementary city name has improved. In the baseline, we had a poor result of 0.57 whereas in our context-based model we got a macro average f1-score of 0.79 which gives us an impressive increase of 0.22. One may be alarmed by the very low results for level and unit but this has to do with the transformers tokenizer which we will go more into in Section 6.8.

6.2 Embedding comparison

One of the first things we did in the first part of the project was to compare how different embeddings performed. As can be seen in Table 5.4, the Byte-Pair embedding wins in almost every category. The results were also presented as confusion matrices in the following Figures: 4.4, 4.5 and 4.3.

6.3 Learning rate

As seen in Figure 5.1 the smaller learning rates perform worse than the larger and take much longer to reach their peak performance. Since FLAIR automatically lowers the learning rate when there has not been any improvement for 5 epochs it is always better to start with a larger learning rate. It also automatically stops training when the learning rate is too low which is why when starting with a higher learning rate it does not run for as many epochs.

6.4 Incompletion disarrangement

Just as expected and in agreement with Li et al. (2014) and Sharma et al. (2018) we could see in Table 5.5 that if you did not take into account the random addresses your performance would suffer. At the cost of slightly lowered accuracy on the normal data set we had a huge increase of 0.33 on the random data set. A comparison that could have been made would be to take the approach of Sharma et al. and instead of training the model on random data we'd instead re-arrange the data to the correct order. This feels like it would be a more difficult problem than just training it on random order and would probably not yield a better result.

6.5 Size of training set

As expected and as can be seen in Figure 5.2 more data yields higher accuracy which is not very surprising. All our tests ran on 30 000 entries which corresponds to 1% of the data due to hardware limitations however if we were to deploy this solution it would be beneficial to train on a larger data set.

6.6 Oversampling

In Table 5.6 we presented the results from tests where we oversampled one of our labels that were underrepresented in the data set. While we see a big improvement using an oversampled model on a similarly oversampled test data set, we see a fall when using the oversampled model on the normal test data set. Since the whole point of oversampling is to increase performance on the normal data set we consider this test a failure. It comes as a surprise since it has already been shown in papers that oversampling improves the results and Viloria et al. (2020) claimed a remarkable improvement in their results by oversampling. A possible

explanation for the poor results might be the very low occurrence of the oversampled field in our normal test set.

6.7 Postal district vs supplementary city

One of the main issues we found during our exploratory data analysis was the similarity between the two fields 'postal district' and 'supplementary city'. In some instances, they were even the same and when they were not one often included parts of the other. We contacted the Danish data authorities to see if there was any logic behind these fields and they informed us that there was none. This makes it very difficult for a machine learning model to understand as there is essentially nothing to learn and is why the supplementary city label has a much lower f1-score than the postal district which can be seen in any of our tests. We successfully increased the accuracy by oversampling which is shown in Table 5.6.

6.8 Level and Unit

While it might seem odd that the results for level and unit saw such a sharp decrease between the FLAIR models and the transformer models, there is likely a simple explanation for this. The reason for this is solely based on tokenization and the structure of the Danish address. As mentioned before the level is always followed by a dot, for example, 'st.'. This in turn makes it very easy for a machine learning model to label correctly. So what is the difference between the FLAIR and transformer model?

The problem lies in its tokenization. FLAIR uses a very simple algorithm for tokenization which puts 'st' and '.' as the same token 'st.'. Transformer on the other hand breaks it up into sub-words resulting in it becoming two different tokens, 'st' and '.'. While it may still seem simple to solve the transformer model had a lot of issues with it and performed very poorly. This can be fixed by changing the tokenizer but as we knew what the issue was we chose not to include it in this thesis due to a lack of time. Similarly since unit always follows level the results of the predictions on unit are also affected by the failure to predict level correctly.

6.9 No context vs context

The big question of this thesis: does adding context help? In short, yes. As per Table 5.3 we got an F1-score of 0.85 using the transformer model without context and 0.92 with context 5.8. While it might be a small increase it shows that using geographical context has merit. Notably, adding context yielded a rather large increase on our weakest label, Supplemental city name, going from 0.57 to 0.79.

Chapter 7
Conclusions

In this thesis, we have experimented with different models and parameters to try to achieve the best result in address parsing. We have confirmed that methods such as randomizing the data and using the right embeddings can benefit you greatly. We have shown that both LSTM and transformer models perform well on the address parsing task (*RQ1*). We have shown that both deep learning models tested in this thesis, LSTM and transformer, performs a lot better than the reference library Libpostal (*RQ2*). Worth noting though is that these are only trained on the type of data that we test on while Libpostal is trained on addresses from all over the world which might make for an unfair comparison. Finally, we have also established that the prediction can be improved by adding geographical context (*RQ3*), however, we believe that further reasearch needs to be done to fully establish how much the geographical context can help.

7.1 Future work

In this chapter we will present some ideas for future work on the topic of our thesis, and how we might have proceeded if we were to continue the work.

7.1.1 Ensemble learning

A potential alternate approach could be to experiment with ensemble learning. Ensemble learning is a method of machine learning that instead of training one big classifier, trains several smaller classifiers which are then combined to obtain a better predictive result. Ensemble models have been shown to outperform single classifiers and can be an appropriate technique to use for many types of tasks (Opitz & Maclin, 1999). We planned to experiment with an ensemble model towards the end, consisting of our FLAIR model, then using output and context etc, then feed into a linear regression model. Due to time constraints, we instead

decided to try a more naive approach using our FLAIR model, closest addresses and then a majority rule decision. However, we were unable to finish this naive experiment in time.

7.1.2 Tokenization

As mentioned in Section 6.8, we saw in our later experiments with the transformer models that the action of removing punctuation from our data set heavily affected the results for the level and unit classes. Experimenting further with the tokenization process, adding more rules, for example, could prove to help get better results more similar to our FLAIR models.

7.1.3 Class distribution

Another area that could be reworked with more in future attempts would be the distribution of classes in our data set. As mentioned previously, to have sufficient data points containing the supplementary city, we used oversampling. Unfortunately, what we did not notice was that many of the data points containing a supplementary city, in return did not contain level or unit. Where before we had a data set that contained 11 times more unit fields than supplementary city fields, after doing the oversampling to get more supplementary city fields, the ratio flipped and we instead had 10 times more supplementary city fields than unit fields. This, along with the tokenization issue mentioned previously is not unlikely to have contributed to our poor results in these classes. It would be interesting to do another experiment where the data set is more balanced, and there is an equal distribution of samples.

7.1.4 Different data set

At the start of the project, we considered several different countries to work on, including Sweden, Norway and France. Had time permitted, we would likely have done some experiments on French addresses. The reason for choosing the Danish data was partly due to its similarity to the Swedish address format that we are already quite familiar with. It would be interesting to see if and how context improves performance for addresses in other countries that might have a different structure than Danish addresses.

7.1.5 More data

If we had had access to more powerful hardware, or if time had not been a constraint so that the code could have been optimized, more data could have been used for training. While throwing more data at the model is not always the solution, it would be interesting to see what difference it would make.

References

Akbik, A., Bergmann, T., Blythe, D., Rasul, K., Schweter, S., & Vollgraf, R. (2019). Flair: An easy-to-use framework for state-of-the-art NLP. *NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations)*, 54–59.

Akbik, A., Blythe, D., & Vollgraf, R. (2018). Contextual string embeddings for sequence labeling. *Proceedings of the 27th international conference on computational linguistics*, 1638–1649.

Barrentine, A. (2016). Statistical NLP on OpenStreetMap [Blog post]. https://medium.com/@albarrentine/statistical-nlp-on-openstreetmap-b9d573e6cc86

Chang, K.-T. (2008). *Introduction to geographic information systems* (Vol. 4). McGraw-Hill Boston.

Chollet, F. (2021). *Deep learning with Python* (Second edition). Manning Publications.

Cord, M., & Cunningham, P. (Eds.). (2008). *Machine learning techniques for multimedia: Case studies on organization and retrieval ; with 20 tables*. Springer.

Craig, H., Yankov, D., Wang, R., Berkhin, P., & Wu, W. (2019). Scaling Address Parsing Sequence Models through Active Learning. *Proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 424–427. https://doi.org/10.1145/3347146.3359070

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of Deep Bidirectional Transformers for Language Understanding.

Du, J. (2019). The Frontier of SGD and Its Variants in Machine Learning. *Journal of Physics: Conference Series*, *1229*, 012046.

Ghosh, B., Dutta, I., Carlson, A., Totaro, M., & Bayoumi, M. (2020). An Empirical Analysis of Generative Adversarial Network Training Times with Varying Batch Sizes. *2020 11th IEEE Annual Ubiquitous Computing, Electronics Mobile Communication Conference (UEMCON)*, 0643–0648. https://doi.org/10.1109/UEMCON51285.2020.9298092

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT press.

Güting, R. H. (1994). An introduction to spatial database systems. *the VLDB Journal*, *3*(4), 357–399.

Haykin, S. (2004). A comprehensive foundation. *Neural networks*, *2*(2004), 41.

Heinzerling, B., & Strube, M. (2018). BPEmb: Tokenization-free Pre-trained Subword Embeddings in 275 Languages. *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. https://aclanthology.org/L18-1473

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, *9*(8), 1735–1780.

Hsu, L. S., & Obe, R. (2021). *Postgis in action*. Simon; Schuster.

Jabbar, H. K., & Khan, R. Z. (2014). Methods to Avoid Over-Fitting and Under-Fitting in Supervised Machine Learning (Comparative Study). *Computer Science, Communication and Instrumentation Devices*, 163–172. https://doi.org/10.3850/978-981-09-5247-1_017

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. *Nature*, *521*(7553), 436–444. https://doi.org/10.1038/nature14539

Leijnen, S., & Veen, F. v. (2020). The neural network zoo. *Multidisciplinary Digital Publishing Institute Proceedings*, *47*(1), 9.

Li, X., Kardes, H., Wang, X., & Sun, A. (2014). Hmm-based Address Parsing: Efficiently Parsing Billions of Addresses on MapReduce. *Proceedings of the 22nd ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, 433–436. https://doi.org/10.1145/2666310.2666471

Manning, C., Raghavan, P., & Schütze, H. (2010). Introduction to information retrieval. *Natural Language Engineering*, *16*(1), 100–103.

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

Nakayama, H. (2018). seqeval: A python framework for sequence labeling evaluation. https://github.com/chakki-works/seqeval

Nugues, P. M. (2014). *Language Processing with Perl and Prolog*. Springer Berlin Heidelberg. Retrieved February 20, 2022, from http://link.springer.com/10.1007/978-3-642-41464-0

Opitz, D., & Maclin, R. (1999). Popular Ensemble Methods: An Empirical Study. *Journal of Artificial Intelligence Research*, *11*, 169–198. https://doi.org/10.1613/jair.614

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., … Chintala, S. (2019). Pytorch: An imperative style, high-performance deep learning library. *Advances in Neural Information Processing Systems*, *32*. https://proceedings.neurips.cc/paper/2019/file/bdbca288fee7f92f2bfa9f7012727740-Paper.pdf

Rhind, G. (2020). Global Sourcebook for International Data Management.

Samuel, A. L. (1959). Machine learning. *The Technology Review*, *62*(1), 42–45.

Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. *Neural Networks*, *61*, 85–117. https://doi.org/10.1016/j.neunet.2014.09.003

Schuster, M., & Nakajima, K. (2012). Japanese and korean voice search. *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, 5149–5152.

Sharma, S., Ratti, R., Arora, I., Solanki, A., & Bhatt, G. (2018). Automated Parsing of Geographical Addresses: A Multilayer Feedforward Neural Network Based Approach. *2018 IEEE 12th International Conference on Semantic Computing (ICSC)*, 123–130. https://doi.org/10.1109/ICSC.2018.00026

Taylor, W. L. (1953). "Cloze procedure": A New Tool for Measuring Readability. *Journalism Quarterly*, *30*(4), 415–433.

Tharwat, A. (2020). Classification assessment methods. *Applied Computing and Informatics*.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., & Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 5998–6008.

Viloria, A., Lezama, O. B. P., & Mercado-Caruzo, N. (2020). Unbalanced data processing using oversampling: Machine learning. *Procedia Computer Science*, *175*, 108–113.

Walber. (2014). *Precision and recall*. https://commons.wikimedia.org/w/index.php?curid=36926283

*

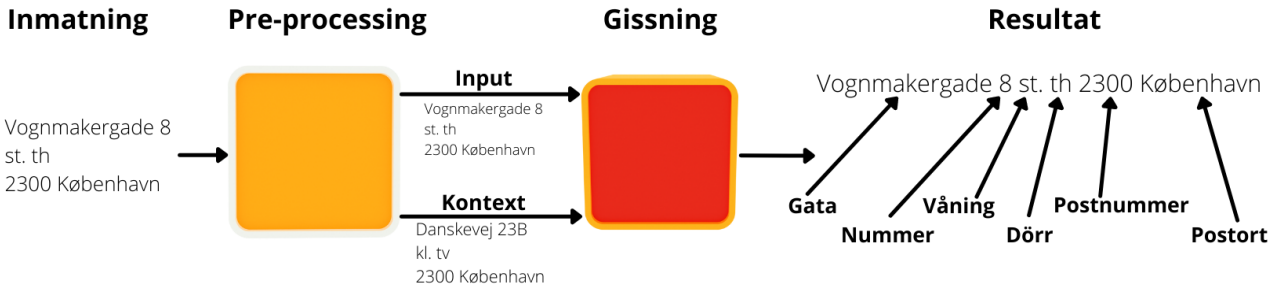
INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2022-03-17

EXAMENSARBETE Improving Address Sequence Tagger Using Geographical Context
STUDENTER Ludvig Eriksson, Mikael Olsson
HANDLEDARE Marcus Klang (LTH)
EXAMINATOR Martin Höst (LTH)

Förbättra en modell för sekvenstaggning med hjälp av geografisk kontext

POPULÄRVETENSKAPLIG SAMMANFATTNING **Ludvig Eriksson, Mikael Olsson**

Att korrekt kategorisera de olika beståndsdelarna i en geografisk adress är ett problem som flera stora företag kämpar med idag. I den här uppsatsen har vi undersökt möjligheten för att förbättra resultatet genom att använda närliggande adresser som geografiskt kontext.



Idag finns det massvis av olika kartverktyg såsom Google maps, Apple maps och även open-source varianten OpenStreetMap. För att alla dessa ska fungera korrekt krävs det att datans olika beståndsdelar taggas korrekt enligt exemplet ovan. För att slippa göra detta manuellt då det är väldigt tidskrävande har enorma forskningssatsningar gjort på hur man kan lösa detta med hjälp av maskininlärning. Vad det innebär är att du ger ett program en massa exempel på hur det ska se ut för att lära sig känna igen mönster och sedan kan det programmet få en adress inmatad och sedan (förhoppningsvis) korrekt klassificera de olika delarna. Detta kallas att träna en modell . Ett vanligt problem med maskininlärning är att datorn kan få svårigheter att förutse nya sorters uppgifter som skiljer sig från den data den har tränat på. I vårt examensarbete har vi undersökt möjligheten att förbättra modeller genom att mata in närliggande adresser. Vi använder en så kallad 'spatial database' vilket är en databas som är gjord för att hantera geografiska punkter och på så sätt kan man snabbt hitta närliggande adresser. När en användare matar in en adress hämtar vi den närmsta adressen och matar in båda dessa till vår modell. Modellen använder då den extra adressen för att kunna göra en mer kvalificerad gissning på indatan. Detta leder till ett förbättrat resultat på flera individuella fält och över hela adressen men då det krävs en sökning i en databas så tar det längre tid vilket leder till färre antal gissningar per tidsenhet.