University of Bergen
Department of informatics

Binary domain classification for Norwegian language in task-oriented dialogue systems

Author: Adrian Tvilde Evensen
Supervisor: Nello Blaser
Co-supervisor: Alex Simpson

UNIVERSITETET I BERGEN
Det matematisk-naturvitenskapelige fakultet

August, 2022

Abstract

Dialogue systems have gained more attention in recent years and have been called "the new app". This is much due to the advancement in *deep learning*, more precisely in *Natural Language Processing* (NLP). An additional factor to the growing popularity of dialogue systems has also been the enabling of integration of *task-oriented dialogue systems* with social media platforms.

The original purpose of this thesis was to take the first steps in developing such a task-oriented dialogue system. One crucial component in a task-oriented dialogue system is the *Natural Language Understanding* (NLU) component. The NLU aims at capturing a semantic representation of a user's utterance. It achieves this by classifying the domain and intent of the utterance, in addition to extracting potential slots in the utterance. Our focus for the thesis revolved around the domain and intent classification of the NLU component.

We were given a collection of utterances conveyed to a driving school via their social media account. Due to the condition of the dataset we received, we simplified the domain and intent classification problem to a binary domain classification. The binary classification task was to determine if an utterance should be handled by a human or the dialogue system. We trained a selection of binary classification models, combining different sentence representations with different machine learning models. We explored the sentence representations *Bag-of-Word* (BoW), *Word2vec*, *Doc2vec* and embeddings created with *Bidirectional Encoder Representations from Transformers* (BERT), in combination with the machine learning algorithms *Logistic regression*, *Random forest*, *Feedforward Neural Network* (FFNN), *Recurrent Neural Network* (RNN) and *Long Short-Term Memory* (LSTM). The models were evaluated using accuracy. Given the poor result of the binary classification task, we did not proceed the development of the NLU component, but instead shifted our focus towards understanding the reasons behind this result.

We observed that increasing the complexity of the model gave better results for the binary classification problem, while changing the sentence representation had little impact, beside BERT's embeddings. The best performing model was an FFNN with BERT's classification token. However, none of the models showed any remarkable results. We concluded that the

main reason for this was the lack of data and the unsatisfactory quality of the data labeling. In addition to this, the utterances in the dataset were quite long and not narrowed down to specific intents, which made them harder to classify. In summary, we experienced that the data played a big part in holding back the machine learning model's performance. This shows the importance of both good quality data, and proper labeling in the development of a well-functioning dialogue system.

Acknowledgements

First, I would like to give a special thanks to my supervisor *Nello Blaser* and my wife *Mari Karoline Mellingen*. Both of them have helped push this thesis into something more than I could ever have achieved by myself.

Nello has provided me with close guidance and has been supportive, shown interest, and driven me forward throughout this thesis. I have truly appreciated his honesty, friendliness and how available he has been. I am forever grateful for his feedback and insight.

Mari has encouraged me, provided me with assistance and kept my sanity in check at home. She has lifted my spirits on multiple occasions and prevented me from taking shortcuts. This experience would not have been as pleasant if it was not for her.

Secondly, I would like to thank Alex Simpson and Marius Nesse at Funbit AS for presenting the case and giving me the opportunity to write this thesis. I am grateful for the support I have received from them and all of the colleagues at Funbit AS.

Lastly, I would like to thank my friends and family who have supported me this year. It has truly been appreciated.

Adrian Tvilde Evensen
31st of August, 2022

Table of content

List of Figures

List of Tables

Abbreviations

ANN	*Artificial Neural Network.*
BERT	*Bidirectional Encoder Representations from Transformers.*
BoW	*Bag-of-Word.*
CBOW	*continuous bag-of-words.*
ELMo	*Embeddings from Language Models.*
FFNN	*Feedforward Neural Network.*
GDPR	*General Data Protection Regulation.*
GPT	*Generative Pre-trained Transformer.*
LDA	*Latent dirichlet allocation.*
LSA	*Latent semantic analysis.*
LSI	*Latent semantic indexing.*
LSTM	*Long Short-Term Memory.*
MLP	*Multi Layer Percepton.*
MultiWoZ	*Multi-DomainWizard-of-Oz.*
NLP	*Natural Language Processing.*
NLU	*Natural Language Understanding.*
NTNU	*Norges teknisk-naturvitenskapelige universitet.*
OOV	*Out Of Vocabulary.*
ReLU	*Rectified Linear Unit.*
RNN	*Recurrent Neural Network.*
SGT	*The Schema-Guided Dialogue Dataset.*
SVD	*Singular value decomposition.*
SVM	*Support Vector Machines.*
t-SNE	*t-distributed Stochastic Neighbor Embedding.*
tf-idf	*term frequency-inverse document frequency.*
UMAP	*Uniform Manifold Approximation and Projection.*

Chapter 1

Introduction

In this chapter we will introduce the background and problem description of this thesis.

1.1 Background

Dialogue systems, commonly referred to as chatbots or conversational agents, are a type of software that are capable of communicating with human users via natural language, usually over the internet. Dialogue systems have gained popularity in the recent years due to the advances in fields of artificial intelligence like machine learning, deep neural networks and natural language processing [45, Ch1 p1]. While there has been done a lot of research on NLP for the English language, the NLP-research for other languages, like for instance Norwegian, has been more scarce.

Dialogue systems can be used in a range of applications, among them customer service. A dialogue system could for example automate customer inquiries, increasing availability, reducing response time and overall achieve higher customer satisfaction and coverage. This also relieves human resources and saves expenses [40, Ch1 p1, Ch8.2 p14] [8].

Many companies use social media platforms to promote their business and services, and to handle their customer care. Social media platforms have changed how customers interact with companies and service providers. Customers may now reach out via private messages

or posts, requesting information or services, or expressing their frustration or satisfaction with the company [45, Ch5.1 p388].

In 2016, social media platforms like Facebook allowed developers to integrate dialogue systems with their platforms. Since then there has been observed an increase in the use and research of dialogue systems [40, Ch2 p3]. In 2019, Facebook also announced that they are going to merge WhatsApp, Instagram and Facebook Messenger. This will allow communication across social media platforms, and may further increase the popularity of dialogue systems [39].

Dialogue systems have been proclaimed to be "the new apps", and are predicted to have a big impact on retail, healthcare and banking [45, Ch1 p1] [8]. The advancement in the field of artificial intelligence and the growing need for automated digital customer services via social media platforms, are the main reasons for the selected topics of this thesis.

1.2 Problem description

The problem description for this thesis was proposed by Funbit AS [13]. Funbit is a digital agency, specializing in digital visibility and marketing, and offers development of digital solutions. Funbit wants to develop and integrate a dialogue system for one of their customers, via their customer's Facebook account. The customer is a driving school, delivering related services. The driving school handles their customer service via their Facebook account. This is time consuming and drains human resources. Some inquiries are repetitive, and some of them could be handled simply by a visit to their website.

To use in the development of the dialogue system, Funbit provided a dataset of utterances collected via the driving school's account. This dataset was then also the basis for this thesis. The dataset had been anonymized and personal information censored. The utterances in the dataset express intents sent from customers to the driving school, asking about related services. The majority of utterances in the dataset were in Norwegian (bokmål and nynorsk) and the thesis mainly focuses on these. There were also some utterances in English, but these were removed early in the process.

The utterances in the dataset were authentic in comparison to utterances one would usually send well knowing one was using a dialogue system, where utterances would be short and specific. In contrast, the utterances in the dataset were long, messy and rarely specific.

The initial problem description for this thesis was to develop a dialogue system based on the given dataset. There are a lot of commercial platforms available to develop and integrate dialogue systems, for instance *Dialogueflow*[1] (Google), *wit.ai*[2] (Facebook) or *Watson*[3] (IBM). These platforms enable anyone, with or without programming or machine learning experience, to develop a dialogue system by defining intents/utterances corresponding with responses/actions. They are very convenient, but the machine learning part is already configured and hidden away from the developer. There are also available programming frameworks for developing dialogue systems. One worth mentioning is *Rasa*[4], which is an open-source framework for Python. Rasa provides a lot of freedom, but is still quite user friendly with default and recommended settings etc. Since this is a thesis in machine learning and the goal is to understand the underlying machine learning models, none of the mentioned conventional solutions were used. However, they were used as a reference throughout the thesis.

Given the scope and complexity of a full functional dialogue system, we narrowed down the problem description to developing and analysing the first component in a dialogue system, the *natural language understanding* component. The NLU component classifies which domain an utterance belongs to, what intent it carries and extracts potential slots within the utterance. This information is then used as a semantic representation of the utterance further down the dialogue system. Our original purpose was to develop and analyse a NLU component with domain and intent classification. Due to the faulty condition of our data, we simplified the problem to a binary domain classification task. The task of the binary classification model was to determine if the utterance should be handled by a human or the dialogue system. This was done to see if the binary domain classification task had any success before preceding with the complete domain and intent classification task. Unfortunately, the results were not particularly impressive, and we shifted our focus to understand why this was the case.

The goal of this thesis was to find out which sentence representation techniques, combined with different machine learning models, performed best at classifying the collection of utterances sent from customers to the driving school. We began with a baseline model were the utterances were represented with *Bag-of-words* combined with a *Logistic regression* model. We then increased the complexity of the sentence representations and machine learning algorithms. For sentence representation we tried *Word2vec*, *Doc2vec* and *BERT*, which was

[1] https://cloud.google.com/dialogflow/docs
[2] https://wit.ai/
[3] https://www.ibm.com/watson
[4] https://rasa.com/

pre-trained on Norwegian. We also explored *Random forrest*, *Feedforward neural network*, *Recurrent neural network* and *Long short-term memory* as machine learning algorithms. After observing the results, we tried to understand why the models did not perform as well as we initially thought.

Chapter 2

Machine learning

In this chapter we will explain some general concept within the field of machine learning and associate them with natural language processing. First will we take a look at what machine learning is, different categories of machine learning and good practices. Then we will look at some relevant machine learning algorithms.

2.1 What is machine learning?

Machine learning is the field of study that enables computers to learn from data rather than being explicitly programmed. *Machine learning algorithms* are used to train *machine learning models*. A model is trained by feeding it data, from which it fits the model's parameter, enabling the model to generalize to unseen data. For instance *linear regression* is an algorithm that fits a line that minimizes the distance from given data points to the line. Liner regression is mentioned later in this thesis as well, but will not be further elaborated. The model can then be used to make predictions or decisions on new data [58, Ch1]. There are both good and poor generalizations. George Box once said: *"All models are wrong, but some are useful"* [44, Ch2.3 p792].

Machine learning can be divided into four categories; *supervised*, *unsupervised*, *semi-supervised* and *reinforcement learning*. Each category corresponds with the type and amount of supervision given during training [58, Ch1 p8]. For the purpose of this thesis, we will explain supervised and unsupervised learning closer in the following subsections.

2.1.1 Supervised learning

In supervised learning both the input data as well as the desired output is fed to the machine learning algorithm during training. The input is usually called *features*, *predictors* or *independent variables*, and is denoted as X. The output is known as *target*, *classes/labels*, or *dependent variables*, and is denoted as y [58, Ch1 p9]. The input data usually consist of *numerical* or *categorical* features, or a mixture of both. However, most machine learning algorithms prefer numerical input [58, Ch2 p69], hence it is usually necessary to encode the categorical features into a vector of some form. *Word embedding*, which is a vector representation of a word, is such an example. We will further explain word embeddings in section 3.3.3.

A model is trained by fitting some parameters to the machine learning algorithm by minimizing a *cost function* for that specific problem. The cost function is a penalty measurement for the model, defining a target for the model given the labeled data [58, Ch1 p23]. One example of a cost function is the *log loss function*, which we will explain in section 2.2.1.

A model could predict either numerical or categorical output. This is respectively referred to as *regression* and *classification*. Regression is the task of predicting continuous/numeric values while classification is the task of predicting a class [58, Ch1 p9]. Classification tasks are involved in many NLP problems (section 4.3.1) and will now be further elaborated.

2.1.1.1 Classification

As mentioned, classification is the task of categorizing an instance as a class/label based on its features. An example of a classification task are spam filters. Spam filters classify incoming mail as either "Ham" or "Spam" [58, Ch1 p9]. NLP has many classification related tasks. For instance, classifying documents or utterances according to their content. *Sequence labeling* is the act of classifying a sequence of instances, for example the words in a sentence. Sequence labeling is therefore especially useful when extracting information preserved in text. This is done by performing *name entity recognition* [51, Ch8 p148], which we write more about in section 3.1.2.

2.1.1.2 Splitting the dataset into training-, validation- and testing sets

An important step when building machine learning models is splitting the dataset. It is common to keep a portion of the dataset untouched for the final evaluation of a model. By doing this we preserve an unbiased evaluation of the final model. This portion of the dataset is known as the *testing set*. The remaining portion of the dataset is used to train and tune the selection of models. A portion of the remaining *training set* is used for selecting the best model. This portion is called the *validation set* and is usually used for *hyperparameter tuning*. Hyperparameter tuning is the process of finding the optimal set of parameters for the machine learning algorithm, so that it can achieve the best possible model. It is common to use 80% of the data for training and validating, and the remaining 20% for testing. However, this is a question about the availability of data. If there is a lot of data available, the testing set may be reduced. Correspondingly, if the dataset is small one should consider increasing the amount of testing data [58, Ch1 p31-32].

2.1.1.3 Performance measures

Another important step when building machine learning models is to evaluate them for both fine-tuning the model and for a final measurement of performance [58, Ch1 p32]. *Accuracy* is a performance metric that tells you the percentage of correctly classified instances done by the model. However, accuracy is sometimes a naive approach when evaluating a classifier. Imagine your spam filter scores an accuracy of 99%, but then it turns out that 99% of your incoming emails are actually "Ham". By only classifying your mail as "Ham", the classifier would get a high but misleading performance score. Therefore, accuracy is not a good measurement of performance if the data is skewed or unbalanced, meaning some classes are much more frequent than others [58, Ch3 p92].

Confusion matrix provide much more insight when evaluating classifiers. This method presents a matrix that show how many instances where correctly classified, and how many were misclassified (see Figure 2.1). Correctly classified instances of a class are referred to as *true positives* (TP). Correctly classified instances of a separate class are *true negatives* (NP). Instances classified as "Ham", but were in reality "Spam" are known as *false negatives* (FN). Lastly, instances classified as "Spam", but that were actually "Ham" are called *false positives* (FP) [58, Ch3 p92].

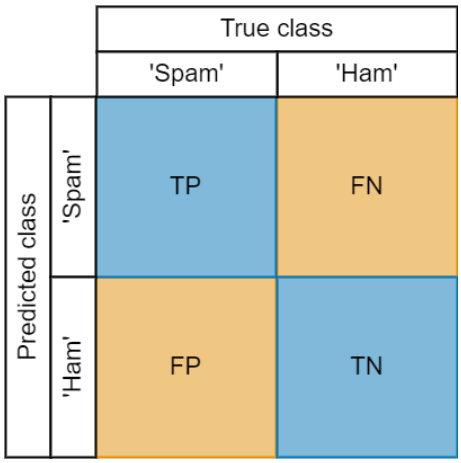

Figure 2.1: Example of an confusion matrix summarizing the prediction of "Spam" and "Ham".

The confusion matrix gives us an overview over what classes a classifier struggles to classify. From here, concrete measurements can be taken towards a better performance. If accuracy does not cover the evaluation of a problem, more complex metrics can be calculated from the confusion matrix, *precision* and *recall*. Precision can be calculated as the following

$$Precision = \frac{TP}{TP + FP}, \tag{2.1}$$

and estimates what portion of the positive identifications were actually correct. Recall is another metric

$$Recall = \frac{TP}{TP + FN}. \tag{2.2}$$

Recall tells us what portion of actual positives were identified correctly. It is common to combine these metrics to one unified metric, *F1 score*. F1 score takes both aspects into account as the following

$$F1\ score = 2 \times \frac{Precision \times Recall}{Precision + Recall}. \tag{2.3}$$

F1 score ranges between 0 to 1, where a score closer to 1 indicates the better model [58, Ch3 p95].

2.1.1.4 Overfitting and underfitting

One of the main challenges when training machine learning models is *underfitting* and *overfitting* [58, Ch4 p136]. This is also known as the *bias-variance tradeoff* (see Figure 2.2). Overfitting refers to the situation when a model is too complex - fitting the model too closely to the training data and fails to generalize to unseen data. Underfitting happens when a model is too simple and therefore fails to model the training data and generalize to new data [58, Ch1 p28-30].

Bias-variance tradeoff expresses the error made by machine learning models as the sum of three different types of errors; *bias*, *variance* and *irreducible errors*. Bias is errors due to wrong assumptions in the learning algorithm. A high-bias model is likely to underfit the data, for instance using linear regression when the data is non-linear. Variance refers to how sensitive a model is to variation in the data. A high-variance model is therefore likely to overfit the data. Lastly, irreducible error is caused by the general noise in the data [58, Ch4 p136].

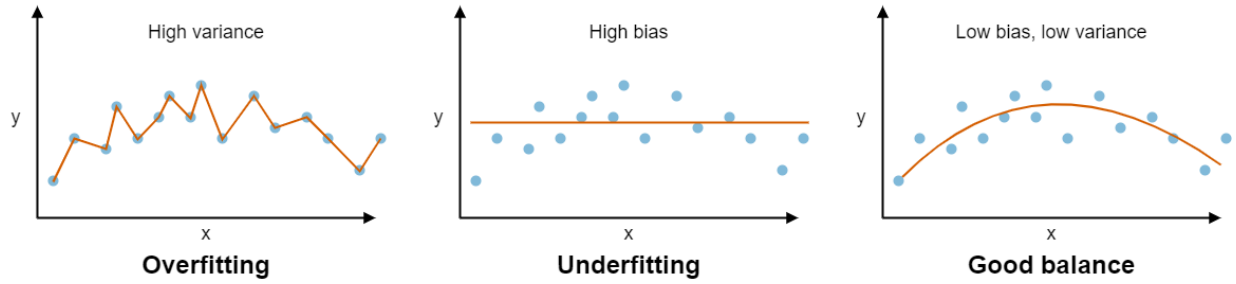

Figure 2.2: Illustration of bias-variance tradeoff.

2.1.2 Unsupervised learning

Unsupervised learning is machine learning techniques that aim to learn patterns from the data without human supervision (no labels). Some applications of unsupervised learning are for example *clustering* and *dimensionality reduction* [58, Ch1 p10]. Clustering is the task of grouping similar instances together [58, Ch9 p238]. Dimensionality reduction aims at simplifying the data by reducing its feature dimensions, usually by creating and selecting new ones [58, Ch1 p12]. This is useful for speeding up training of machine learning models and for visualization of the data [58, Ch8 p216]. *Uniform Manifold Approximation and Projection* (UMAP) and *t-distributed Stochastic Neighbor Embedding* (t-SNE) are both

dimension reduction techniques used for visualizing high-dimensional data. We are going to look more into t-SNE in section 2.4.1, and UMAP in 2.4.2. Another application of unsupervised learning, and quite useful in NLP, is *representation learning*. Representation learning are methods that automatically learn features from the data [51, Ch5.1.1 p81]. This is done through *self-supervised learning*. Self-supervised learning generates labels from the available data, which it then uses for training [51, Ch6.8 p113]. Word2vec and other word embedding methods utilize this technique [63, Ch1.5 p8]. Word2vec and other embedding techniques will be explained in section 3.3.

2.2 Machine learning algorithms

This chapter will explain different machine learning algorithms that are relevant for this thesis. We will begin with *logistic regression*, followed by *random forrest* and lastly *Artificial Neural Network* (ANN), which lay the foundation for the following section about Neural network architectures.

2.2.1 Logistic regression

Logistic regression is a common supervised learning technique used for binary classification, i.e distinguishing data into two different classes. It estimates the probability of an instance belonging to a particular class [58, Ch4 p144]. Logistic regression can also be generalized for multiple classification. This is known as *multinomial logistic regression*, or *softmax regression* [58, Ch4 p149].

Logistic regression works very similar to linear regression. It computes the weighted sum of the input features, but then applies the *logistic function*, which is done as follows:

$$p(\theta X^T) = \frac{1}{1 + e^{-\theta X^T}}. \tag{2.4}$$

The logistic function outputs a value between 0-1, representing the probability of a class. The probability is computed via the logistic function $p(\theta X^T)$ given the model parameters θ and the transpose input features X^T. Logistic regression aims at finding the correct parameter

that minimizes the cost function $J(\theta)$. Logistic regression uses the *log loss function*, also known as *cross-entropy* [58, Ch4 p146]

$$J(\theta) = -\frac{1}{m}\sum_{i=1}^{m}[y^i log(\hat{p}^{(i)}) + (1 - y^i)log(1 - \hat{p}^{(i)})]. \tag{2.5}$$

In log loss, m denotes the number of instances, y^i denotes correct class of that instance (0 or 1), while $p^{(i)}$ is the probability given by the logistic function.

Log loss is practical when we optimize the cost function according to probability [58, Ch4 p150]. Logistic regression optimizes the model parameters by utilizing optimization algorithms, usually by *gradient descent* which we will explain in section 2.2.3 [58, Ch4 p146].

2.2.2 Random forest

Random forest is an *ensemble learning method* used to train classification models. An ensemble learning method combines multiple models into one model. The ensemble model makes prediction by taking the average or the majority of prediction from the collection of models [58, Ch7 p191]. Random forest ensembles multiple models of *decision trees*, which will be explained next.

Decision trees are models structured with nodes and edges, resembling trees. Each node has a decision boundary that defines which edge to follow. Decision trees make predictions, starting at the *root node* then proceeds to move down the tree according to decision boundaries. This continues until it finds a *leaf node*. A leaf node is at the end of a tree representing the decision made by the tree, see Figure 2.3. The decision boundaries are defined with a threshold value for a feature, for example; *petal length (cm)* $\leq$ *2.45* [58, Ch6]. We will not elaborate on the algorithms that create the decision boundaries, but rather give the essence of what they aim at doing. The algorithm recursively splits the dataset according to one of its feature and its threshold which produces the "purest" subsets. Purity refers to the portion of similar classes within each subset. The algorithm continues to split the subset until some stopping criteria is met. Some of these stopping criteria are *maximum depth* and *minimum samples per leaf* [58, Ch6 p182].

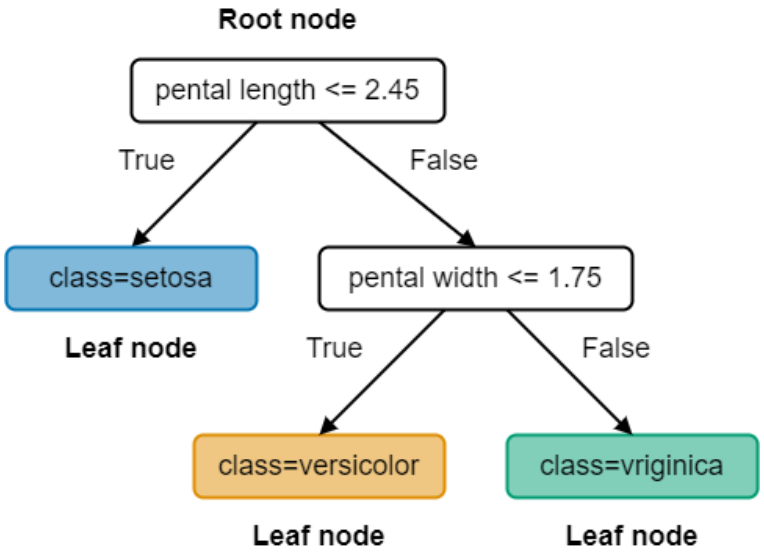

Figure 2.3: Figure of a decision tree's decision boundaries and leaf nodes, and is based on [58, Figure 6.1]

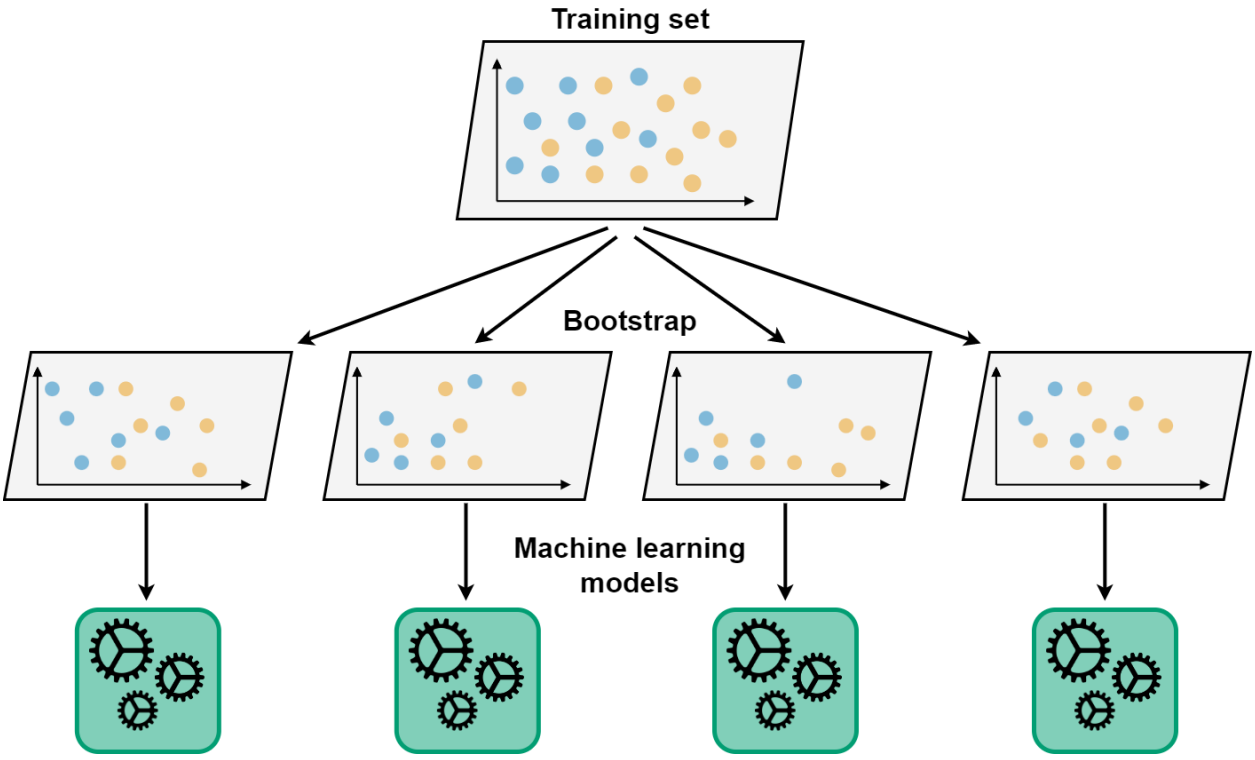

Figure 2.4: Figure of a bagging. Based on [58, Figure 7.4]

Random forest creates a number of decision trees which it ensembles. To obtain diversity

between the trees and avoid training the same decision tree over and over again, the random forest utilize a method called *bagging*. Bagging is also known as *aggregated bootstrapping*, which is random sampling with replacement, meaning some instances might be sampled multiple times, while others not at all [58, Ch7 p195]. Figure 2.4 illustrates bagging, where different models are created of different bootstrapped datasets. Additionally, random forest apply random sampling of the features, only taking into account a portion of the available features in the dataset at each split. The portion is set via a hyperparameter called *maximum features*.

2.2.3 Artificial neural networks

Artificial neural networks, or just neural networks, are computational systems that where inspired by the biological brain [57, Ch1 p13]. ANNs are networks of *artificial neurons* connected by edges. The edges are associated with weights, which are the neural networks parameters. The artificial neurons take the weighted sum of the input edges and applies an *activation function* to it [57, Ch1] [58, Ch10]. The activation function introduces non-linearity to the network, which is necessary for the network to learn complex relationships [47, Ch1 p11-12]. The most common and recommended activation function is the *Rectified Linear Unit* (ReLU)[57, Ch6.1 p171]. ReLU outputs the weighted sum z, if z is higher than zero. If it is less than zero, ReLU outputs zero; $\text{ReLU}(z) = max(0, z)$.

The arrangement of neurons and connections of edges may be structured in many different ways, creating complex architectures of ANNs. One of the simplest ANN is the *perceptron*, see Figure 2.5.

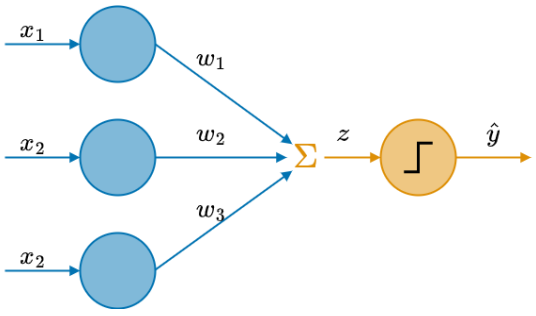

Figure 2.5: Figure of a perception with three inputs.

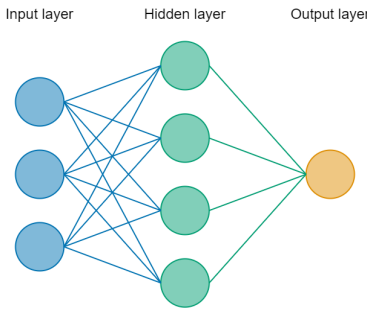

Figure 2.6: Figure of a Multi-layer perceptron.

The perceptron takes the weighted sum z of inputs and passes it through the *threshold logic unit* (TLU). The TLU takes the weighted sum and applies the *step function* as the activation function, to compute the output $\hat{y}$ [58, Ch10 p281-283]

$$\text{step function}(x) = \begin{cases} 0 & x < 0 \\ 1 & x \geq 0 \end{cases}.$$

If we stack multiple perceptrons together, we get what is called a *Multi Layer Percepton* (MLP) [58, Ch10 p285]. MLPs have one input layer, one or more hidden layers and one output layer[58, Ch10 p286], see Figure 2.6. Increasing the number of hidden layers in the network, increases the depth of the network. This concept is what *deep learning* refers to [57, Ch6 p165]. Deeper networks are able to learn more complex non-linear functions from the data [42, p60].

MLP uses *gradient decent* (see section 2.2.3.1) to train the weights of the network [58, Ch10 p286]. Gradient decent is the most common method to train neural networks [57, Ch1 p17], and is used to "tweak" the model's parameter in the right direction.

2.2.3.1 Gradient decent

Gradient descent is an optimization algorithm, which finds the optimal parameters for a model by "tweaking" them in the right direction iteratively [58, Ch4 p119]. To implement gradient decent we need to compute the gradient of all the parameters based on the error given by the cost function, $\nabla J(\theta)$ [58, Ch4 p123]. Once we have the gradients we nudge the

parameter in the right direction, and this minimizes the cost function. Gradient decent can be applied in three different ways; *batch-*, *stochastic-* and *mini-batch gradient decent*.

Batch gradient decent runs through the entire training set, before correcting the parameters. This method is therefore terribly slow when the training sets are very large [58, Ch4 p123-125].

Stochastic gradient decent only takes one random instance from the dataset at each step. It then computes the gradients based on the error produced by this one instance and change the model parameters accordingly. This makes the algorithm converge much faster. However, since it is random and only considers one instance is the cost function is very irregular and bounces up and down. In other words, it will not converge in the same way as batch gradient decent [58, Ch4 p126-128].

Mini-batch gradient decent is between the two extremes we have just explained. It randomizes and splits the data into smaller portions called batches. For each batch are the gradients calculated and used to update the models parameters. Mini-batch gradient decent is therefore both faster than batch and more stable than stochastic [58, Ch4 p129].

To efficiently compute all of the gradients and update every single parameter in a model, is the *backpropagation algorithm* used. The backpropagation algorithm is a widely used method for training neural networks [58, Ch10 p287]. We will explain the backpropagation algorithm in the next section.

2.2.3.2 Backpropagation

Backpropagation utilizes gradient decent to fit a model's parameters according to the training data. It works in two "steps". The first step is known as the *forward pass*, where each instance in the mini-batch is passed through the network and predicted an output for. In addition to this, all intermediate values are computed throughout the network and preserved for later. The second step is called the *backward pass*. Here, the gradient of each parameter is computed based on the error given by the cost function. It then uses the *chain rule* ($F'(g(x)) = f'(g(x))g'(x)$) and the preserved values to efficiently compute the gradient for each parameter throughout the network [58, Ch10 p287-288].

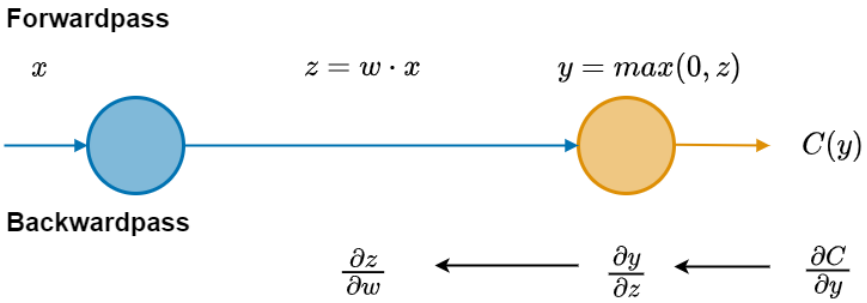

Figure 2.7: Figure of backpropagation for a simple perceptron. The perceptron consisting of one input (x), one weighted edge (w) and one output (y). During the forward pass the perceptron computes z, the product of w and x, which it applies the ReLU activation function to and outputs y. The backward pass computes the partial derivatives of the cost function C, the ReLU function y and the product z.

We will demonstrate backpropagation for a simple perceptron (see Figure 2.7) with a ReLU activation function.

We use gradient decent to adjust the parameter w according to gradient ∇w calculated from the cost function C and the *learning rate* α, as follow

$$w_{new} = w_{old} - \alpha \cdot \nabla w. \tag{2.6}$$

The gradient expression can be expanded using the chain rule like this

$$\nabla w = \frac{\partial C}{\partial w} = \frac{\partial C}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w}. \tag{2.7}$$

We will use the *mean square error* as the cost function C for this example. The variables we are going to use are: $x = 2.0$, $w = 2.0$, $\hat{y} = 1.0$ and $\alpha = 0.1$. The first step of backpropagation is the forward pass

$$
\begin{aligned}
z &= w \cdot x = 2.0 \cdot (2.0) = 4.0, \\
y &= max(0, z) = 4.0, \\
C &= \frac{1}{2}(y - \hat{y})^2 = \frac{1}{2}(4.0 - 1.0)^2 = 4.5.
\end{aligned}
$$

After the forward pass we preform the backward pass using the computed value y to compute the gradient, as follow

$$
\begin{aligned}
\frac{\partial C}{\partial y} &= (y - \hat{y})(1) = (4.0 - 1.0) = 3.0, \\
\frac{\partial y}{\partial z} &= \begin{cases} z < 0, & 0 \\ z > 0, & 1 \end{cases} = 1, \\
\frac{\partial z}{\partial w} &= x = 2.0.
\end{aligned}
$$

Once we have all the partial derivatives associated with ∇w following equation 2.7 is the gradient easy to compute

$$
\frac{\partial C}{\partial y}\frac{\partial y}{\partial z}\frac{\partial z}{\partial w} = 3.0 \cdot 1.0 \cdot 2.0 = 6.0.
$$

With the gradient we can preforme the gradient step using equation 2.6

$$
w_{new} = 2.0 - 0.1 \cdot 6.0 = 1.4.
$$

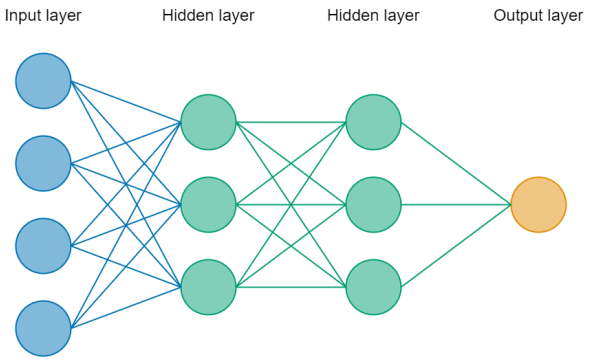

Figure 2.8: Figure of a feedforward neural network with two hidden layers.

Once w have been updated, we can preform a second forward pass to shows that the cost function have been reduced from 4.5 to 2.42. The perception is then closer at estimating the actual value.

2.3 Neural network architectures

In this section will we look closer at some of the architecture used in deep learning. First we will explain the simplest architecture, called *feedforward neural network*, followed by *recurrent neural network* which covers relevant concepts for NLP. Then we will look at the *long short-term memory* architecture, and lastly will we explain the *transformer* architecture, which made a big impact on the field when it was introduced.

2.3.1 Feedforward neural network

Feedforward neural networks are the same as MLPs, which we have already explained in section 2.2.3. FFNNs are pretty straight forward, as the network consists of one input layer, one or more hidden layers and one output layer, see Figure 2.8. It is called "feedforward" because the information flows in an forward motion; from the input layer, through the intermediate computations in the hidden layers and then finally through the output layer [57, Ch6 p1].

A FFNN with at least one hidden layer using any non-linear activation function and with enough hidden units, will be able to approximate any continuous function mapping from one dimensional space to another [57, Ch6.4.1 p194].

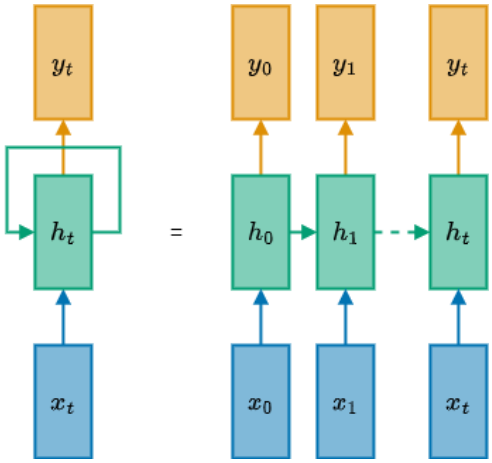

Figure 2.9: Unrolled recurrent neural network. The recurrent neural network takes an input x_t, passes it through the hidden layer h_t and outputs a value y_t. The recurrent connection allows information to flow from one time step, to another. This figure is inspired by a figure from Christopher Olah's blog [33].

2.3.2 Recurrent neural networks

The term recurrent neural networks refer to any type of network that have a cycle within its network. This meaning that at least one unit within the network is dependent directly or indirectly on its previous state. The previous hidden state is passed on till the next hidden state for each time step t, see Figure 2.9. RNNs are hard to interpret and difficult to train because of this relation. However, they have been proven to be extremely effective with language [52, Ch9.2 p186]. The cycle, known as the recursive link, work as a memory for the network. The recursive link is the main difference between RNNs and FFNNs.

One problem with basic FFNNs is that they require a fixed input size and produces a fixed output size (one-to-one relationship). This is not very convenient when processing, or generating text, and the same can be said for any type of sequential data. As mentioned, the recursive link in an RNN works as a memory. This enables the network to remember sequential data, and overcomes the limitations of fixed lengths [52, Ch9.2 p187]. RNNs therefore enable a framework where more complex architectures are possible; many-to-one, one-to-many or many-to-many [35] [57, Ch10.2 p372], see Figure 2.10.

RNNs are convenient networks for a range of NLP tasks. The many-to-one design is useful for *sequence classification* and for *language models*, see more in section 3.1.1. If we combine the many-to-one design with the one-to-many design, we get an *encoder-decoder* network.

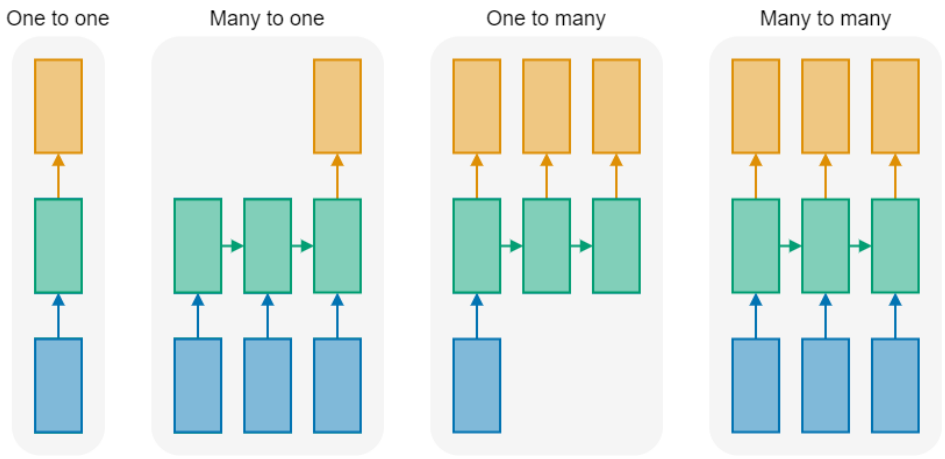

Figure 2.10: Overview of some designs possible with recurrent neural networks. The rectangles represent vectors while the arrows are functions. Blue rectangles are input vectors, green are hidden state vectors, and the orange are output vectors. **(1)** One to one illustrates feedforward neural networks with a fixed input and output size. **(2)** Many to one is a useful design for sequence classification as it takes a sequence in and outputs a label. **(3)** One to many is suitable for generating image caption since it can take an image as input and output a image caption text. **(4)** The many to many design takes a sequence as input and also outputs a sequence and is therefore useful for sequence labeling. The figure is inspired by a figure from Computer Science Ph.D Andrej Karpathy's blog [35].

Encoder-decoder network is useful for NLP tasks like *question answering* and *machine translation* [52, Ch10.3 p219]. The input, for instances a sentence in a given language, is encoded into a contextualized representation, and then decoded into the same sentence in another language. The many-to-many design is also suitable for *sequence labeling*, where each item in a sequence is given a labeled. We will explain more about sequence labeling in section 3.1.2.

A plain RNN will quickly run into some challenges. The first challenge is learning backwards in time. Due to the nature of the recurrent connections the gradients backwards in time will gradually vanish. Meaning the network will take very long, or fails at learning what information is important to carrying forth or not [59, Ch1 p1]. This is known as the *vanishing gradients* problem [52, Ch9.6 p198]. Because of the vanishing gradients, it is quite challenging to perform backpropagation backwards in time. The gradients become smaller and smaller further back in time, until they eventually are driven towards zero [52, Ch9.6 p198]. When the gradients are zero, the network does not learn. Lastly, due to the nature of RNNs, they are very slow to train. This is because they operate on sequential data which is hard to compute in parallel [52, Ch9.6 p200]. More complex architectures have been designed to overcome these problems, among them are the long short-term memory, which we well look at in the next section, and the transformers which we will look at in section 2.3.4.

2.3.3 Long short-term memory

Long short-term memory is a recurrent network architecture designed to overcome the vanishing gradients problem which occurs in regular RNNs [59, p1] [52, Ch9.6 p198].

LSTM introduce *memory cells* and *gate units*, which it utilizes alongside the hidden states. The memory cells, sometimes referred to as *context*, is a vector where information may be written to or read from. This information flow is regulated by the gate units [59, Ch4 p6] [52, Ch9.6 p198]. These are the following gate units; the *forget gate*, the *input gate* and the *output gate*, see Figure 2.11.

The first gate unit is the forget gate. The forget gate concatenates the previous hidden state h_{t-1} and the input x_t and runs it through a sigmoid layer. The forget gate creates a vector f_t with values between 0 and 1 which it multiplies with the memory cells C_t. Values in f_t indicates how much of each cell it should forget. The second gate is the input gate. It

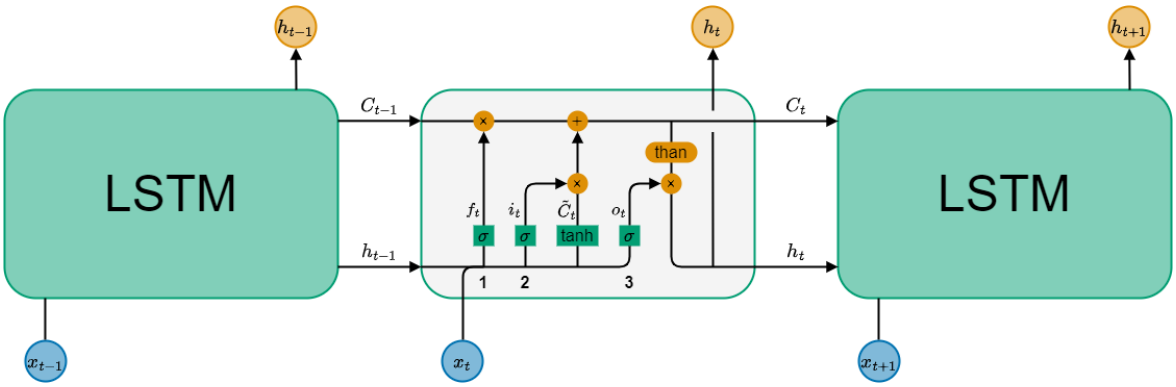

Figure 2.11: Figure illustrating the logic happening inside an LSTM block. The green rectangles represents neural networks layer with the given activation function, while the orange circles are pointwise operations. **(1)** The forget gate. **(2)** The input gate. **(3)** The output gate. The figure is inspired by a figure from Christopher Olah's blog [33]

computes what information to add to the memory cells. It does this by computing candidates i_t using a sigmoid layer, which is then multiplied with the information $\tilde{C}_t$ and then added to C_t, updating the memory cell. $\tilde{C}_t$ is a layer with tahn activation function. Both i_t and $\tilde{C}_t$ is computed from h_{t-1}, x_t and h_t. The last gate unit is the output gate. The output gate decides what to output (h_t) by computing candidates o_t from x_t and h_{t-1}, which it then multiplies with $tanh(C_t)$. $tanh(C_t)$ scales the value of the memory cells C_t to be between -1 and 1 [33] [52, Ch9.6 p198] [57, Ch10.10.1 p404].

The recurrent connection of memory cells become a loop known as the *Constant Error Carousel* (see Figure 2.12). The constant error carousel overcomes the vanishing gradients problem by keeping a constant error flow equal to 1.0 throughout each time step with the identity function, enabling the gradient to flow directly through the network [59, Ch3.2 p5] [70, Ch8.1 p19] [57, C10.10.1 p405].

2.3.4 Transformers

Transformers are neural networks with an encoder-decoder architecture. They are based around the problem with recurrent networks and the success of attention mechanism in encode-decoder models [73, p1]. Transformers are designed to handle sequential data like RNN, but are not recurrent. Since they are not recurrent, they instead use *positional encoding* to keep track of the sequential order (we will not elaborate on positional encoding) [73, Ch3.5] [52, Ch9 p185]. Avoiding recurrent computations enables the transformer to be parallelized,

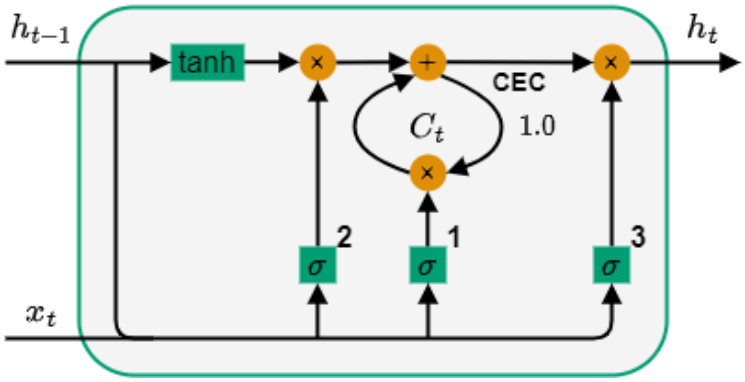

Figure 2.12: Figure of a LSTM block, illustrating the *Constant Error Carousel* (CEC) which is a loop of the memory cells allowing the gradients to flow directly through the network overcoming vanishing and exploding gradients. **(1)** The forget gate. **(2)** The input gate. **(3)** The output gate. The figure is inspired by [57, Figure 10.16]

significantly reducing the time and resources they demand during training and usage [73, Ch1 p2].

Attention is the ability to compare an item of interest to other items within the same sequence. The simplest form of attention is the dot-product between two items represented as vectors (x_i and x_j). The dot-product is a scalar value indicating the similarity of the items. The larger the value, the more similar the items are [52, Ch9.7 p201]. The more similar an item is, the more attention it is given. Attention is used to reinforce or reduce the information of an item taken into account when performing computations. We will demonstrate this with a very simple equation based on the equations in [52, Ch9.7 p201];

$$y_i = \sum_{j=1}^{n} \text{dot}(x_i, x_j) x_j \tag{2.8}$$

The output y_i is computed as the sum of attention $\text{dot}(x_i, x_j)$ multiplied with x_j for all n elements. Figure 2.13 illustrates a simple visualization of attention.

The transformer uses an attention mechanism called *self-attention*. Self-attention helps relating different positions over long distances of the same input sequence to compute a sequence representation [50] [73, Ch2 p2]. To compute the attention, the transformer uses *scaled dot-product attention*. It is similar to what we just explained (equation 2.8), but includes scaling (d_k), normalization via softmax and some additional features. In scaled dot-product

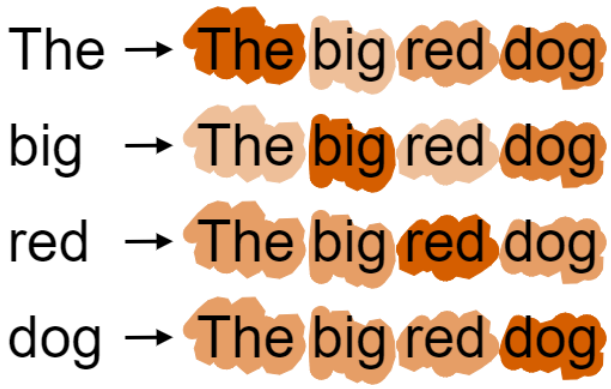

Figure 2.13: Illustration of how attention weights different words in a sentence given the word of interest.

attention, items are mapped via trainable networks into three different roles that the items play in attention; *query*, *key* and *value*. The items then play the different roles depending on what item is the one of interest at any given moment in the process. Query is the representation of an item of interest, while Key is the item representation being compared with the query. The dot product of the query and keys is then computed and normalized, as the attention. Finally, Value is the representation of the actual value of an item being mapped with the attention [73, Ch3.2.1]. The queries are kept in matrix Q, keys in K and values in V. The attention function is defined as:

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V \tag{2.9}$$

Transformers take the scaled dot-product attention a step further by stacking several layers of them together. Each layer then learns different relationships between items. The stacked layers are called a *multi-head attention* layer [73, Ch3.2.2 p4] [52, Ch9.7.2 p206], see Figure 2.14.

The multi-head attention layer plays a major role in both the encoder and decoder part of the transformer [73, Ch3]. The encoder consists of N identical layers, composed of two layers. First layer is a multi-head self-attention mechanism, followed by a FFNN. Both layers have a residual connection with a normalization layer. The decoder is also a stack of N (see Figure 2.15) identical layers. The decoder have the same layers as the encoder, but have also a third layer. The third layer is an multi-head attention over the output from the encoder. Each layer in the decoder have an residual connection followed by a normalization layer [73, Ch3.1 p3].

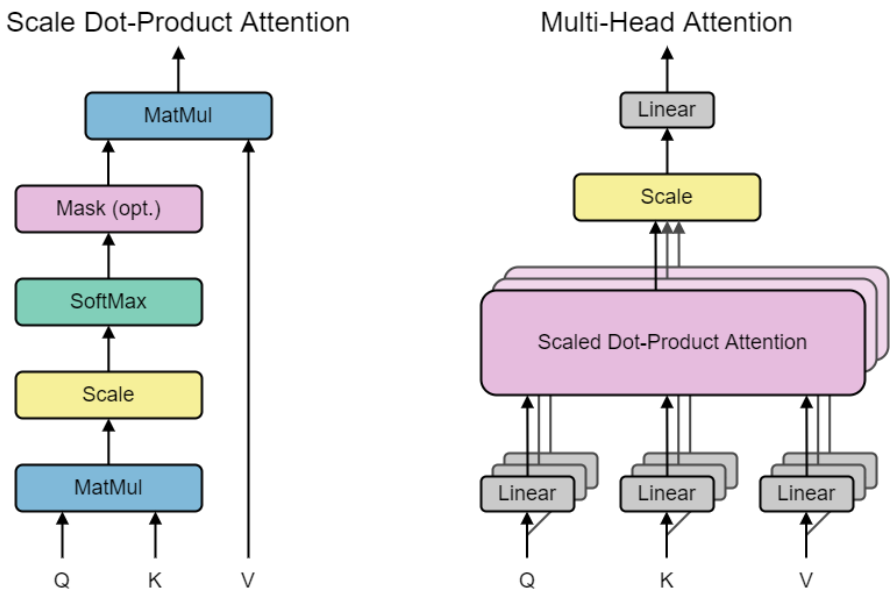

Figure 2.14: Figure of the architecture for the Scale Dot-Product Attention (left) and the Multi-Head Attention (right). This figure is based on [73, Figure 2].

2.4 Dimensionality reduction methods

As mentioned in section 2.1.2, dimensionality reduction is considered an unsupervised learning task. It is usually used for compression, feature extracting or visualization. This is done by reducing the dimensions of the data [58, Ch1 p12] [58, Ch8 p216]. In this section we will explain two useful visualization techniques; *t-distributed Stochastic Neighbor Embedding* and *Uniform Manifold Approximation and Projection*. Both t-SNE and UMAP are non-linear dimension reduction techniques. However, the interpretability of the data is lost when applying non-linear reduction techniques, meaning the new dimensions in lower space have no specific meaning [64, Ch6 p45].

2.4.1 t-SNE

t-SNE aims at keeping similar instances close and dissimilar instances apart when performing dimensionality reduction [58, Ch8 p235]. t-SNE computes the conditional probability of all the neighboring points under a Gaussian distribution in high-dimension, and then tries to compute a similar conditional probability distribution for the same neighbors in lower dimensional space. The points are initially mapped randomly onto lower dimension space.

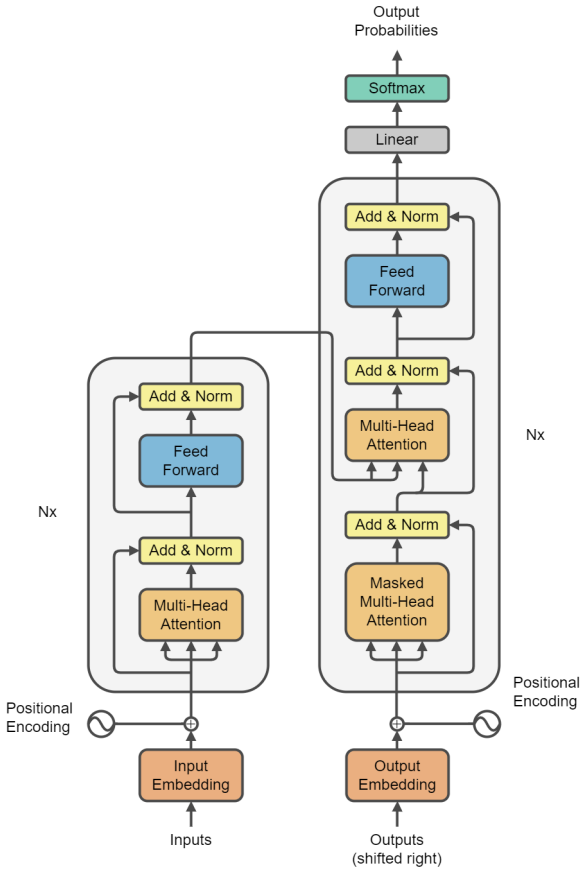

Figure 2.15: Figure of the transformer architecture. This figure is based [73, Figure 1].

Then similar instances are moved closer to each other, and dissimilar instances moved apart, iteratively [72, Ch2 p2581]. The lower dimension conditional distribution uses *student's t-distribution* rather than *Gaussian*. t-SNE also uses gradient decent to minimize the cost function for making the lower dimension conditional probability distribution as similar to the high dimensional distribution as possible.

For a while, t-SNE was the state-of-the-art dimensionality reduction technique used when visualizing, until UMAP was proposed. UMAP is arguably better at preserving the global structure of the data, and is faster to train then t-SNE [64, Ch5.1 p30, Ch8 p50].

2.4.2 UMAP

UMAP works very similarly to t-SNE. The difference between them is mainly linked to how the high-dimensional space is computed and how the algorithms optimize the lower-dimension representation. UMAP constructs a high-dimensional graph representation called

"fuzzy simplicial complex". This is just a graph where the weighted edges are the likelihood of two points being connected. UMAP then optimizes to make the lower-dimension as similar to the graph representation as possible. We will not further explain the algorithm used by UMAP. UMAP preserves more of the global structure, is faster and more scalable than t-SNE [64, Ch8 p51] [34].

Chapter 3

Natural language processing

Natural language processing is a subfield of linguistics and computer science (mainly artifical intelligence), considering interactions between human language and computers. NLP techniques allow computers to understand human language via text or speech and communicate back in human language [62, vii]. Typical applications for NLP are speech recognition, dialogue systems (section 4.2), information retrieval, question answering, and machine translation. All of these applications are strongly impacted by deep learning [62, vii].

3.1 NLP tasks

In this section will we briefly explain two specific NLP tasks relevant for dialogue systems. First we will explain language models, which are relevant to how modern semantic representations are made. Secondly we will explain name entity recognition which is used for information extractions.

3.1.1 Language models

Language models are models that assign probability to a sequences of words, based on what words they have already seen. N-grams models are a simple example of this: Given the sequence of words (the n-gram), what is most likely to be the next word? Today the state-of-the-art language models are usually based on RNNs and transformers [52, Ch3.1 p31] [63,

Ch4.3 p61]. The learned probability distributions can then be used as a vector representation of the words (word embeddings), and may be further used in other natural language tasks. The vector representation is beneficial since the vector representation has a smaller amounts dimension and gets rid of the sparsity [63, Ch1.4 p5].

3.1.2 Name entity recognition

Named entity recognition (NER) is the task of labeling a sequence with tags. Named entity is anything that can be referred to with a tag; name, date, place, etc. [52, Ch8.3 p164]. A sequence is run through a model and outputs a sequence of tags of the same length as the input. The named entities can then be extracted based on the tags. This task is known as sequence labeling. RNNs are quite suitable for such tasks [52, Ch8 p159], as we mentioned in section 2.3.2. The *many-to-many* architecture in Figure 2.10 illustrates how this could be done.

3.2 Text normalization

Text normalization is usually performed before almost any NLP task, and is also referred to as *text preprocessing* [52, Ch2.4] [67, Ch2.1 p10]. Text normalization is necessary to convert the text into a more convenient standard form [52, Ch2 p2]. Some common steps of text normalizing are *tokenization* and normalizing word formats [52, Ch2.4 p14].

3.2.1 Tokenization

Tokenization is the task of separating words from the text, referred to as *tokens*. Words are usually separated with whitespace, but the seperation can in some cases be more challenging. For instance; *I'm* should be divided into *I* and *am*. Sometimes we would like to segment multiple words into one token; *New York* and *rock 'n' roll* for example. In some languages, like Japanese, words are not even separated by whitespace, which makes tokenization a bit more challenging [52, Ch2 p2-3]. However, these examples are less relevant for Norwegian.

After the text has been tokenized, different techniques may be applied on the tokens to normalize the text. Depending on the task, different compositions of normalization techniques are used to preprocess the text into a suitable format [52, Ch2.4.4 p20] [31].

3.2.2 Normalizing word formats

As mentioned, there are different techniques to normalize the text. The simplest technique is *case folding*. In case folding every token is converted into lowercase. "Hello" is converted into "hello" and both versions of the word are now the same token [52, Ch2.4.4 p21]. In the following we will go through several normalization techniques including *lemmatization* and *stemming*, *stop word removal* and noise removal.

Lemmatization is the process of reducing the inflectional form of a word to its *lemma* (root word). For instance, *am*, *are*, and *is* have the same lemma *be*. Lemmatization could be done via a dictionary lookup. Stemming is a simpler but cruder method for reducing the inflectional forms. It does this by chopping the ends of words to get their stem (the form of the word with all inflectional affixes removed), and hoping it gets it right [30] [52, ch2 p3, Ch2.2.4 p20-21]. The result of either performing stemming or lemmatization of selected words is shown in Table 3.1.

Table 3.1: Table comparing the result after performing either stemming or lemmatization of selected words.

Word	**Stemming**	**Lemmatization**
Love	Lov	Love
Loves	Lov	Love
Loved	Lov	Love
Loving	Lov	Love
Innovation	Innovat	Innovation
Innovations	Innovat	Innovation
Innovate	Innovat	Innovate
Innovates	Innovat	Innovate
Innovative	Innovat	Innovative

Stop word removal is the process of removing highly frequent words that may carry little semantic value. These words are called *stop words* and are collected into a *stop list* that is used to filter the words from the text [51, Ch23.1.2 p468]. There are however no universal list or agreed upon rules on what words to include in your stop list. Some cases can use a list of 9 words, others a list of 571 [67, Ch19.2.2.2 p458].

Noise removal is the process of removing what is considered as unnecessary noise for a specific task. This might be as simple as disposing punctuation, or removing html syntax [31].

3.3 Semantic representation

Most NLP related tasks rely on semantic representation of words, phrases, sentences or documents. These representations typically appear in vector form. In this section we will start off by explaining the *distributional hypothesis* (section 3.3.1), and sparse and dense vectors (section 3.3.2). Then will we explain some of the techniques used for creating word (3.3.3) and sentence representations (3.3.4).

Section 3.3 is inspired by the blog *From Words To Vectors* [12]

3.3.1 Distributional hypothesis

A very simple, but extremely profound concept in NLP is the distributional hypothesis. The distributional hypothesis is elegantly summarized by John R Firth [55]:

> *"You shall know a word by the company it keeps."*

The distributional hypothesis is the basis for semantic word representation learning. Following the hypothesis one can project the semantic meaning of a word into vector space [63, Ch2.3 p14, Ch3.1 p43]. This representation of a word in terms of a vector is known as *word embeddings* [52, Ch6 p102]. Words that occur in similar context, tend to have similar meaning [51, Ch6 p96]. Hence, they tend to appear closer in vector space. These representations are usually projections to vector space [63, Ch3.2 p45]. Word2vec (section 3.3.3.4) for instance, utilizes this concept [63, Ch2.1 p14]. Figure 3.1 shows an example of positive, negative and neutral words representations, projected into 2D-space for visualization.

3.3.2 Sparse and dense

We can categorize vector representations into *sparse* and *dense* [51, Ch6.13 p123].

Sparse refers to vectors, matrices or tensors usually of high dimension where most of the values are zero. High-dimensional sparse vectors are very ineffective to work with considering these vectors carry little information and take up a lot of storage [15].

Dense vectors, in terms of semantic representations, refers to vectors with only 50-1000 dimensions containing real-valued numbers rather than mostly zeros. It turns out that dense vectors perform better in every NLP task. The intuition behind this is that a classifier has fewer parameters to learn which helps with generalization and avoids overfitting. Moreover, dense vectors do a better job at capturing similarities between words [51, Ch6.8 p112].

Figure 3.1: Visualizing an example of projecting positive, negative and neutral words in vector space. The axis function only as a coordinate system for how words roughly relate to each other. This figure is based on [52, Figure 6.1]

3.3.2.1 Distributed representation

Distributed representation refers to object representation produced by deep learning algorithms. Distributed representation are typically low-dimension, dense vectors [63, Ch1.1 p2]. For example, word2vec learns a distributed representation of words [63, Ch3 p2]. Distributed representation of words has significantly improved the performance of almost all NLP tasks [63, Ch Preface p vi].

3.3.3 Word representation

Word representation, aims at representing words in vector-space. As mentioned, these vector representations are what is known as embeddings [51, Ch6 p96]. Sometimes embeddings refer more strictly to dense vectors like word2vec (3.3.3.4), rather than sparse vectors [51, Ch6.2 p100]. You could either train a new embedding based on your *corpus* with for example word2vec, or use a pre-trained embedding, like BERT (3.3.3.6). A corpus is a computer-readable collection of text or speech [52, Ch2.2 p11].

3.3.3.1 Basic numbering

The initial thought when one thinks of representing words for a machine learning algorithm, might be a alphabetical numbered list of the corpus *vocabulary*. The vocabulary is a list of

distinct words within the corpus and is denoted as V[51, Ch2.2 p12] (see Table 3.2). However, this enumerated representation carries no meaning. Also, it does not handle words that are not within the vocabulary, also including misspellings. These words are known as *Out Of Vocabulary* (OOV), unknown words. Lastly, the semantic ordering usually gives a misguiding relation between the words. This is a drawback considering that machine learning algorithms might pick up this relation and evaluate the words according to the semantic order [37].

Table 3.2: Example of encoding each word with a unique number.

Word	**Value**
A	1
...	...
Bil	23
...	...
Kjøretime	534
...	...
Sertifikat	1243
...	...
Trafikk	1323

3.3.3.2 One-hot-encoding

One-hot encoding avoids the ordered relationship by treating each word individually. Words are represented as a vector with the same dimension as the vocabulary. Each dimension/index corresponds to a unique word in the vocabulary. Only one index can be "1" at the time. The remaining dimensions are "0" [63, Ch2.2 p14]. For example: *"The quick brown fox jumps over the lazy dog"* would result into the matrix shown in Table 3.3.

Table 3.3: One-hot encoding of the phrase "The quick brown fox jumps over the lazy dog".

Word	**One-hot**
the	[1, 0 ,0 ,0 ,0 ,0, 0 ,0]
quick	[0, 1 ,0 ,0 ,0 ,0, 0 ,0]
brown	[0, 0 ,1 ,0 ,0 ,0, 0 ,0]
fox	[0, 0 ,0 ,1 ,0 ,0, 0 ,0]
jumps	[0, 0 ,0 ,0 ,1 ,0, 0 ,0]
over	[0, 0 ,0 ,0 ,0 ,1, 0 ,0]
the	[1, 0 ,0 ,0 ,0 ,0, 0 ,0]
lazy	[0, 0 ,0 ,0 ,0 ,0, 1 ,0]
dog	[0, 0 ,0 ,0 ,0 ,0, 0 ,1]

However, one-hot encoding have some problems. Firstly, the vectors quickly become very high-dimensional and sparse. Norwegian has over 300,000 words [6] and encoding a Norwegian word would result in a 300,000-dimension vector with only one index as "1". One-hot encoding also does not capture any semantic relation between words. They are all independent units and are equally distanced apart in vector space. Meaning, "apple" has the same relation to "banana" as it has with "cat" [63, Ch2.1 p13]. One-hot encoding does not solve the OOV problem either.

3.3.3.3 Latent semantic analysis

Latent semantic analysis (LSA), sometimes referred to as *Latent semantic indexing* (LSI) [52, Ch6 p131], learns word representations from the *term-document* matrix. The term-document matrix has rows of words and columns of documents. Each row has cells carrying the frequency of a word/term in each documents [52, Ch6.13 p129], see Figure 3.2. LSA applies *Singular value decomposition* (SVD), and factorizes the term-document into three matrices.

$$M = E\Sigma D^T \tag{3.1}$$

M is the term-document matrix. E corresponds to the word embeddings, D to the document embeddings, and Σ the singular values/importance. Each row vector in E represents a word that corresponds with a term in M. The word embeddings are represented as a distribution of topics, hence it is a dense vector. How important a topic is, is defined by the singular value. We can remove a portion of the less important topics, to reduce the vector size of the embeddings [63, Ch2.3.2 p17].

3.3.3.4 Word2vec

Word2vec is a toolkit proposed and released by Google in 2013 [63, Ch2.3.3 p18]. In the paper where word2vec was introduced, two very similar neural network architecture models were proposed for learning distributed representations of words. These models are known as *continuous bag-of-words* (CBOW) and the *Skip-gram* [63, Ch2.3.3 p18]. The paper observed large improvements in accuracy using these methods, and that they achieved this at a much

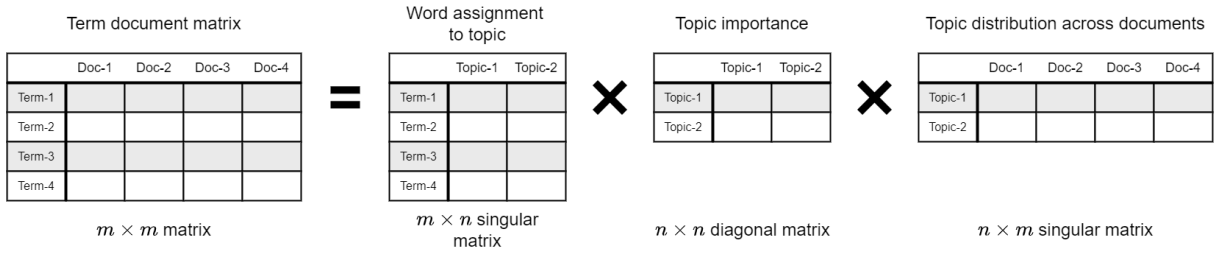

Figure 3.2: Illustration how SVD is used in LSA to generate word and document representations distributed as topics. This figure is base on a figure found in the blog post; *Latent Semantic Analysis using Python* [18].

lower computational cost [71, p1]. Word2vec was for a while the de facto for word embedding [36].

Both the CBOW and Skip-gram are shallow neural networks. They only consist of one input layer, one hidden/projection layer and one output layer [71, Ch3.1 p4]. The dimension of the hidden layer is between 50-100, while the input and output layer have the same dimension as the vocabulary [71, Ch1.1 p2]. The input word/words are encoded using one-hot encoding of the vocabulary. The output is also one-hot encoding, using a softmax function to predict the correct word. After training, the final embeddings are stored in the projection matrix, the weights between input layer and projection layer [71, Ch2.1 p3] [41].

CBOW learns the embeddings by predicting the center word given the context, see Figure 3.3. By context we mean words surrounding a word within a window. The window is a hyperparameter and is denoted as C. The window defines the range of precursing and subsequenting words to include during training. Training is achieved by finding the optimal parameters for the model that maximizes the *log-likelihood* function:

$$L = \frac{1}{T}\sum_{t=1}^{T} log\, p(w_t|w_{t-C}, ..., w_{t-1}, w_{t+1}, ..., w_{t+C}). \tag{3.2}$$

The network computes the softmax function denoted as $p(w_a|w_b)$. The model is then fed a sequence of training words $[w_1, w_2, w_3, ..., w_T]$ to process [71, Ch2.1 p3] [41]. Skip-gram is very similar to CBOW, but instead learns the embedding by predicting the context given a target word, see Figure 3.3. Skip-gram is also fed a sequence of training words, but optimizes its parameters to maximize the *average log probability*

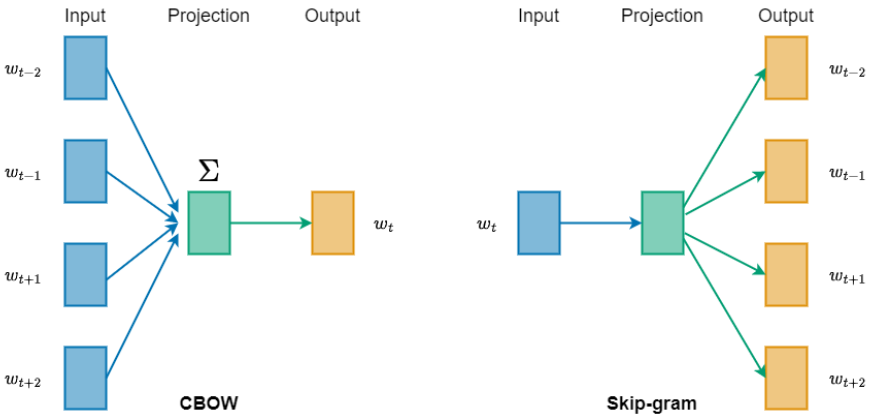

Figure 3.3: Illustration of the CBOW and Skip-gram architecture. CBOW predicts a target word $w(t)$ given the context. Skip-gram predicts the context given the target word $w(t)$. This figure is based on [71, Figure 1]

.

$$L = \frac{1}{T}\sum_{t=1}^{T}\sum_{-C\leq j\leq C, j\neq 0} log\, p(w_{t+j}|w_t). \tag{3.3}$$

Increasing the range of the context/window, improves the quality of skip-grams word embedding, but also increases the computational complexity. [65, Ch2 p2-3]

Skip-gram is better at capturing semantic relationships, while CBOW is slightly better at syntactic relationships. However, Skip-gram takes much longer than CBOW to train [71].

Word2vec also demonstrated the analogical properties hidden within the embeddings. By performing simple algebraic operations like; $King - Man + Woman$, it would result in a vector very close to *Queen* [66]. Figure 3.4 illustrates the vector location of "Man", "Woman", "King" and "Queen", and how they relate.

The distributional representation provided by word2vec is much better than our previous representation techniques [71, Ch1 p1]. Still, the word embedding is restricted by the vocabulary and does not solve OOV (3.3.3.1). Word2vec embeddings are *static embeddings*, meaning the model learns a fixed representation for each word [51, Ch6.8 p112]. So how do we distinguish between "Apple" the fruit and "Apple" the company? This is where *Contextualized word embedding* come in.

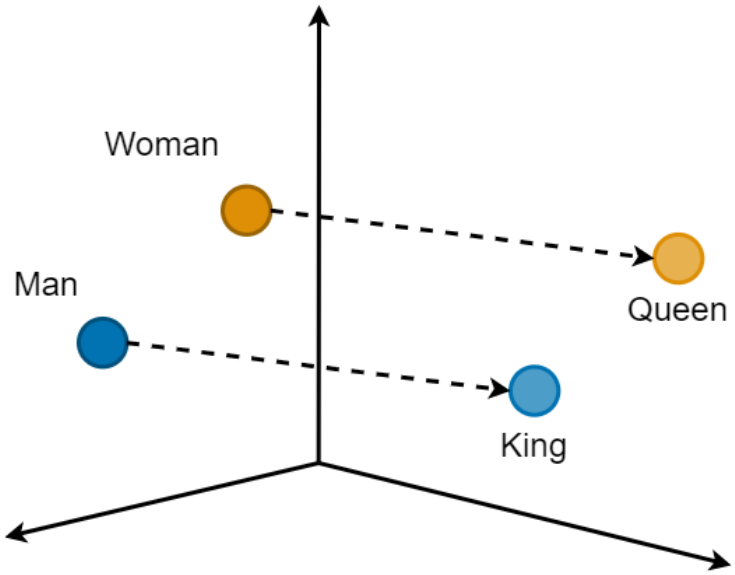

Figure 3.4: Illustration showcasing the relationship between “woman”,and “queen”, compared with the relation between “man” and “king” in vector space.

3.3.3.5 Contextualized word embedding

NLP took a big leap in 2018 when *pre-trained language models* like BERT, *Generative Pre-trained Transformer* (GPT) and *Embeddings from Language Models* (ELMo) came out. Pre-trained language models are trained on a large corpus, use more advanced architectures and more computing resources, than static embeddings like word2vec. They also take into account the context of the words, which results in a complex dynamic representation of words based on their surrounding words. This is especially useful when a word has multiple meanings. A new trend also emerged along with pre-trained language models. We can now use the pre-trained language models for new NLP tasks and achieve state-of-the-art performance by extracting embeddings or *fine-tune* the models for the specific task [63, Ch1.4 p6]. Fine-tuning is the process of taking a pre-trained model and further train it, usually with some extensions of the network. The model is trained for a new specific task, but utilizes the perception of language within the pre-trained model [52, Ch11 p243].

Pre-trained language models had huge success and got a lot of attention in the NLP and machine learning communities [63, Ch1.4 p6]. Both BERT and GTP were inspired by the success of transformers (section 2.3.4), which they use [63, Ch4.4.4.2 p67].

3.3.3.6 Bidirectional encoder representations from transformers

BERT is a language representation model. It is a multilayered bidirectional transformer encoder based on the architecture we explained in section 2.3.4. It consists of multiple layers of transformer blocks that make up the encoder [54, Ch3]. BERT's framework was designed to pre-train a bidirectional representation of text, followed by fine-tuning the model for different tasks. BERT is pre-trained using two supervised tasks; *masked language modeling* and *next sentence prediction* [54, Ch3.1]. In masked language modeling, a certain percentage of the input tokens are masked, and the model predicts these masked words. In sentence prediction the model aims at predicting which one out of two possible sentences is the correct following sentence [54, Ch3.1]. BERT's input and output representation use *WordPiece* embedding [54, Ch3]. WordPiece divide words into a limited set of common sub-words called WordPieces [74, Ch Abstract]. For example, "embedding" is split into "em", "##bed," and "##ding". When the model sees a new or misspelled word, it might have learned a representation of that word's subwords. This solves the problem with OOV. Additionally, this reduces the input/vocabulary size of the embedding as WordPiece only has a vocabulary of 30,000 tokens [54, Ch3].

3.3.4 Sentence representation

Sentence representation is an important task for many NLP applications. For example, text summarizing, machine translation, sentimental analysis and dialogue systems all include/use sentence representation [63, Ch4 p59]. Before the deep learning era, sentences were represented as one-hot-encoded vectors, bag-of-words or *term frequency-inverse document frequency* (tf-idf) [63, Ch4.1 p59].

3.3.4.1 Bag-of-words & TF-IDF

Bag-of-words is a simple and common method for representing sentences and documents. BoW have the same length as the vocabulary. Each index represents a word, as with one-hot encoding, and the corresponding value is the frequency of that word in the sentence/document (see Table 3.4). Simplified, BoW is the same as summing up all the one-hot encodings occurring in a sentence [63, Ch5.2 p92].

Table 3.4: Example of bag-of-word inspired by a figure found in the blog; *Spam Filtering Using Bag-of-Words* [29].

Sentences	**the**	**red**	**dog**	**cat**	**eats**	**food**
the red dog	1	1	1	0	0	0
cat eats dog	0	0	1	1	1	0
dog eats food	0	0	1	0	1	1
red cat eats	0	1	0	1	1	0

Tf-idf is a method that enhances bag-of-words ability by weighting the importance of words in a document [63, Ch5.2 p93]. Tf-idf does this by punishing highly frequent words across documents, and reward highly frequent words occurring only in few documents. By doing this, we get rid of words that appear regularly but has little meaning, such as; "a", "an" and "the" [63, Ch4.2 p60]. Tf-idf is calculated as following:

$$\begin{aligned} \mathrm{tf}_{t,d} &= \log_{10}(\textit{term-document matrix} + 1) \\ \mathrm{idf}_t &= \log_{10}\frac{N}{\mathrm{df}_t} \\ \text{tf-idf}(t,d) &= \mathrm{tf}_{t,d} \cdot \textit{idf}_t \end{aligned} \tag{3.4}$$

Tf-idf is the product of the term frequency (tf) and indirect document frequency (idf). df_t is the document frequency of a term t, and N the total number of documents [52, Ch23.1.1 p496].

Both BoW and tf-idf have two main issues. The vector space quickly becomes sparse and of high dimensions. Secondly, the sequence of words and their semantic meaning are not taken into account. This means that "Alice stole from Bob" is equal to "Bob stole from Alice" [63, Ch4.1 p59-60]. A simple approach to overcome the sparsity could be done by taking the average of all the word embeddings, for instance by using word2vec. However, this approach still ignores the order of the words [61, Ch1]. To preserve the information about the word order we can therefore use a language model such as doc2Vec.

3.3.4.2 Doc2vec

Doc2vec is inspired by word2vec and created by the same lead author. It has two algorithms; *distributed memory*, which is similar to CBOW, and *distributed bag of words*, which is similar to Skip-gram. Figure 3.5 illustrates their architectures.

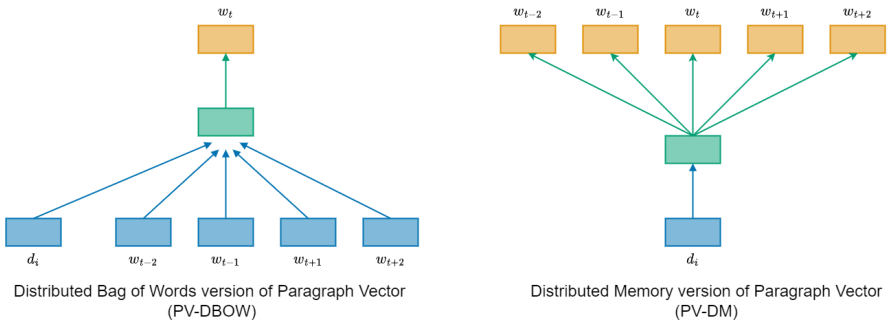

Figure 3.5: Illustration of distributed memory and distributed bag of words. This figure is based on [61, Figure 2, Figure 3] [61]

Both models create a distributed representation of sentences and documents. Distributed memory is considered to consistently work better than distributed bag of words [61, Ch3.4]. The representations may further be fed to any conventional machine learning techniques as features for analyzing or solving tasks [61, Ch2.2].

3.3.4.3 BERT for sentence representation

BERT is also viable for sentence and document representation. BERT was trained to unambiguously represent a single or a pair of sentences. The paper that introduced BERT defines a "sentence" as *"an arbitrary span of contiguous text, rather than an actual linguistic sentence"*. The "sentence" is converted into a "sequence" of tokens. There is then added a special token ($[CLS]$) in front of every sequence. This special token is known as the classification token and is the aggregated representation of the sequence. It may be used for classification tasks. Another special token is ($[SEP]$). This token is used when we pass a sentence pair to BERT. This is for instance useful for question-answer tasks [54, Ch3].

3.3.5 Document representation

Document representation aims at capturing the semantic information of a whole document; blog, article, paper etc. We have already covered some methods for document representation; bag-of-words, tf-idf and doc2vec. Another approach for document representation is *topic modeling*. Topic modeling is an unsupervised machine learning technique that finds hidden semantic structures across documents. It is helpful for understanding large amounts of data and searching/clustering similar documents [63]. *Latent dirichlet allocation* (LDA) is perhaps the most common method. In LDA, each document is represented as a distribution of topics, and each topics is a distribution of words, see Figure 3.6.

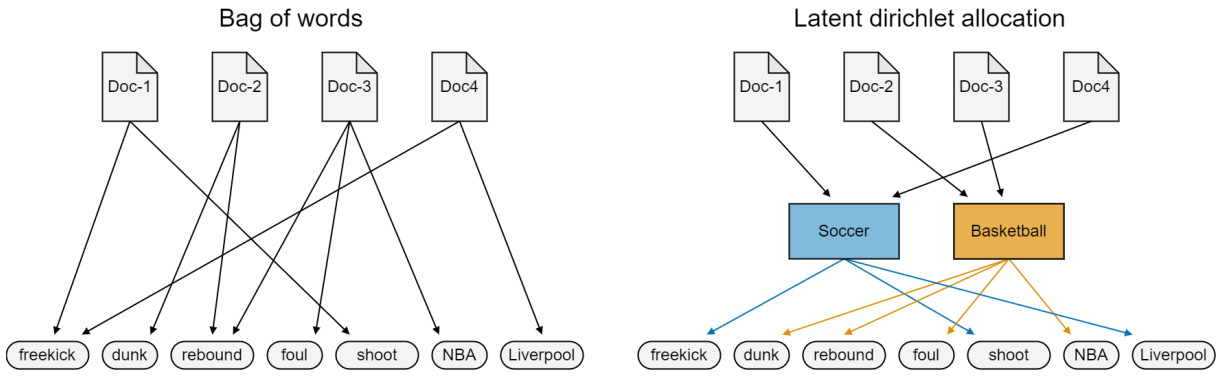

Figure 3.6: Illustration comparing bag-of-word and latent dirichlet allocation. This figure is base on a figure found in the blog post; *Topic Modeling with Latent Dirichlet Allocation* [32].

Chapter 4

Dialogue systems

This chapter covers what a dialogue system is, the two categories dialogue systems fall into, and the modern standard architecture for a dialog system, namely the *dialogue-state architecture*.

4.1 What is a dialogue system?

Dialogue systems, sometimes called *conversational agents* or *virtual agents* [62, Ch3 p49], are programs that interact with human users via natural language [52, Ch24 p521]. They may also be referred to as *chatbots*. However, in [52, Ch24] and this thesis chatbots refer to a category of dialogue systems, see section 4.2.1.

Dialogue systems may be used for a range of applications, like *educational environments*, *customer service*, *health care* and *industrial use cases*. In education, dialogue systems can be a personal assistant for students providing educational support and content, as well as helping with administrative issues. Dialogue systems can also be available 24 hours for customer support. Further more, they can be of assistance in health care, providing information about illness, symptoms, treatment, products and services. Another useful application of dialogue system, they could be used to handle booking reservations for restaurants [40, Ch8 p13-15].

4.2 Dialogue system categories

Dialogue systems can be divided into two categories; *chatbots* and *task oriented-dialogue systems*. Chatbots, also known as a *non-goal-oriented dialogue systems* or *open-domain dialogue systems*, are used to simulate human conversation, usually for entertainment. A task-oriented dialogue system, or *goal-oriented dialogue system* aims to help users complete tasks through dialogue [52, Ch24 p521]. Since the two different categories serve different purposes, they also have different system designs and components [62, Ch3.1 p50] [76, p1].

4.2.1 Chatbots

Chatbots are the simplest type of dialogue systems. They have the main goal of mimicking human conversation by picking the most likely response based on the utterance from the user. They are mostly used for entertainment. There are two types of architecture for chatbots; *rule-based* and *corpus-based* [51, Ch24.4 p496]. It is also possible to combine both methods into a *hybrid architecture* [51, Ch24.2.3 p503-504].

The rule-based architecture uses rules, or patterns, to recognize different phrases which are then transformed into responses. Rule-based systems have been around for a long time and are still used today [51, Ch24.2.1 p498-500]. *ELIZA* is considered to be the first and most important chatbot in the history and was developed in 1966. ELIZA used rules to imitate a Rogerian psychologist where patients statements are reflected back at them [52, Ch24.2.1 p527]. *Pandorabots* is a modern platform for developing dialogue systems and is based around *artificial intelligence markup language* (AIML) [24]. AIML is an extension of the markup language XML and is used to define the rules in the dialogue system [3].

Corpus-based systems rely on human-to-human conversation instead of rules. They are very data-intensive and require a lot of it for training. The system responds to users by using *retrieval* or *generative* methods. Retrieval methods finds and copies the most suitable response from the corpus. Generative methods generate a response, usually with the help of a language model or a encode-decoder model [51, Ch24.2.2 p500-503].

4.2.2 Task-oriented dialogue system

Task oriented dialogue systems try to solve a specific task within a domain as efficiently as possible. The task is given by the user through language, either as text or by speech. The tasks usually involve finding information within a database and presenting it to the user, gathering information from a user, or performing certain actions [53, Ch3.1 p760-761].

Most modern task-oriented dialogue systems are based on the *dialogue-state* architecture, also called the *belief-state* architecture. The dialogue-state architecture revolves around *frames* [52, Ch24.3 p534, Ch24.4 p537-538]. Frames are data structures that capture the semantic representation of what the user utters or queries. Each frame has a collection of *slots* that need to be initialized, specifying what the dialogue system needs to know [52, Ch24.3 p534] [62, Ch2.1 p25].

4.3 The dialogue-state architecture

The Dialogue-state architecture consists of six components. *Automatic speech recognition*, *natural language understanding* (4.3.1), *dialogue-state tracker* (4.3.2), *dialogue policy* (4.3.3), *natural language generation* (4.3.4) and *text-to-speech*. A system with all of the components is known as a *spoken dialogue system*. The pipeline of a spoken dialogue system is shown in Figure 4.1. This thesis will not elaborate more on the automatic speech recognition and text-to-speech components. However, the four remaining components will be further explored in this chapter, and makes up what is called a *textual dialogue system* [51, Ch24.4 p508-509].

Sometimes the dialogue state tracker and dialogue policy are combined into one component. This component is called the *dialogue manager*. The dialogue manager has the same purpose as the dialogue state tracker and the dialogue policy in one [62, Ch3.1 p50] [53, Ch3.3 p763].

4.3.1 Natural language understanding

Natural language understanding is the first component of a dialogue system. It tries to capture a semantic representation of a user's utterance by solving three tasks; *domain classification*, *intent determination* and *slot filling*. Domain classification refers to the process of defining the domain within what the user is talking about. Is the utterance about restaurants

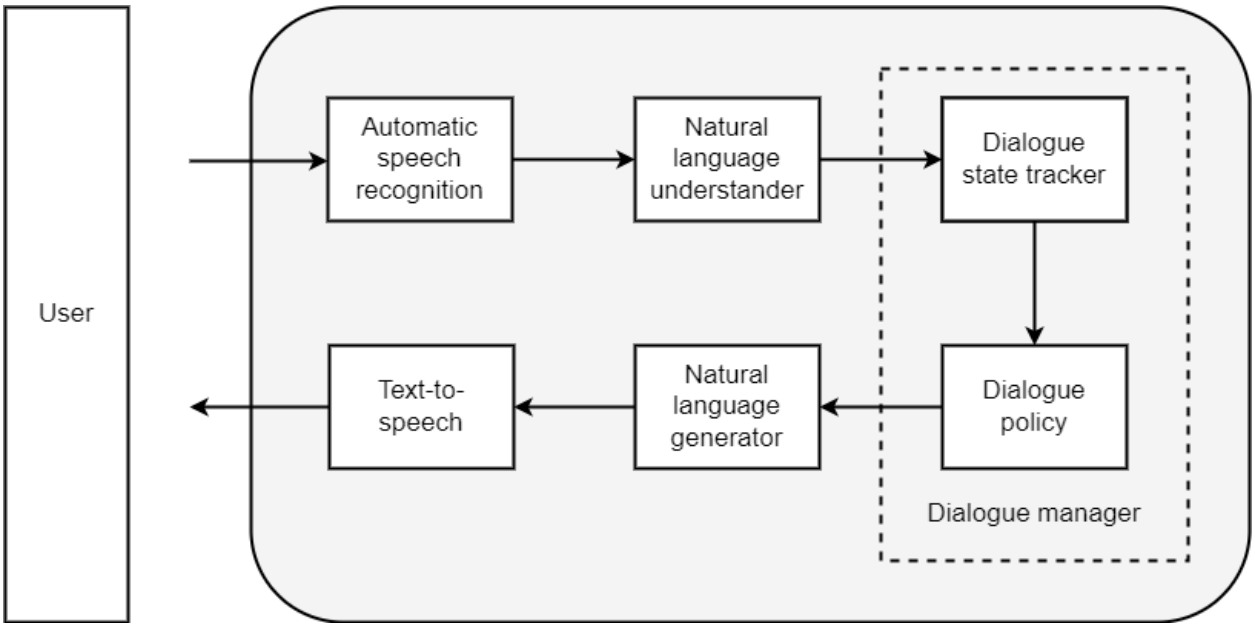

Figure 4.1: Pipeline of the spoken dialogue system consisting of; automatic speech recognition (translates speech to text), natural language understanding (creates a sentence representation from the utterance), dialogue state tracker (tracks the state of the dialogue), dialogue policy (chooses what action to perform next), natural language generator (generates the response) and text-to-speech (generates speech based on the response).

DOMAIN:	AIR-LINE
INTENT:	SHOW-FLIGHTS
ORGIN-CITY:	Boston
ORGIN-DATE:	Thursday
ORGIN-TIME:	morning
DEST-CITY:	San Francisco

Figure 4.2: An example of a frame extract from the utterance "Show me morning flights from Boston to San Francisco on Tuesday". The key to each slot is listed in the left column, while the key-values are listed in the right column. This example is taken from [51, Ch24.3.2 p506].

or airlines? Intent determination aims at figuring out what the user wants to accomplish. Does the user want to book a flight or ask for the weather tomorrow? The last task is slot extraction and aims at extracting any parameters given in the users utterance. These parameters are then used to fill slots belonging to the respective intent. For instance, to provide a weather forecast, the system would need to know the location of interest for the user [51, Ch24.3.2 p506-507].

Here is an example: The utterance *"Show me morning flights from Boston to San Francisco on Tuesday"*, would belong to the domain AIR-TRAVEL and the user's intent would be SHOW-FLIGHTS. The corresponding slots to fill would be; ORGIN-TIME="morning", ORGIN-CITY="Boston", DEST-CITY="San Francisco" and ORGIN-DATE="Tuesday" [51, Ch24.3.2 p506]. Once the information has been extracted, it is put into a semantic representation, which is known as a frame [62, Ch3.3.1 p55]. Figure 4.2 shows an example of a frame given the example we have just used.

Before any of the mentioned tasks can be solved, however, the utterance first needs to be processed into a sentence representation, for instance by passing it through BERT (Ch 3.3.3.6) [51, 24.4.2 p510].

4.3.1.1 Domain and Intent classification

Both domain classification and intent determination are defined as classification problems [62, Ch2.3.1 p27]. As mentioned, the domain is the topic of the dialogue. In cases where

O	O	B-ORGIN-TIME	O	O	B-ORGIN-CITY	O	B-DEST-CITY	I-DES-CITY	O	B-ORGIN-DATE
Show	me	morning	flights	from	Boston	to	San	Francisco	on	Tuesday

Figure 4.3: An example of BIO-tagging. Each word is assigned a tag/label. B indicates the beginning of a span of interest, I is inside a span, and O means outside any span of interest [51, Ch8.3 p154].

there is only a single domain, this task would be unnecessary. Many domains would require multiple frames to distinguish slots for different actions and domains [62, Ch24.3.1 p506]. The multi domain dialogue systems have become the modern standard. Domain classification is usually done as a top-level delegation followed by intent determination and slot filling. This way of modeling provides some advantages, like abstracting domains actions and slots. But there are also some disadvantages. For example, you would need to train as many models as there are domains [62, Ch2.4.3 p37].

4.3.1.2 Slot filling

Slot filling aims at extracting potential slots given in an utterance [51, Ch24.3.2 p506]. Slot filling is considered one of the most challenging tasks in spoken language understanding [62, Ch3.2.1 p53] and is defined as a sequential classification problem [62, Ch2.4.3 p37].

Earlier systems used handcrafted rules to extract slots, and this technique is still common in the industry [51, Ch24.3.2 p506]. The handcrafted rules can be used as a descent solution while generating data. When the handcrafted rules have provided sufficient data, the developer may use bootstrapping machine learning techniques that can outperform the original rules [51, Ch24.4.2 p511]. With the use of machine learning each word is passed through a classifier and paired with a label. The label might correspond with a slot or not. The words are then extracted and assigned to a slot based on the label [51, Ch24.4.2 p511].

One technique to solve the sequence classification problem is with BIO-tagging. BIO-tagging labels each word/token in a sentence with a tag. B indicates the beginning of a span of interest, I means inside the span of interest, and O means outside the spans of interest [51, Ch8.3 p154]. Figure 4.3 shows an example of sequence labeling with BIO-tagging of the sentence *"Show me morning flights from Boston to San Francisco on Tuesday"*.

As mentioned, domain classification and intent determination are defined as classification problems, while slot filling is a sequence classification problem. Because they are different

problems they initially required different solutions. With the advances in deep learning, it is however possible to solve both types of problems with one unified model. Hakkani-Tür [62, Ch2.4.3 p37] proposed a single RNN approach that preformed all three tasks. The utterance is enclosed with an ⟨BOS⟩ (beginning of sentence) and a ⟨EOS⟩ (end of sentence) tag, and the domain and intent classification is done on the ⟨EOS⟩ output.

4.3.2 Dialogue state tracker

The dialogue state tracker, sometimes called the belief tracker, maintains the systems perception of the current state of the dialogue. It does this by estimating the dialogue state, also known as the belief state, at each step of the dialogue [62, Ch3.3.2 p56] [75].

A simple dialogue state tracker might just use the output of a sequence model at the end of each sentence [51, Ch24.4.3 p512], or the current state of a given frame, as a representation of the dialogue state. A more sophisticated dialogue state tracker would also include the previous dialogue state, and the latest utterance from both the user and the system. The state-of-the-art for dialogue state tracker uses neural networks to estimate the dialogue state [62, Ch3.3.2 p56].

4.3.3 Dialogue policy

The dialogue policy decides what action to performe based on the dialogue state. It aims at guiding the dialogue toward completing a task successfully [62, Ch3.2.3 p54]. Early systems had quite simple policies; ask questions until the frame is filled, then preform a query [51, Ch24.4 p509]. More sophisticated policies could be defined as a probability distribution of actions, given the dialogue state. The probability could be estimated using a classification algorithm. Even more sophisticated policies could be trained via reinforcement learning [52, Ch24.4.4 p542].

4.3.4 Natural language generation

The last component of the dialogue system is the natural language generation. This component generates a response based on the output from the dialogue policy. A simple and quite common solution is to return a pre-written template as the response. A more sophisticated solution is to generate the response based on the context passed from the dialogue policy. A generated response seems more natural, but sufficient training data for this method is unfortunately hard to come by [51, Ch24.4 p509] [53, Ch3.3.1 p12].

Chapter 5

Related work

This thesis is based on research done in the field of dialogue systems. Most influential books we have based this thesis on are *Speech and Language Processing* [52], *Deep Learning in Natural Language Processing* [42] and *Representation Learning for Natural Language Processing* [63].

In this chapter we cover similar/related work for this thesis. We will cover two papers and one master thesis. The first paper (section 5.1) compares combinations of both different embedding techniques and machine learning algorithms for intent classification of utterances. It also experiments with hierarchical classification. The master thesis (section 5.2) also evaluates combinations of embeddings and machine learning algorithms for intent classification. The master thesis focuses more heavily on hierarchical classification and Norwegian utterances. Both the first paper and the master thesis share similarities with this thesis in regards to the project goal, chosen embeddings (word2vec and BERT) and machine learning algorithms (FFNN, LSTM). The second paper introduces a large dataset as a new benchmark for multi-domain dialogue systems, and underlines the importance of proper datasets.

5.1 Intent Classification for Dialogue Utterances

The *Intent Classification for Dialogue Utterances* [69] investigates several machine learning methods for classifying intents of utterances. Among them was flat classification compared with hierarchical classification. They use *Naïve Bayes* combined with BoW as their baseline.

The dataset used was the curated dataset proposed in [46]. The dataset consists of three small corpora (206, 100 and 190 utterances) with disjoint intents. Each corpora had respectively associated number of intents; 2, 4 and 7. From the dataset, they computed several variations of word embedding using *CBOW* for *word2vec*, *GloVe* and *FastText*. The word embeddings were summed up, averaged and kept as sequences, and used as representation of the utterances.

The machine learning algorithms they explored were *Support Vector Machines* (SVM), *long short-term memory* and *Bidirectional LSTM*. The methods were evaluated using *macro-F1*, which is the unweighted mean of all F1 scores for all the classes. The models were compared with commercial solutions like Rasa and Watson, to name a few.

The baseline model for the flat classifier had a macro-F1 score of 0.541. The SVM models with different embedding techniques ranged from 0.657 to 0.752. The recurrent networks on the flat classifier scored macro-F1 between 0.502 to 0.605. The best performing recurrent model was LSTM with FastText scoring 0.605. The baseline model for hierarchical classification had a macro-F1 score of 0.614, and the SVM models ranged between 0.642 and 0.782. The best model was the hierarchical SVM model combined with FastText average and scored 0.782.

The paper concluded the following. The best performing model was hierarchical, however hierarchical classifiers had mixed results across word embeddings. Taking the average of the word embeddings gave better results than taking the sum [69, p86], this indicated that it is useful correcting the length of the utterance [69, p87]. The SVM models out preformed the LSTM models indicating that the order of words had little impact, due to short utterances. LSTMs are less useful when instances are short [69, p83]. Finally, the paper models were on par with the commercial methods, which they were measured against.

5.2 Exploring pretrained word embeddings for multi-class text classification in Norwegian

In the master thesis *Exploring pretrained word embeddings for multi-class text classification in Norwegian* [43] from *Norges teknisk-naturvitenskapelige universitet* (NTNU), they explore pre-trained Norwegian word embeddings for a hierarchical classification on utterances. They

used pre-trained word2vec (of dimension 100 and 300) and BERT embeddings combined with FFNN and LSTM. They use a baseline model made with Naïve Bayes combined with BoW and used macro-F1 to evaluate the models.

The dataset they evaluated was a large corpus of collected utterances from an existing dialogue system used in Norway's largest bank *Den Norske Bank* (DNB) [16]. The dataset consisted of 1,716 classes and 267,000 utterances were 193,000 of them where Norwegian and following groupings was of Finnish and English. The utterances had varying lengths, but generally quite short.

The baseline model achived an accuracy of 51.6% and a macro-F1 score of 0.396. The thesis evaluated mainly three models; FFNN with word2vec, LSTM with word2ved and FFNN with BERT. First model achieved an accuracy of 67.3% and macro-F1 score of 0.657. The second model scored 86.8% accuracy and 0.849 macro-F1. The last model had an accuracy score of 91.1% and an macro-F1 score of 0.886.

The thesis achieved the best performing model with an FFNN architecture combined with BERT embedding. Second best was LSTM with word2vec of 100 dimension embeddings while LSTM with word2vec embeddings with dimension of 300 was worse than 100. The thesis concluded that word2vec embeddings with the correct model at best come close to models using BERT's embeddings in performance. BERT performed well regardless of model.

5.3 MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling

In the *MultiWOZ - A Large-Scale Multi-Domain Wizard-of-Oz Dataset for Task-Oriented Dialogue Modelling* [48], the authors introduces a new benchmark dataset for dialogue systems. The dataset is a large collection of fully labeled human to human written conversation spanning across several domains. The dataset consists of 7 domains, more than 8,000 dialogues and 113,000 utterances/turns. Additionally the paper establishes a baseline for the dataset for comparison to future studies.

The dataset was introduced to overcome the fundamental obstacle around available data for dialogue systems. The data was collected using crowd sourcing and the Wizard-of-Oz setup

where a human plays the role of the dialogue system. The setup was designed to be easy to operate for the wizard and the users where given easy to follow tasks to fulfill.

Chapter 6

Method

This section covers the methods we used in this thesis. It will cover the following: Data∼(6.1), here we write about the content of the data, point out concerns regarding privacy, and the condition of the data. Labeling (6.2), explains the importance and challenges around data labeling, and the setup we used in this thesis. In Pre-processing (6.3), we will cover the different pre-processing techniques we used to enhance the dataset. Then we will explore the data in Exploratory data analysis (6.4), presenting some descriptive statistics and label distributions. In Generating sentence embeddings (6.5), we generate and visually inspect all of the fixed sized sentence representations. The creation of sequential word embeddings are explained in Generating sequential word embeddings (6.6). Finally, we describe machine learning algorithms with fixed sized embeddings in Training binary classification models (6.7), and the sequential embeddings in Training sequential binary classification models (6.8).

6.1 Data

First we start of by covering the content of the data, followed by a section about privacy and personal data. Lastly we will address the lack of labels in the dataset.

6.1.1 Dataset

As mentioned in the Introduction, the dataset in this thesis was provided by Funbit AS. It is a collection of chat messages/utterances from 1,219 customers sent to a driving school. The dataset consist of 4,948 rows and four columns/features. The columns are; *timestamp* (date and time of the utterances), *generated user id* (random generated id), *enumerated conversation* (the order of the utterance from a user) and the user's *utterance*. The utterances were about driving school related services. This includes general information about licenses and courses, prices and availability etc. Most of the utterances are in Norwegian, both Nynorsk, Bokmål and a mixture. There were some occurrences of English utterances as well, but as the main focus in this thesis was classification of Norwegian utterances, the English utterances was removed during the labeling process.

6.1.2 Privacy

The dataset contain real utterances from real people and involves distribution of personal data. Personal data is any information that is identifiable to a natural person. To name a few; name, identification number, location data, or physical, physiological, genetic, mental, economic, cultural or social specific factors [7].

This thesis did not concern with any personal data, but rather the generic utterances. However, in the original data, personal data were present in the data within the utterances providing information to the driving school (we processed a anonymized dataset). This could be information like phone numbers and e-mails. The privacy information given via text are more challenging to anonymize/censor than separate features in a dataset because they are within the utterance.

We were given a pre-anonymized dataset. All of the utterances had been anonymized with a random id, while the personal data within the utterances had been replaced with masked tags. The masked tags represent the type of information it had replaced. For example; phone numbers was replaced with "@_phone" and email addresses with "@_email". The anonymization is also in line with the data minimization principal from *General Data Protection Regulation* (GDPR) [27]. Because of the pre-anonymized dataset, this work did not process any direct or indirect personal data. The thesis was also registered in RETTE. RETTE is the University of Bergen's registration and overview system for projects concerning personal data [26].

6.1.3 Missing labels

The data was not labeled when we first received it. Additionally, the dataset was a collection of every utterance the driving school had recevied to their Facebook page, both concrete and specific utterances, as well as vague and nonspecific ones. However, it was a good example of how data might appear in a real-life scenario. We will describe more of the labeling process the data had to undergo in the following section.

6.2 Labeling

Labeled data is essential to solve a classification problem. All of the tasks done by the natural language component in a dialogue system are classification problems or sequence classification problems. We therefore needed labeled data to train the classifiers that defines what domain each utterance belongs to and what intent it carries. Labeling the utterances for named entity recognition was skipped since this would have taken a significant amount of time.

6.2.1 Challenges with labeling utterances

The task of labeling utterances is usually very time-consuming and complex [51, Ch13.5 p271]. The MultiWOZ paper points out that most time consuming and challenging part of any dialogue data collection is annotation of the data [48, Ch3.4 p5]. One of the biggest challenges when annotating is defining the set of dialogue acts/intents, which the utterances can be grouped into [49]. We took some inspiration from the MultiWOZ paper when we planned our own approach for labeling the dataset.

Defining suitable labels for domains and intents was challenging considering the levels of services provided by a driving school. For instance, there are multiple types of licenses with different criteria regarding courses, experiences and evaluations. An utterance may therefore involve one or more of the mentioned factors.

Labeling data precisely lay the foundation for creating any supervised model. A recent interview with AI-pioneer Andrew Ng [4] underlines the importance of proper labeling. In the article, Andrew explains that noisy and inconsistent data will affect the performance of the system, and that his company therefore has built tools to flag inconsistent data so that it easily can be identified and relabeled.

6.2.2 Labeling setup

At the very beginning of the thesis we were given a csv-file with the unlabeled dataset. At one point we decided to hand-label the dataset, and therefore we had to label each utterance as efficiently as possible. We used Google Sheets where we defined a list of domains and a list of intents that were selectable via a drop-down menu. Since the utterances might have multiple domains and intents they were given two columns for both (Domain 1, Domain 2, Intent 1, Intent 2). Additionally, we added a column for different driver license categories (B, A, A1, BE, etc.).

The initial set of domains and intents were based on the services provided by the driving school. They were also based on visual inspection of the utterances in the dataset. Some of the initial domains were; "Licenses", anything related to a driving license, "Courses", anything related to courses offered by the driving school, and "Complex", an utterance too complex to be handled by the dialog system. Furthermore, some of the initial intents were; "getInformation", customer asks for information within a domain, "getBooking", customer request their booking order within domain, and "getPrice", customer asks for the price of a product within a domain. All of the domains and intents are specified in the *Guidance for domain and intent labeling* in Appendix A of the thesis. We used the guide as documentation to keep track of the different labels throughout the labeling process. We were well aware that we would continuously adjust the domain and intent labels as we got to know the content of the dataset better. We did so to the best of our ability.

We did all the hand-labeling manually and with a lot of respect for the complexity and importance of the labeling. Additionally, we took some measures to enhance the quality of the data by reviewing and re-labeling a couple of miss-labeled utterances. Once we had some labeled data we could analyze and process the data, and lastly build and evaluate the machine learning models.

6.3 Pre-processing

When we analyzed the corpus, we performed three steps of pre-processing. We gradually increased the level of normalization, and observed the effect of each step. Before we analyzed the corpus, we had to split the labeled data into three; training, validation and test set (section 2.1.1.2). We used the ratio 60/20/20.

6.3.1 Tokenizing

First we tokenized each utterance in the corpus into lists of tokens. We used *NLTK* for tokenization. NLTK is a Python library used for NLP [21].

6.3.2 Normalization

After tokenizing the utterances we applied some normalization to them. We removed punctuations (“.”, “,”, “?”, “!”, “(”, “)”, “:”, “*”) and mapped each token to lower case (case folding). Secondly, we removed stop words by using a stop list provided by the NLTK for Norwegian. This stop list contains 172 unique words that were removed from the corpus. We also removed rare words that occurred less than three times in the corpus.

6.3.3 Lemmatization

Lastly, we lemmatized the tokens using *spaCy*, another NLP library in Python [17]. We used a trained pipeline provided by spaCy, “nb_core_news_sm” to map each token to its lemma [22].

6.4 Exploratory data analysis

Once the data had been tokenized and normalized, we explored it through Exploratory data analysis (EDA). Exploratory data analysis is a widely used method in data science. EDA is used to analyze and summarize the main characteristics of the data, usually through descriptive statistics and visualization. This helps to discover patterns and anomalies in the data, and provides a better understanding of the data as a whole [10].

6.4.1 Descriptive statistics

A quite common, simple and fast method for looking at the main characteristic of a dataset is through *descriptive statistics*. Descriptive statistics are statistical procedures that simplify and summarize the data [56, Ch1.2 p8]. The summary usually captures central tendencies, dispersions and the shape of the data. We used mean and median as the central tendency, and min and max to describe the range. We summarized the length of each of the utterances in terms of tokens, and the length of each conversation in terms of the number of messages sent by a user. We then used histograms to visualize the distribution.

The specific descriptive statistics for analyzing the data took inspiration from this blog [11] and will be more closely described in the following.

6.4.1.1 Frequencies of words and phrases

The simplest way of getting a feeling of the corpus is by counting tokens and observing which tokens are the most frequent. We visualized the most frequent tokens after both the first (6.3.1), second (6.3.2) and third (6.3.3) pre-processing steps.

We can take this method a step further by observing what sequences of words that are most frequent, also known as *n-grams*. The approach for analyzing the frequency of n-grams is exactly similar to the analyzing of the frequency of tokens. The only difference is that we are measuring the frequency of sequences of n tokens and not of the singular tokens in themselves. This reveals the most common phrases in the dataset and may indicate what type of questions are most frequently asked. We did this by first displaying phrases of the corpus when only punctuation was removed. We used n-grams of two, three and four words. Then we removed n-grams that carried any stop words and displayed the remaining n-grams that consisted of two and three words.

We used vertical histograms to visually display the distributions of the frequent tokens and phrases.

6.4.2 Label distribution

It is also common to plot the distributions of classes within the dataset. This gives an insight how the data is distributed across the classes. We are then able to spot majority and minority classes. We visualized both the "domain 1" and "intent 1" distribution via a histogram, see section 7.3.2.

6.5 Generating sentence embeddings

After the explorative data analysis, we generated the different types of sentence representations. The embeddings we created were BoW, the average of word2vec embeddings of utterances, doc2vec and BERT's classification tokens.

We used t-SNE and UMAP to visualize the embeddings and to observe if there were any patterns that correlations with the hand-labeled data. Keep in mind that the generated dimensions had lost their interpretability due to non-linearity (section 2.4). t-SNE captures local relations between utterances (section 2.4.1), while UMAP captures local and global relations between utterances (section 2.4.2).

The different embedding techniques used different degrees of pre-processing. Table 6.1 shows an overview of Dataset A and B, and their pre-processing steps. When generating embeddings, the need for pre-processing is more vital than when extracting them via a pre-trained model. Pre-processing reduces the variation and noise within the data, which makes it more viable to generate embeddings from. We used Dataset A, which was the most heavily pre-processed dataset of the two, when we generated the embeddings. Dataset B was used when extracting the embeddings from BERT. Pre-trained models, like BERT, are used to extract embeddings, are less dependent on pre-processing and might even benefit from the variation. This is because the models have been trained on large corpus and already have good representations of all the words.

Table 6.1: Overview of datasets with different pre-processing applied.

Datasets	**Case folding**	**Removed punctuation**	**Removed stop words**	**Removed rare words**
Dataset A	✓	✓	✓	✓
Dataset B	✓	×	×	×

Bag-of-words is widely used as sentence representation along side with a baseline model. It is the simplest form for representing a sentence or document. Word2vec is another quite common and successful embedding technique used to represent both words and sentences, for instance in [69] and [43]. Doc2vec is however not that widely used after what we observed when we researched sentence representation. Still, doc2vec suited our need of representing utterances that consisted of one or more sentences. This, in comparison to word embeddings that only represents single words, or tf-idf that represents entire documents. BERT is again widely used and usually provides good results for NLP related tasks, as it did in [43].

6.5.1 Bag-of-words

We generated the BoW embeddings for each utterance using *scikit-learn* [28]. We created BoW embedding based on Dataset A. Each sentence was generated into a BoW represented as a vector with the same dimension as the vocabulary (section 3.3.4.1).

6.5.2 Word2vec average

We trained the word2vec embeddings on Dataset A using the *Gensim* NLP library [38]. We chose the algorithm, skip-gram, but left the remaining hyperparameters as default. The reason for choosing skip-gram was because the corpus was relatively small and we could therefore afford the heavier algorithm and increase the semantic capturing (section 3.3.3.4). We used the algorithm parameters; *epochs=50* and *min_count=3*. "epochs" are the number of iterations over the training data and "min_count" defines the threshold where words are ignored if they occur less often than the threshold. This is similar to the last normalization processes described in subsection 6.3.2. The algorithm was also given the parameters *workers=1* and *seed=42* for reproducibility.

After training the word2vec model, we calculated the average embedding of the word embeddings of each utterance. The average embedding was used as the sentence representation.

6.5.3 Doc2vec

For the doc2vec embeddings did we trained the model on Dataset A, also using the Gensim library [9]. Here we also left the hyperparameters as default. We choose the distributed memory algorithm as the paper states it is consistently better than distributed bag of words (section 3.3.4.2). We used the algorithm parameters; *epochs=50*, *min_count=3*, *workers=1* and *seed=42* here as well.

6.5.4 BERT classification tokens

We used a pre-trained version of BERT to create the embeddings of the utterances. The pre-trained model was made and distributed by *Nasjonalbiblioteket AI Lab* [23] [60], via *Huggingface*. Huggingface is an AI community which provides an API accessing pre-trained models in Python [2]. We used the *nb-bert-base* provided by Nasjonalbiblioteket AI Lab [20]. For this we used Dataset B and ran each utterance through the model and extracted the CLS token from the embedded sequence.

Since BERT is pre-trained there are embeddings for all of the subwords occurring in the dataset and therefore there is less need for pre-processing the data. Also, the information which is not removed in Dataset B may prove useful. This is different from the other sentence representation which were trained from scratch, and were more dependent on pre-processing. This is important for reduce the noise, variation and training time, as we did with Dataset∼A.

6.6 Generating sequential word embeddings

The sentence embeddings where also representatd as sequences of word embeddings. These sequences of word embeddeds was then fed to RNN and LSTM networks. To embed the words we used word2vec and BERT. We utilized the same models to generate the word embeddings, as we did for the word2vec average (section 6.5.2) and the CLS token generated by BERT (section 6.5.4).

6.6.1 Word2vec word embeddings

To generate the sequences of the word embeddings, we used Dataset A and the same model we used to generate the averaged word2vec embeddings. The sequence embedding is done in the exact same way as the word2vec average, but instead of computing and saving the averaged vector, the entire sequence of word embedding was saved. The embedded sequences was then of the dimension; (length of the sentence $\times$ 100).

6.6.2 BERT word embeddings

We used the same BERT model [23] as we did when extracting the classification tokens, to generate the sequences of word embeddings. The last hidden state generated by the model was stripped from the CLS (first) and SEP (last) tokens. The remaining vectors was saved as the sentence representation. Since BERT utilizes the WordPiece embedding were the embedded sequences usually longer than the utterances them self in terms of words. This slowed down the training of the recurrent networks since they had to process longer sequences (see section 2.3.2).

6.7 Training binary classification models

This section will cover the selection and training of the different compositions of machine learning algorithms and embedding techniques. Knowing the suboptimal quality of our dataset and challenges around the utterances, we concluded to simplify the classification problem. By simplifying the problem we could see if this would give any promising results, before attempting the complete problem. The problem was simplified into a binary classification: Should the utterances be handled by the dialogue system or by a human? The following domains were grouped to be handled by the dialogue system; *"Courses", "General", "Licenses", "IntensiveCourses", "Corona", "Payment", "OnlineCourse", "PackagePrice", "CustomerReturne", "Giftcard", "Finance", "DrivingTest", "Rudskogen/Førerutviklingskurs"*. The following domains should be handled by a human; *"Request", "Toss", "Complex", "Advice", "Other"*.

First, we defined a baseline model for the binary classification problem. For the baseline model we used logistic regression with bag-of-words as sentence representation. A baseline model is a simple model used to evaluate successive models as their complexity gradually increases. We chose to evaluate the models performance based on accuracy. We chose accuracy because it tells us overall how the binary classification model performed. Additionally it is simple and easy to understand. Accuracy made it easy to interpret the fundamental problem in this thesis.

After training the baseline model, we trained some additional models with logistic regression, random forest and FFNN for all the different sentence embeddings; bag-of word, word2vec average, doc2vec and BERTs classification token.

We chose logistic regression as the baseline model since it is a simple algorithm for making binary classification models. Random forest is also generally quite common and usually give good results. Still, random forest does not seem to be frequently used to solve NLP related tasks, after what we have observed. FFNNs however are quite common in NLP related tasks, and are especially often used for classification, for instance in [43]. Another common neural network architecture for NLP tasks are recurrent networks (see section 6.8), for instance simple RNNs, but especially LSTMs, which were used in both [69] and [43].

6.7.1 Logistic regression

As mentioned, the first model we trained was the baseline model, which was with logistic regression and bag-of-words as sentence representation. Bag-of-words is the simplest way of representing the utterances. We chose logistic regression because it is a simple algorithm for binary classification, originally with no hyperparameters. Another optional algorithm to create the baseline model could have been naïve Bayes, which is commonly used in similar problems, for example in [69] and [43].

Following the baseline model, we created additional models with logistic regression for word2vec average, doc2vec and BERT embeddings. Each model was trained on the training data and evaluated on the validation set.

6.7.2 Random forest

The second machine learning algorithm we used was random forest. We trained 30 different models for each embedding via random search. Random search picks a random values within a range of a hyperparameters. We searched the hyperparameters; *min_samples_leaf* within the range of 1-100, *max_features* between 10% and 100% and *max_depth* within the range of 5-50. We then selected the best model for each embedding based on the accuracy achieved on the validation set.

6.7.3 Feedforward neural network

The last machine learning algorithm we used for the fixed sentence representation was FFNNs. In FFNN, each network was constructed with one input layer, one hidden layer and one output layer. The size of the input layer depends on the dimension of the embedding, while the size of the output layer was two. We trained 30 feedforward neural networks for 30 epochs for each of the embeddings. We used random search for the number of hidden units and the learning rate *alpha*. For the number of hidden units we searched the range between the input and output size. For the learning rate we searched between 1 and 0.001 using the logarithmic function 10^x and picking a random value between 0 and -3. Finally, we selected the best performing model according to the accuracy score on the validation set for each embedding.

6.8 Training sequential binary classification models

We trained four different sequential binary classification models using the RNN and LSTM architectures combined with either word2vec or BERT word embeddings.

Training recurrent networks took significantly more time than training the other machine learning models. Because of this we only trained 10 models for 30 epochs for each the four models. We then selected the best models according to the accuracy scored on the validation set.

All of the sequential binary classification models, had one input layer, one hidden state layer and one output layer. The input size depended on the type of word embedding. For word2vec, the models had an input size of 100 dimensions, while the models using BERT had the size of 768 dimensions. The output layer was of size 2, while the size of the hidden layer was chosen using random search. We searched the range between the size of the input and output layer. Random search was also applied to choose the learning rate. The learning rate was selected using the logarithmic function 10^x, where x was a randomly chosen value between 0 and -3.

The sequential models needed some extra setup before training. To efficiently train them we had to transform each sequence into equally sized lengths. Once the samples were of the same length they could be stored and processed as a tensor. A tensor is a matrix with

three or more dimensions. This was done by padding each sequence with trailing "0"s. The network could then do computation on mini batches of tensors. Since each sample now were of the same length, the network needed to extract the correct hidden state according to the length of the sample. Once the hidden states were extracted could they be forwarded through the output layer.

Chapter 7

Results

In this section we present the results we got from the methods. We will cover the labeling and pre-processing of utterances, the exploratory data analysis, visualize sentence representation of the utterances with t-SNE and UMAP and finally the performance of the different combinations of sentence representations and machine learning algorithms.

7.1 Labeling

The labeling was quite challenging because of the complexity and variations of utterances. Some were easy and straight forward, like:

> *I would like a driving lesson.*

While others were more complex and could contain multiple domains and intents. For instance, the following example was labeled with the domains *Courses* and *License* and have the intents *getPrice* and *getAvailable*:

> *Hi, what is the price for Driving-at-dark course and when is the next available driving lesson?*

Another challenge was the personality characteristic in a lot of the messages. These variations do not provide any relevant information for the dialogue system and are more noise than they are helpful. Still, such characteristics display a very human way of communicating. See for example the complexity in the following message:

> *I've always wanted to drive a motorcycle since I was a child. However, I have children of my own now which makes it challenging. How flexible are the lessons? I took a motorcycle course a couple of years ago, is this still valid do you think? Also, I've been recommended Terje by a friend, is he available?*

Defining suitable labels for domains and intents was also challenging because of the levels of services the driving school provided. There are multiple types of licenses with different criteria regarding courses, experiences and evaluations. We ended up labeling 18 domains and 32 intents in the dataset. Some labels were initially added, but turned out not to be that relevant during the labeling process. Others were added along the way after observing the already labeled data and deciding that a domain or an intent was missing. We took some inspiration from the MultiWOZ paper [48] for the labeling approach. At the end we had hand-label 2,800 of the 4,948 utterances in the dataset.

7.2 Pre-processing

The 2,800 labeled utterances was split into a training set with 1,680 instances, a validation set with 560 instances and a testing set with 560 instances. When producing Dataset A, we performed case folding and removed both punctuation and stop words in the training set. This resulted in a vocabulary of 3,271 tokens. Then we removed rare words, meaning words that occurred less than three times (section 6.3.2), and this further reduced the vocabulary to 936 tokens.

7.3 Exploratory data analysis

In this subsection we present the graphs and observations we discovered through our exploratory data analysis. First, we will go through the simple descriptive statistics of dialogues in the corpus, including the frequencies of words and n-grams. Then we will look at the distribution of the hand-labeled classes.

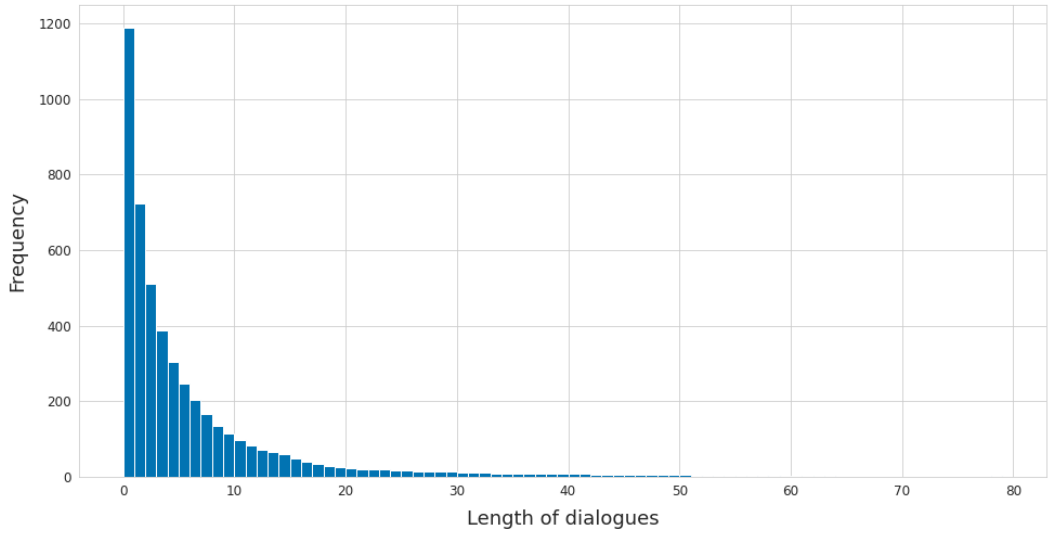

Figure 7.1: Histogram showing lengths of dialogues, number of messages sent by the users, provided in the dataset. The plot is right-skewed since longer dialogues happen less often.

7.3.1 Descriptive statistics

The length of each dialogue varied from one message to a maximum of 96. The histogram in Figure 7.1 shows the distribution of the lengths of dialogues. The average length of a dialogue consists of 7.5 messages. However, when the distribution is skewed, the median is a better estimation of central tendency, representing more of the majority of the messages. The median of the dialogue length was 4 messages [56, Ch3.5 p75].

The length of each utterance, in terms of words, varied from a minimum of one word to a maximum number of 1,451 words. The distribution is skewed here as well and has a mean of 84.9 words and a median of 56 words per utterance. See Figure 7.2 for the distribution og lengths of utterances.

7.3.1.1 Frequencies of words and phrases

We counted the frequency of each token and plotted 20 of the most common tokens. These 20 tokens are displayed in Figure 7.3.

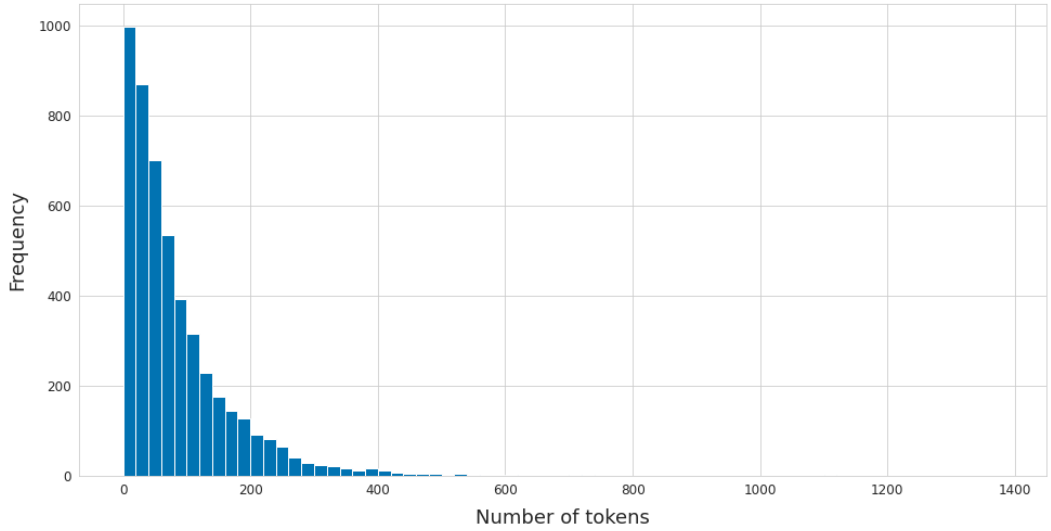

Figure 7.2: Histogram showing length of messages in terms of tokens/words provided in the dataset. The plot is right-skewed since longer messages are rarer than short ones.

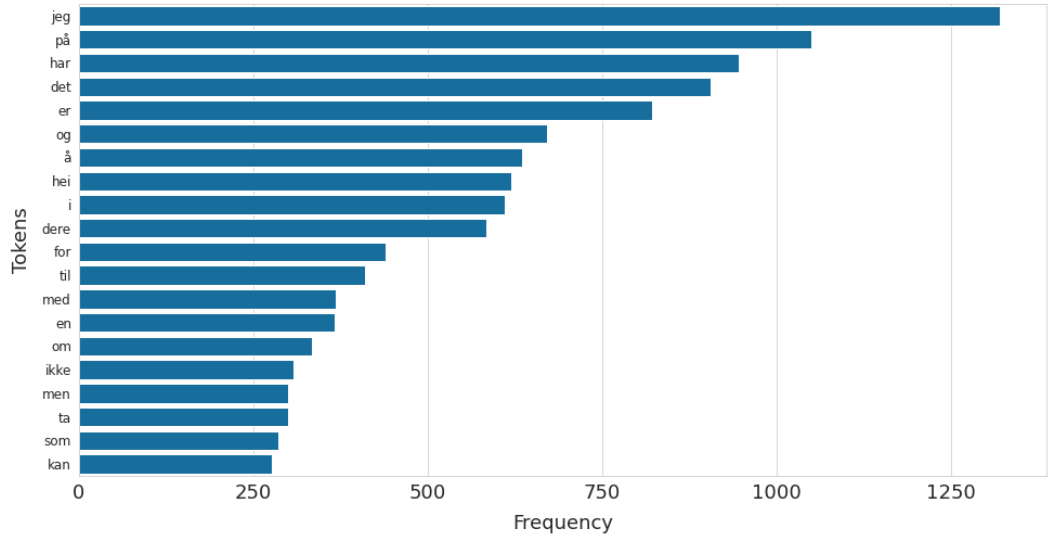

Figure 7.3: Histogram showing 20 of the most frequent tokens in our corpus.

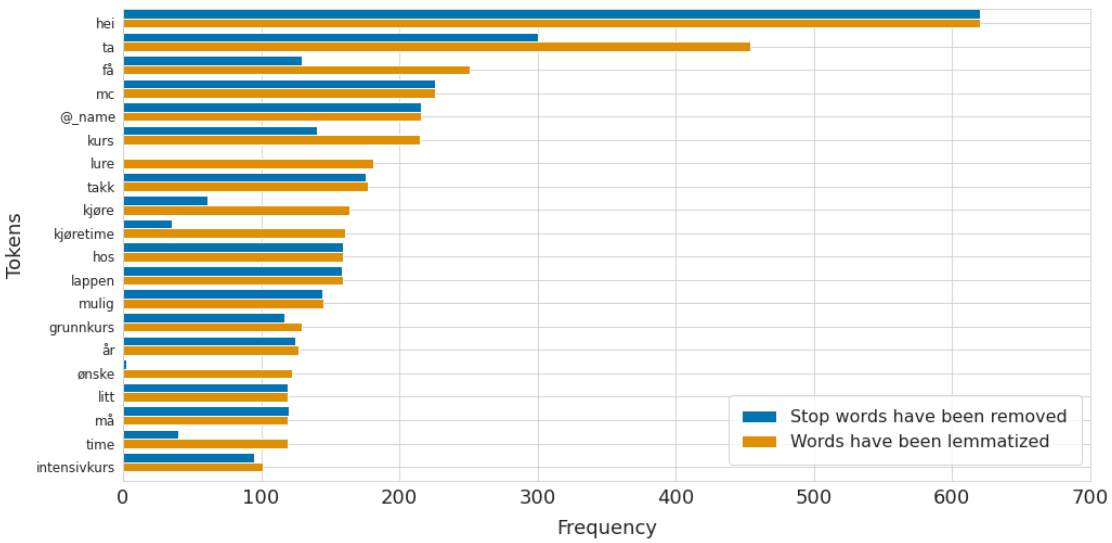

Figure 7.4: Chart showing the 20 most common words after removing stop words, and comparing the effect of lemmatization.

The tokens are quite common and do not capture any characteristic of the corpus. Just by looking at the most common tokens, the corpus could concern just about anything, and it certainly does not suggest references to a driving school's services. As observed in Figure 7.3, a simple word count provide little insight into our corpus. Figure 7.4 shows what words that are most common after we removed the stop words (blue bars). Additionally, we show the most common words after removing stop words and lemmatization (orange bars in Figure 7.4).

However, we had to manually adjust some of the mapping of words. For instance, spaCy converted "kurs" to "kurse". After manually finding and correcting 61 of such instances in various degrees. Lemmatization was only used for analyzing the corpus, and was not performed as a pre-processing step when we created the embeddings.

In Figure 7.4 we observe that the most common words now relate more to driving school activities. We can also see the effect of lemmatization. The most notable instance of lemmatization is "lure" which increased its frequency drastically. After lemmatization "lure" also captured the conjugations "lurte" and "lurer".

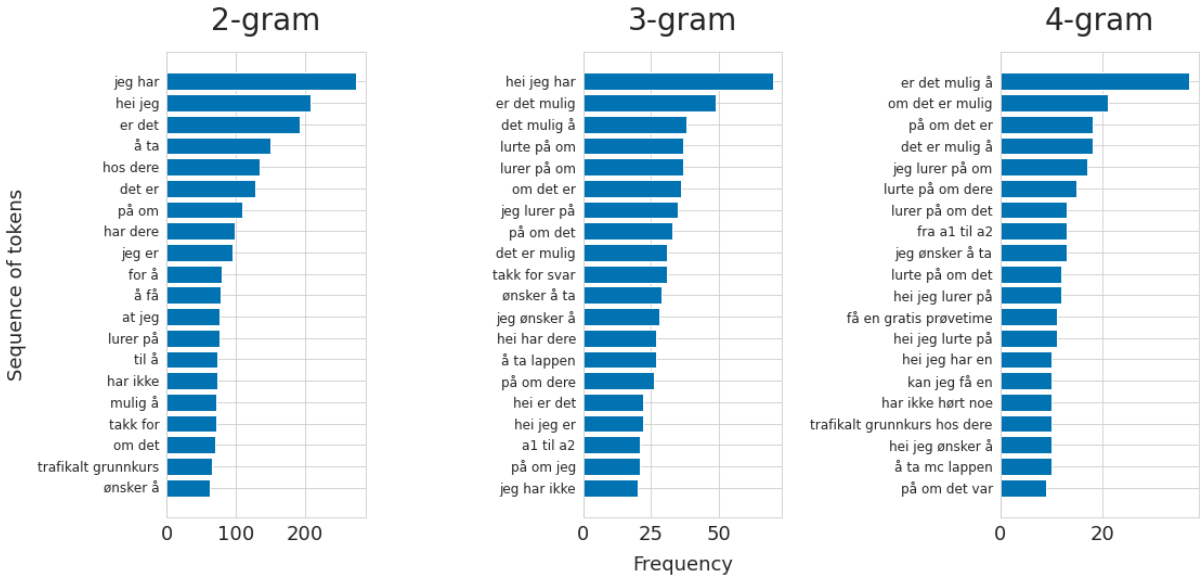

Figure 7.5: Chart showing most common 2-, 3- and 4-grams.

7.3.1.2 N-grams

We created n-grams of; two, three and four tokens, and analyzed the top 20 phrases (Figure 7.5). We kept the stop words when creating the n-grams to prevent inconsistent sequences.

The frequency of n-grams drops when n increases, and show that longer sequences are rarer. Studying the second graph in Figure 7.5, we observe that "lurte på om" and "lurer på om" are treated as two separate instances, and could be an argument for using lemmatization in some cases. There were some instances in the top 20 phrases that relate to driving school activities. However, most of the phrases are more general and service oriented. We therefore went a step further and removed n-grams where stop words appeared, see Figure 7.6.

In Figure 7.6 we can see which unique phrases are sent to the driving school. These phrases regard different licenses and related courses. Some of the masked-tags also appear here, for instance "@_name" and "@_phone".

7.3.2 Label distribution

The label distribution of classes over domains are shown in Figure 7.7. "Complex", "License" and "General" are the top three most common classes. "Finance", "Giftcard" and "Corona" are the least common classes.

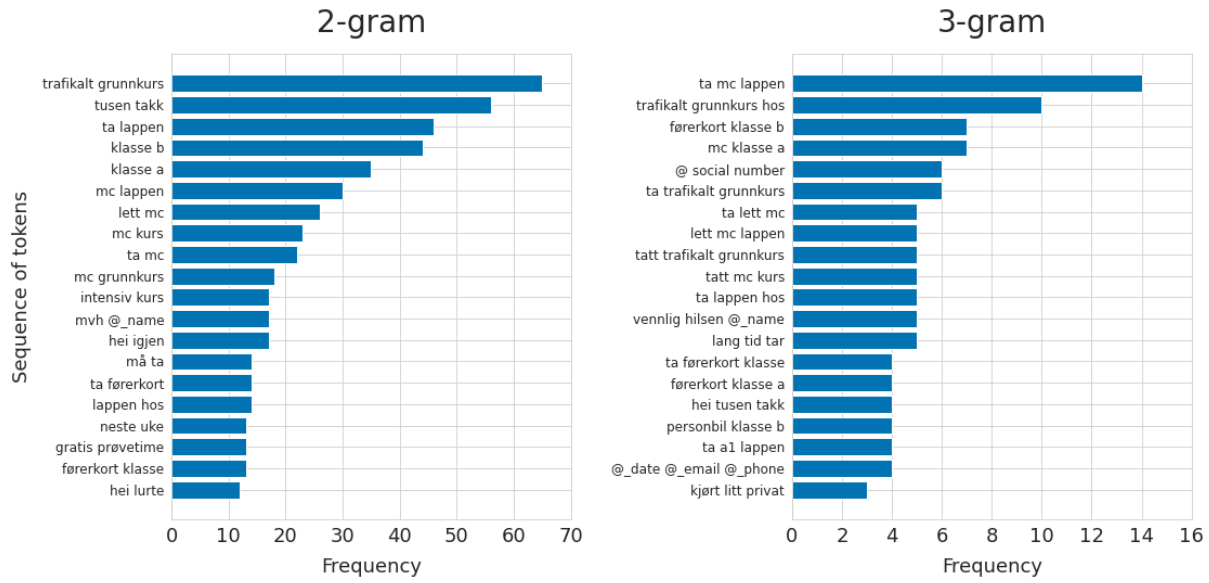

Figure 7.6: Chart showing the most common 2- and 3-grams after removing n-grams containing stop words.

Each domain has a selection of intents. All the different intents and their distribution is shown in figure 7.8. Some utterances were to complex and were just given a domain that indicated that they should be handled by humans. “Complex” and “Request” are domains where this applies. Samples within these domains were usually not given an intent. We gave samples with missing intents the label “human”. There are 794 instances labeled with “human”. “getAvailable”, “endOfConversation” and “confirm” are the most common classes of intents.

As mentioned, we simplified the classification problem into a binary problem, dividing the domain labels into two labels; “chatbot” (handled by the dialogue system) and “human” (handled by a human). There were 987 (58.8%) utterances labeled as “chatbot” and 692 (41.2%) utterances labeled as “human” in the training set. The distribution is shown in Figure 7.9.

7.4 Creating sentence representations

In this section we present the created sentence representations via visualization. We used t-SNE and UMAP to project the embeddings into 2D space. As mentioned in section 6.5

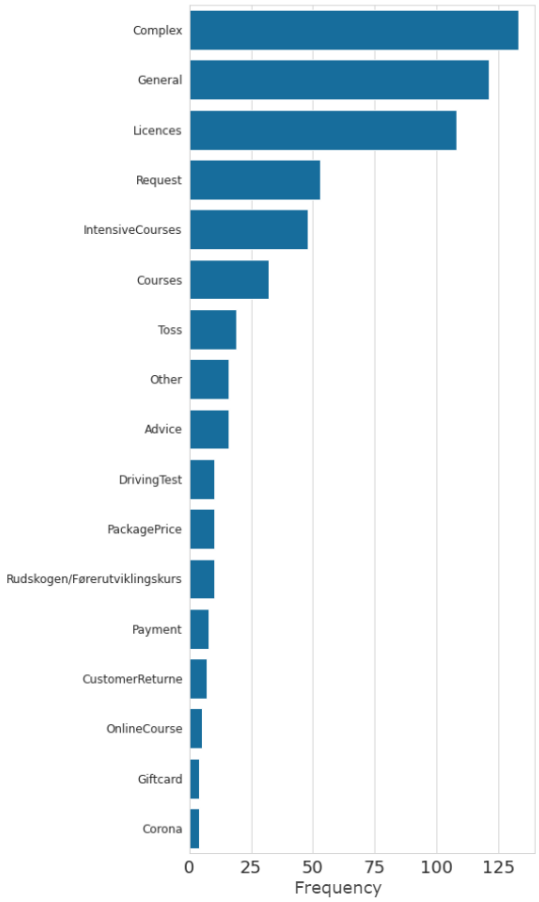

Figure 7.7: Histogram of domains in the dataset.

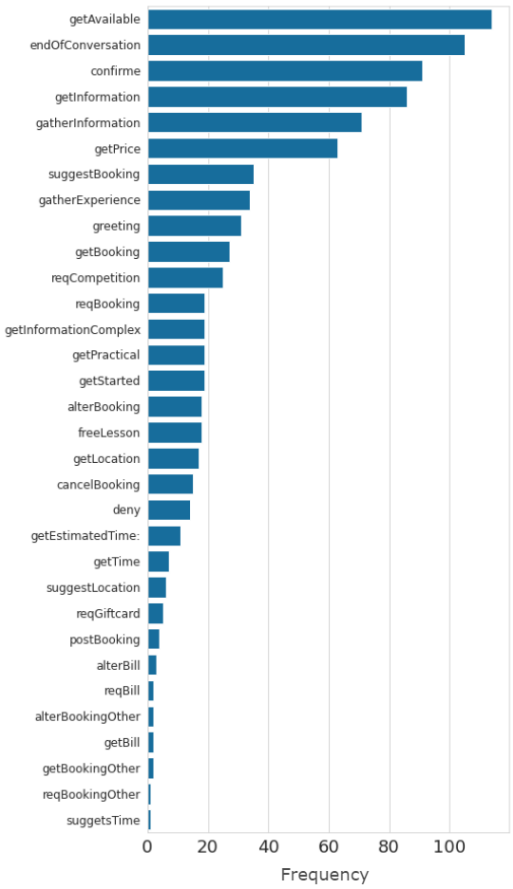

Figure 7.8: Histogram of intents in the dataset.

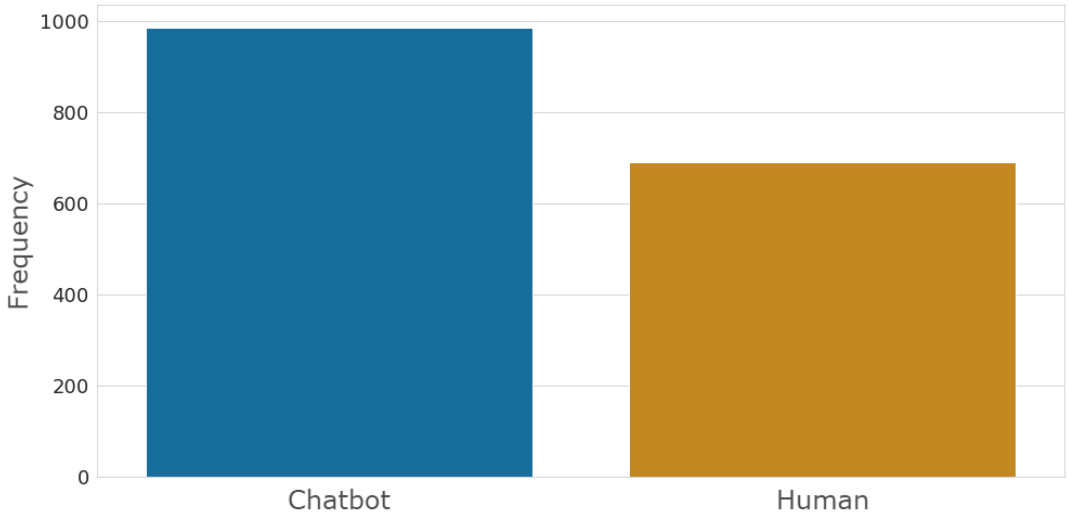

Figure 7.9: A barplot showing the distribution between two labels; “Chatbot” and “Human”.

the axes of both t-SNE and UMAP do not carry any meaning. They are only useful for visualization. In each plot we display the 5 most common classes for both the domains (left plot) and intents (right plot). The reminding labels are grouped under a unified label; “Other labels”.

7.4.1 Bag-of-words

The first embedding we visualize is BoW. The initial dimension of the generated bag-of-words was 3,271. However, 1,871 (57%) of the dimensions represented single occurrence of words in the training corpus. We therefore removed words that occurred less then three times. After removing rare words we had reduced the dimension down to 936. Figure 7.10 displays each of the utterance’s representation as a BoW mapped into 2D-space. The 2D representation is generated via t-SNE (section 2.4.1).

There appears to be some small clusters at (35, 0) in Figure 7.10. These corresponds to utterances such as; “Ok, takk”, “Supert, takk” and “Oki, takk for hjelpen”. There are no dense areas in terms of labels except for the green areas in the domain plot. The green area are located around coordinate (35, 0). Figure 7.11 shows the projection of BoW with UMAP.

Figure 7.10: Two scatter plots of the same utterances represented as a bag-of-words, mapped into 2D-space using t-SNE. The left plot is labeled with domain classes, and the right plot is labeled with intent classes.

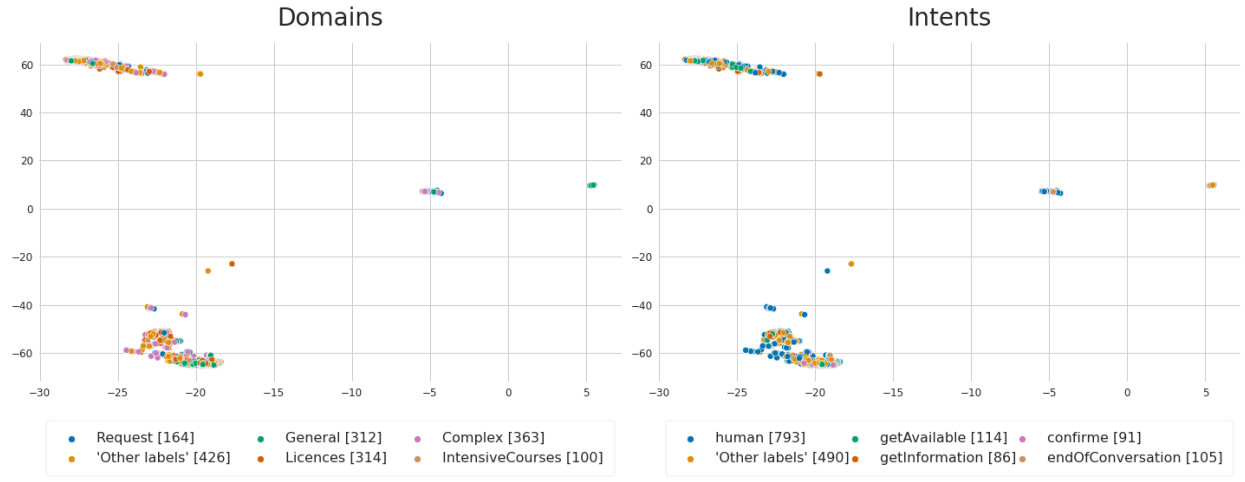

Figure 7.11: Two scatter plots of the same utterances represented as a bag-of-words, mapped into 2D-space using UMAP. The left plot is labeled with domain classes, and the right plot is labeled with intent classes.

The UMAP-plots show a few more distinct clusters, however they do not correspond with the hand-labeled classes in neither the domain or intent plot. In the following we will refer to the domain plot. The wide cluster at (-25, 60) is a collection of varying utterances, but had in common that they were opening messages in the conversation. For example; "Hei, når er neste kurs?" and "Hei, jeg vil ta lappen". The large cluster at (-22, 60) varied alot as well, but were utterances providing or asking for information. The cluster at (-5, 5) were utterances containing "Tusen takk", while the cluster at (5, 5) were utterances like; "Takk" and "Supert, takk for det".

7.4.2 Word2vec average

In this subsection we visualize the averaged word2vec embeddings using t-SNE and UMAP. The samples are projected from the original 100 dimensions down to two dimensions. First we will look at the t-SNE plot.

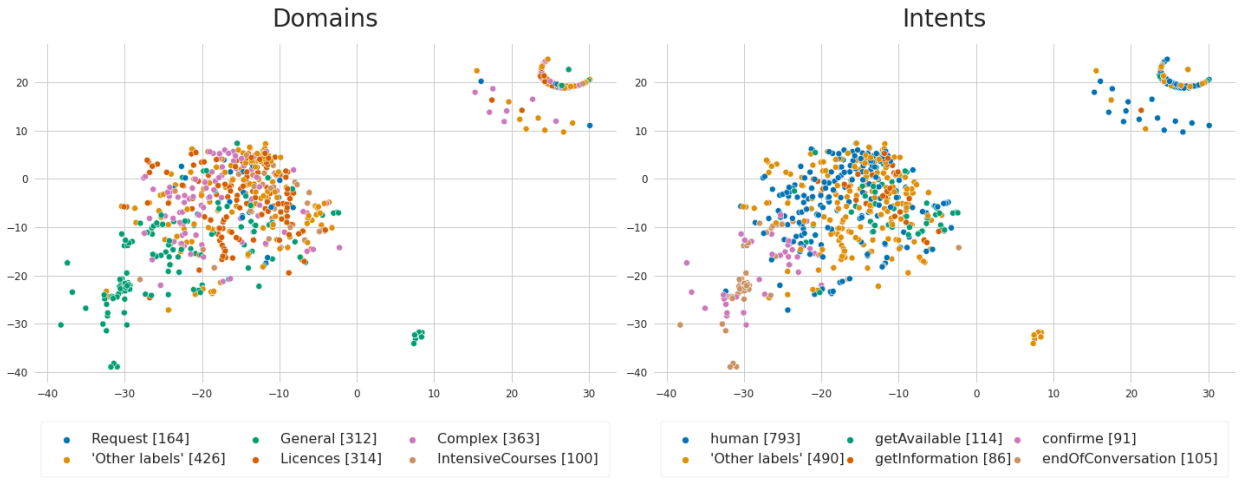

Figure 7.12: Two scatter plots of the same utterances represented as the average of word2vec embeddings, mapped into 2D-space using t-SNE. The left plot is labeled with domain classes, and the right plot is labeled with intent classes.

The t-SNE plot in Figure 7.12 show three distinct clusters. Referring to the domain plot, the largest cluster at (-10, -20) has a clear dense area corresponding to some of the domains. The domains "General" (green) and "licenses" (orange) have a noticeable boundary between them. The green area were utterances of "Oki takk", "Ja" and "Den er grei". The orange area had a variation of driving license related utterances. For instance, the orange line at

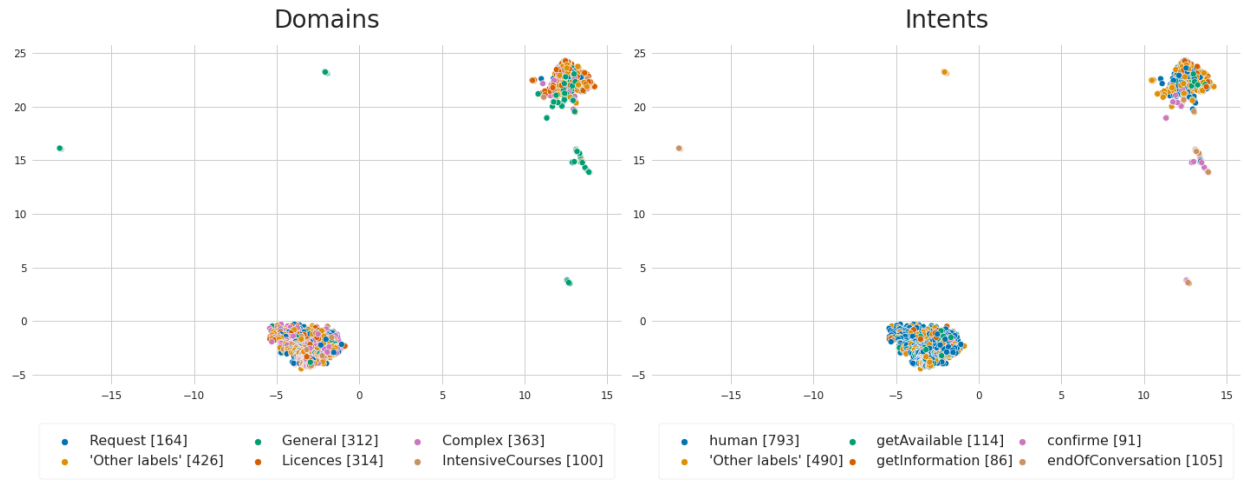

Figure 7.13: Two scatter plots of the same utterances represented as the average of word2vec embeddings, mapped into 2D-space using UMAP. The left plot is labeled with domain classes, and the right plot is labeled with intent classes

(-18, -12) were experiences related, for example; “Har ikke kjørt før” and “Jeg har kjørt en god del”. While the orange field at (-9, -6) were motorcycle related, containing keywords like; “MC”, “A1” and “A2”. The cluster at (8, -32) were utterances of greetings; “Hei” and “Heisann”. The strange ark at (25, 20) had a high variance of utterances. Their coherence seems to be that they contained dates and time expressed with numbers. We present the UMAP plot of word2vec average in Figure 7.13.

UMAP projected the utterances into two distinct clusters. There were not observed any remarkable differences in terms of the content of utterances between the clusters. However, the cluster at (13, 23) had much simpler (short and precise) utterances and had some internal groupings of utterances. For instance, the orange cluster of “licenses” at (11,23) in the domain plot, contains utterances asking for free lessons; “Kan jeg få en gratis kjøretime” and “Jeg ønsker en gratis prøvetime”. While the area around (14, 22) had utterances asking about prices. The second cluster at (-4, -3) were highly variated and showed no internal grouping of the utterances.

There are four smaller cluster as well. The utterances at (-18, 16) were all the same; “takk”. At (-2, 23) the utterances were also the same; “hei”. The cluster at (13, 4) were utterances of gratitude; “tusen takk” and “flott takk”. And the last cluster at (13, 15) was also utterances of gratitude; “Okei, takk”, “Supert!” and “Mange takk”.

7.4.3 Doc2vec

In this section we visualize the doc2vec embeddings. Unfortunately, doc2vec turned out to be difficult to reproduce [25]. We did our best to make it as consistent as possible. The samples are projected from the original 100 dimensions down to two dimensions.

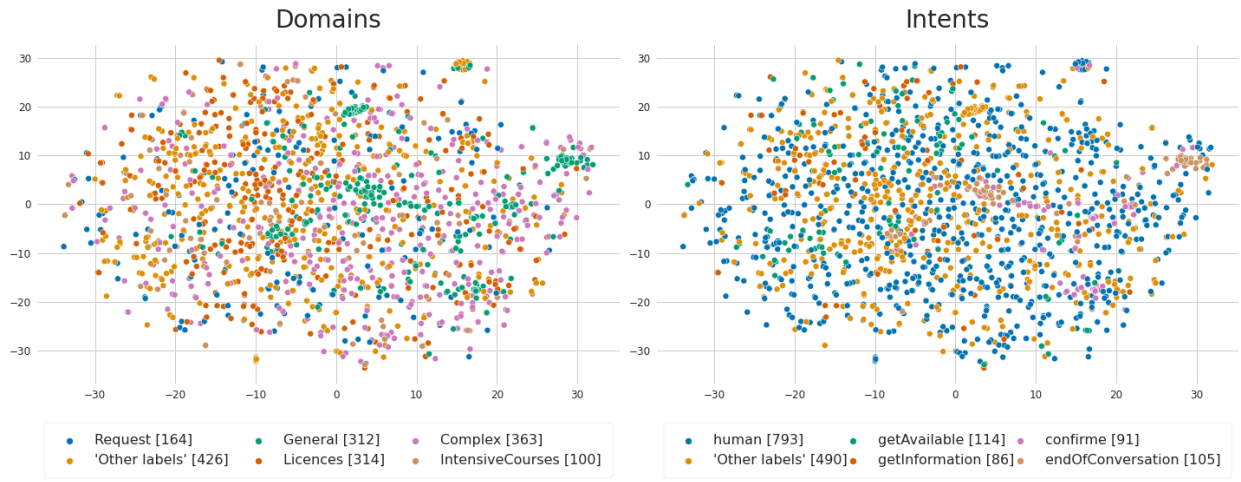

Figure 7.14: Two scatter plots of the same utterances represented via doc2vec embeddings, mapped into 2D-space using t-SNE. The left plot is labeled with domain classes, and the right plot is labeled with intent classes.

In Figure 7.14, t-SNE projects the utterances into a relative even distribution with no distinct clusters, except for the circular cluster at (14, 29). This cluster are utterances with rare words or punctuation which have been removed, hence they are actually empty. There are some dense spots of different classes, mostly visible in the domain plot as green areas. These areas are the same as we have mentioned earlier, consisting of utterances like: “Takk”, “Hei” and “Tusen takk”. Besides these, there were no obvious areas of correlated utterances. The visualization with UMAP is shown in Figure 7.15.

In the UMAP plot, the same green areas in the domain plot appears at (6.5, 9) “Hei”, (8, 5) “Tusen takk” and (9, 3) “Takk”. The cluster at (6, 1) was mostly empty utterances, similar to the cluster we pointed out in the previous section. The areas around (11.5, 6) were of utterances carrying confirmation. For example; “Ja, det har jeg”, “Okey, da søker jeg om det” and “Ja, A2 til A”. Besides this, the distribution was quite varying.

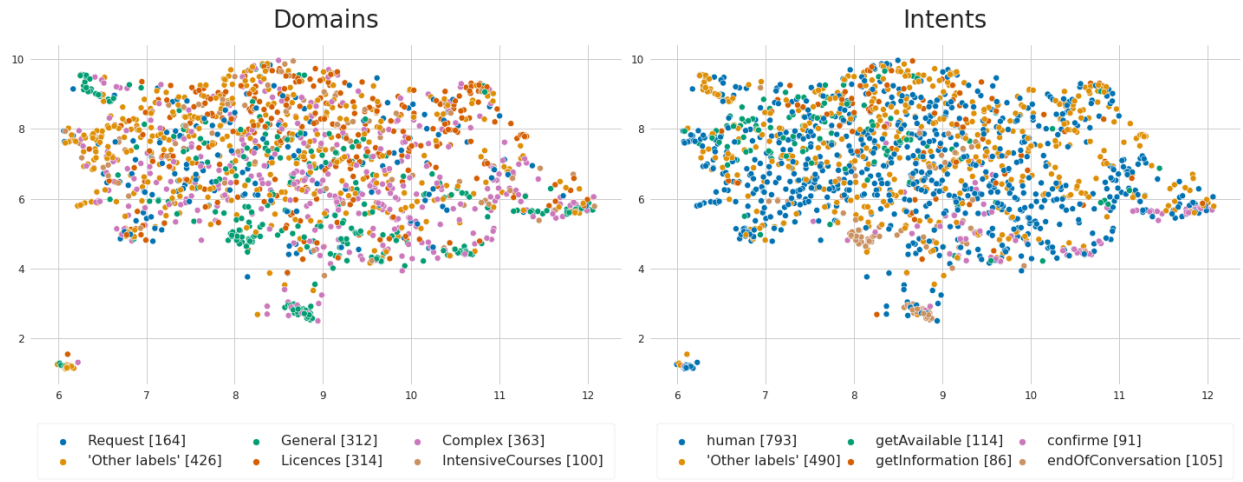

Figure 7.15: Two scatter plots of the same utterances represented via doc2vec embeddings, mapped into 2D-space using UMAP. The left plot is labeled with domain classes, right plot is labeled with intent classes.

7.4.4 BERT

The last embedding we are going to visualize is the embeddings made form BERT. The samples are projected from the original 768 dimensions down to two dimensions. We will start off with t-SNE.

There appears to be some patterns separating the clusters to a certain degree in Figure 7.16. The utterances in the upper half (above y equals 0) were simpler, shorter and more precise, than those found in the lower half. The clusters appearing at (28, 50) and around (26, 14) are utterances expresses gratitude. Interestingly enough, the instances around (-20, 0) are also utterances of gratitude, but with emoticons like; ":)" and ":D". Utterances at (32, 22) are greetings, while those around (10, 35) consists of confirmations and rejections; "Den er god" and "Nei, har ikke det". Still, there was a very high variation in the utterances when inspecting the plot closer. The visualization with UMAP is shown in Figure 7.17.

UMAP projected the utterances into mainly three clusters. The wide cluster at (12, 13) contained varying and complex utterances. Utterances appearing at (6, 11) also contained mostly varying, long, challenging requests form users. However, there were some simpler, first-line requests in the left half of this cluster. The cluster at (11, 4) had simple, short and precise requests, and showed some internal groupings of the utterances. Utterances around (9, 2) reflected confirmations and gratitude. The area around (10, 4) expressed experience

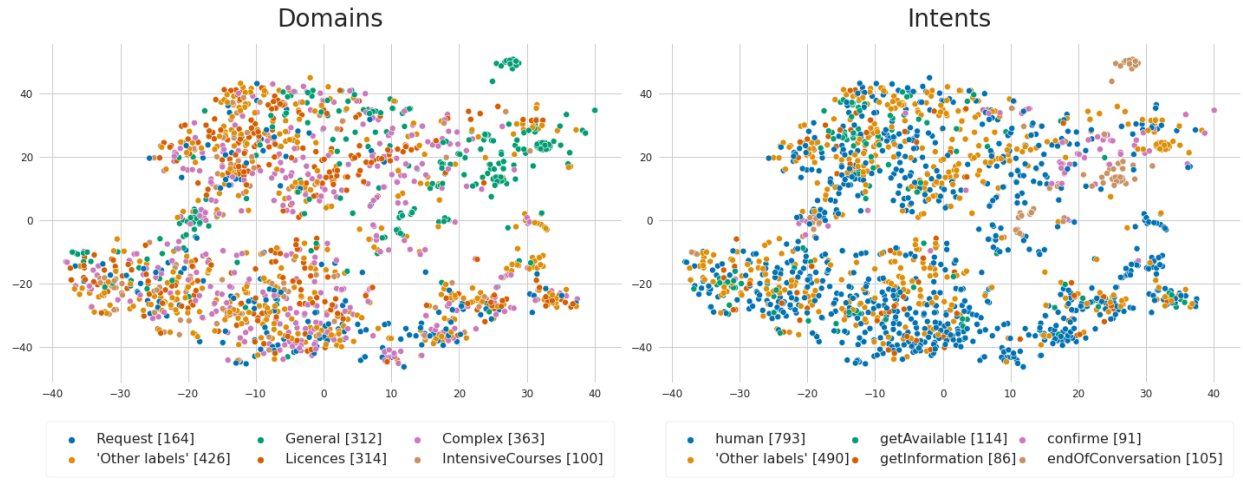

Figure 7.16: Two scatter plots of the same utterances represented via BERT embeddings, mapped into 2D-space using t-SNE. The left plot is labeled with domain classes, and the right plot is labeled with intent classes.

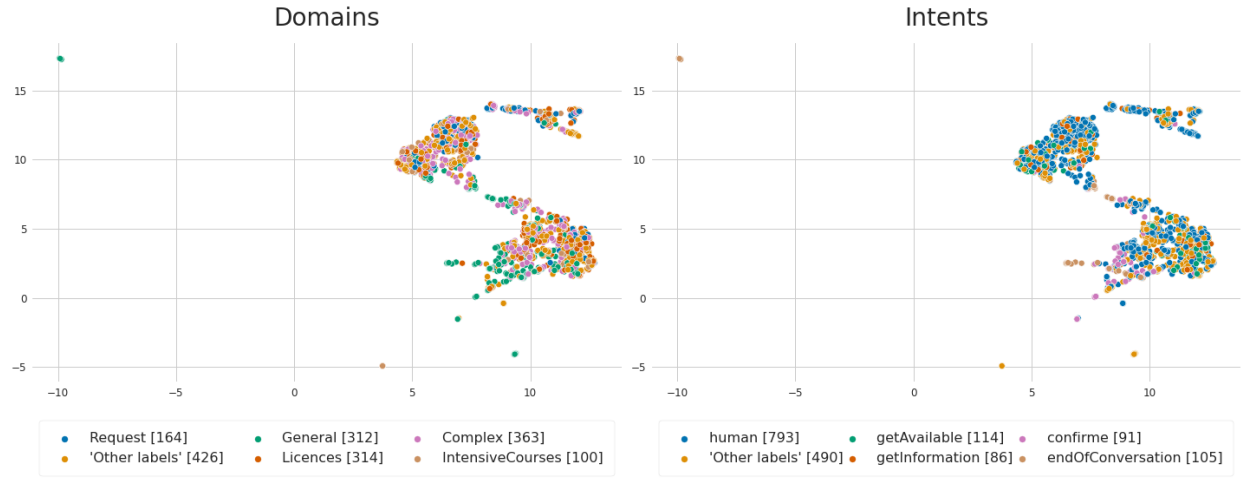

Figure 7.17: Two scatter plots of the same utterances represented via BERT embeddings, mapped into 2D-space using UMAP. The left plot is labeled with domain classes, and the right plot is labeled with intent classes.

and requirements from the user. Utterances around (12, 3) seemed to concern to course and motorcycle related questions.

The small cluster at (-10, 17) consisted of; “takk”, while the one at (9, -4) were greetings; “Hei” and “Hallo”. The passage at (7.5, 8.5) was combinations of confirmations and gratitude; “Flott, takk for svar” and “Supert, takk for hjelpen”. While utterances at (9.5, 7) was utterances counting emoticons; “:)”, and were also mostly confirmations and expressions of gratitude.

7.5 Trained binary classification models

As we mentioned in section 6.7, knowing the complexity and challenges around our utterances, limited data and quality of the labels, we choose to first simplify the the classification problem. Additionally, we did not observe any groupings of our labeled data in the plots we showed in the previous sections. We divided the dataset according to the domain labels since we already filtered out challenging utterances with specific labels (see section 6.7).

7.5.1 Baseline model and logistic regression models

The first model we trained was the baseline model, which was with logistic regression and bag-of-words as the baseline model. The baseline model scored an accuracy of 68.4% on the validation set. After the baseline model, we a trained logistic regression model for each of the remaining embeddings. The model trained with word2vec average achieved 64.3%, the model trained with doc2vec achieved 63.0%, and the one with BERT’s classifications tokens got 71.3%.

7.5.2 Random forest models

The second machine learning algorithm we used was random forest. We trained 30 models for each of the different embeddings via random search. We selected the best preforming model for each embedding and measured their performance on the validation set. The model trained for bag-of-words scored 64.1%, the model trained with word2vec average achieved 71.2%, doc2vec 67.7% and the one with BERT’s classification tokens scored 69.6%. Their corresponding hyperparameters found via random search are listed in Table 7.1.

Table 7.1: Showcase of four different random forest models and their hyperparameters trained on different sentence representations.

Embeddings	**min_samples_leaf**	**max_features**	**max_depth**
Bow	4	0.394	17
Word2vec average	16	0.614	22
Doc2vec	4	0.499	29
BERT	10	0.1	9

7.5.3 Feedforward neural networks

The last models we trained for the fixed sentence representation was via FFNN. For each of the embeddings we trained 30 FFNNs for 30 epochs using random search. Each model was selected based on the validation score. The network for the bag-of-words embeddings achived 72.3% while the one for word2vec average got 67.7%. doc2vec scored 67.3% and BERT embeddings 73.9%. The hyperparameters we found for each of the networks are listed in Table 7.2.

Table 7.2: Overview of hyperparameters corresponding to best preforming FFNN respectively to embeddings.

Embeddings	**alpha**	**hidden_units**	**epochs**
Bow	0.464	792	18
Word2vec average	0.462	23	26
Doc2vec	0.046	67	16
BERT	0.046	257	29

7.5.4 Summarized performance of the binary classification models

We summarized the validation score of all the binary classification models in Table 7.3. The scores correspond to the models trained with the different machine learning algorithms on the different embeddings.

Table 7.3: Overview of validation accuracy for each model. The models are trained with either logistic regression, random forest or feedforward networks. They are also trained using either bag-of-words, word2vec average, doc2vec or BERT embeddings of utterances.

Algorithm	**BoW**	**Word2vec average**	**Doc2vec**	**BERT**
Logistc regression	0.684	0.643	0.630	0.713
Random forest	0.641	0.718	0.677	0.696
Feedforward network	0.723	0.677	0.673	**0.739**

By comparing the machine learning algorithms, random forest performed better overall than logistic regression. FFNN did again in total perform slightly better than random forest. Overall, regarding the embeddings, the doc2vec representation had the worst performance. Bag-of-words was slightly better than word2vec average, while BERT's classifications tokens performed overall the best.

The model who achieved the highest accuracy score for this binary classification problem was the one combining FFNN with BERT embeddings. By using an FFNN with BERT embedding the performance was increased by 5.5% from the baseline accuracy score of 68.4% to 73.9%.

7.6 Sequential binary classification models

In this section we will present the four sequential binary classification models trained using the RNN and LSTM architecture with word2vec and BERT word embeddings.

7.6.1 Recurrent neural networks

We trained two RNN models, one with the word2vec embeddings and one for the BERT embeddings. The RNN with word2vec achieved an of 65.9% accuracy on the validation set. The RNN model trained for the BERT embeddings scored an accuracy of 68.6%. The hyperparameters for each model is listed in Table 7.4.

Table 7.4: Overview of recurrent neural network hyperparameters corresponding to the best preforming models using the word embeddings: word2vec and BERT.

Word embeddings	**alpha**	**hidden_units**	**epochs**
Word2vec	0.1748	59	7
BERT	0.0265	141	15

7.6.2 Long short-term memory

Using the LSTM architecture, the model trained on the word2vec embeddings achieved an accuracy of 70.4%, while the one trained on the BERT embeddings scored 70.0%. The hyperparameters for each of the LSTM models are shown in Table 7.5.

Table 7.5: Overview of hyperparameters corresponding to the best preforming LSTM with the word embeddings generated form word2vec and BERT.

Word embeddings	**alpha**	**hidden_units**	**epochs**
Word2vec	0.1748	59	7
BERT	0.0265	141	15

7.6.3 Summarized performance of the sequential binary classification models

As with the binary classification models, the accuracy scores from each model trained on the sequential embeddings are listed in Table 7.6. Each score corresponds with a neural network architecture and a word embedding technique.

Table 7.6: Overview of validation accuracy for RNN and LSTM networks, using word2vec and BERT word embeddings.

Algorithm	**Word2vec word embeddings**	**BERT word embeddings**
RNN	0.659	0.688
LSTM	**0.704**	0.700

RNN combined with word2vec embeddings performed the most poorly. Changing out the embedding increased the accuracy score to 68.8%, while switching to LSTM increased the score with word2vec to 70.4%. LSTM with word2vec embeddings achieved the highest score of the recurrent networks. LSTM combined with BERT had only barley worse results than LSTM with word2vec.

7.7 Summary of all the models

The recurrent models took significantly longer to train then the models with fixed sized input, and especially with the word embeddings generated with BERT.

We trained in total 16 different models, combining different machine learning algorithms and embeddings techniques. Four of the models were also recurrent neural networks.

The first model we trained, the baseline model, scored an accuracy of 68.4%. There were 7 models that had a lower accuracy score than the baseline model. The worst of them, logistic regression with doc2vec, had an accuracy score of 63.0%.

The worst preforming reccurent network, RNN with word2vec, scored 65.9% which was also below the baseline model. The best preforming recurrent model, LSTM with word2vec, achieved 70,4% which was better than the baseline model, but was still beaten by three models trained on sentence representations.

The model that achieved the highest accuracy score, was the feedforward network trained on classification tokens generated by BERT. It managed to score 73.9%, which was an improvement of 5.5% from the baseline model. We evaluated the final model on the testing set, of which it scored an accuracy of 73.2%. The final model also had an F1 score of 0.639 which we will compare with the F1 score of the related work in section 8.3.

7.8 Summary of results

We tested out the following sentence representations: bag-of-words, average of word2vec embeddings, doc2vec embedding, BERT's classification token, sequence of word2vec embeddigs and sequence of BERT's word embeddings. We used the following machine learning algorithms to train our models: Logistic regression, random forrest, feedforward neural network, recurrent neural network and long short-term memory.

The utterances in the dataset had high variation, were inconsistent and complex. Hence we choose to train binary classification models first, observe how they preform, before potentially expend the classification problem. We used accuracy to measure the performance of the models. The binary data distribution was 58.8%/41.2%.

We trained first an baseline model using logistic regression with bag-of-words embeddings. The baseline model scored an accuracy of 68.4%. The best performing model was an feedforward neural network combined with BERT's classifiaction tokens. It achieved an accuracy of 73.9%. The best preforming recurrent model was an long short-term memory model with word2vec embeddings, which scored 70.4%. The worst preforming model wast logistic regression with doc2vec embedding, and it only had an accuracy of 63.0%.

Due to the performances we did not advance with the classification problem. We concluded the models were not preforming remarkable, because of the quality of the data. However, we observed that increasing the complexity of both the machine learning algorithms and sentence representation made some performance improvements.

Lastly, we evaluated the finale model, FFNN with BERT classification tokens, on the test set. The model scored an accuracy of 73.21% and F1 score of 0.639.

Chapter 8

Discussion

In this chapter we will discuss the processing and results of the data labeling, sentence representation and the machine learning models considered in this thesis. We set out to explore which combination of sentence representation combined with a machine learning algorithm would perform the best on the given dataset.

8.1 Data, labeling and quality

The data labeling was a very time-consuming process. We managed to label 2800 of the 4948 available utterances due to time constraint. This left us with a relatively small dataset for the machine learning models to train on. Another challenge with the data labeling was some inconstency in the actual labeling. The reasons for this were complex; continuous updates of the labeling guidelines based on the already observed data, varying conception of which label the data should be distributed, and human error. It was challenging to label the data, mostly due to the amount of different information in the utterances. As mentioned in section 6.4.1, the average number of tokens in an utterance was 84.9. To put this into perspective, we compared our dataset with two benchmark datasets; *The Schema-Guided Dialogue Dataset* (SGT) [68] and *Multi-DomainWizard-of-Oz* (MultiWoZ) [48]. We also compared our dataset to the dataset in a similar master thesis; *Exploring pretrained word embeddings for multi-class text classification in Norwegian* [43] from Norges teknisk-naturvitenskapelige universitet. The comparision is visualized in Table 8.1.

Table 8.1: Comparison of our dataset with the two benchmark datasets; The Schema-Guided Dialogue Dataset (SGT) and Multi-Domain Wizard-of-Oz (MultiWOZ), and a dataset used in a similar master thesis from Norges teknisk-naturvitenskapelige universitet (NTNU).

***** Contains only one side of the dialogue

****** Contains typos

Compairing datasets				
Metric	**Our**	**NTNU**	**MultiWOZ**	**SGT**
# dialogues	1,219	NA	8,438	16,142
Total # of utterances	2,800	193,000	113,556	329,964
Avg. turns per dialogues	7.5*	NA	13.46	20.44
Avg. length of utterances	84.9	9.6	13.13	9.75
Total unique tokens	6,638**	8,790	23,689	30,352

There are two things that are worth noticing in the comparison of the datasets. Firstly, our dataset only has a fraction of total utterances compared to the others. Secondly, the average length of the utterances are notably higher than the others. The length of the utterances indicates that there is more information in each utterance. However, the higher amount of information would often be a mixture of multiple intents in each utterance, and just more noise or irrelevant information. We pointed out that both of these instances were occurring in section 7.1. This makes the data both harder to label reassuringly, and harder to process for the machine learning algorithms. This problem was also pointed out by a Rasa employee answering a question at their forum [19];

> *"it's impossible to build a bot that can handle very long user inputs simply because NLU model will fail to classify them correctly. The user inputs you train your assistant to understand should be short and correspond to a specific intent label which is impossible to achieve with very long inputs."*

Both Google's Dialogflow and Rasa therefore recommend to keep the intent's training data distinct from each other to enable the dialogue system to properly be able to distinguish utterances in production [14] [1].

In retrospect we could have handled the data differently. Firstly, we could have filtered away complex utterances by limiting the maximum length of the utterances with a certain

threshold. We could also have spent more time defining the guidelines for labeling the data. In addition, we could have increased the data quality by having multiple people label the data simultaneously and overlapping, detecting and correcting inconsistent labeling. For instance, in the dataset used in Intent Classification for Dialogue Utterances, one of the corpuses were labeled by 5 different workers to ensure data quality [46, Ch5.2 p177].

Lastly, we could have spent more time iterating the dataset, correcting mislabelings and applying changes to the labeling according to the final, updated guidelines. However, this work is immensely demanding in terms of resources and time, and there would therefore have been even less labeled data if we were to complete the thesis in the same amount of time. Furthermore, the removal of more complex utterances would have given us less data to work on, especially since long utterances were so common in our dataset, compared to the dataset from NTNU, MultiWOZ and SGT (Table 7.1).

8.2 Sentence representation

We either generated the representations (bag-of-words), train and mapped the sentence representations (word2vec and doc2vec) or mapped/extracted them via a pretrained model (BERT). We did not perform any form of hyperparameter tuning when we trained the embeddings. We used some recommended settings and left the rest as default. This was mostly because evaluating embeddings on their own is difficult. The most important evaluation metric for embeddings is done via machine learning tasks, by observing whether or not the embeddings improve the performance of the tasks [52, Ch6.12 p128].

We visualized the different embeddings and observed some patterns, or clusters, with t-SNE and UMAP. There were also a few dense areas corresponding with the hand-labeled classes. We observed some of the patterns and found a few correlations between them, but the distribution of the utterances was still highly varying.

The performance of the different sentence representations was surprising. Bag-of-words would outperform averaged word2vec, doc2vec and the sequence representation with word2vec, accross many machine learning algorithms. This might indicate that word2vec failed to capture a mapping of semantic representation of words, due to the small size of the corpora. In that case, the same possible reason for under-performing applies to doc2vec as well, since the algorithms doc2vec uses are very similar to word2vec's algorithms. However, the sentence representation created with BERT did out-perform BoW.

8.3 Machine learning models

The last part of our discussion revolves around the different machine learning algorithms we experimented with. We had two variations of machine learning models, for the first variation the sentence representation was fixed sized, while for the second variation the sentence representation was sequences of word embeddings. We will begin discussing the fixed sized variation.

As we mentioned in section 7.6, the overall performance rose as we used more complex models. We experimented with a few different hyperparameters for both random forest and FFNN. We could have tried out even more or different hyperparameters, or different ranges for our chosen hyperparameter. However, we kept both the classification problem and models as simple as possible without any remarkable success.

Network complexity seem to matter for the recurrent models as well. LSTM performed better than regular RNN. LSTM did however not outperform the FFNN model. Unfortunately, the recurrent networks were slow to train and we were only able to explore a few different variations.

We compared our final model with the best performing models presented in Related work ([69] and [43]). The comparison is presented in Table 8.2.

Table 8.2: Comparison of our models F1 score with the model in Exploring pretrained word embeddings for multi-class text classification in Norwegian and the model in Intent Classification for Dialogue Utterances.

Compairing datasets			
Model	**# utterances**	**# labels**	**F1**
SVM w/FastText average [69]	496	13	0.782
FFNN w/BERT [43]	193,000	1,716	0.886
FFNN w/BERT (Our)	2,800	2	0.639

The two models we compared our model with are notably better when comparing F1 score. Additionally, our model only distinguished between 2 labels, which is easier than the 13 labels and 1,716 labels the two models classified. The model in [69] was also based on quite a lot fewer utterances than our model. This indicates that our data was not optimal. Unfortunately, the average length of utterances was not reported in [46], so we could not

compare it to the average length in our dataset. Still, we have an indication that the utterances were on average shorter than in our dataset, as the intents were described to be disjoint. This in comparison to our data, where each utterance could contain multiple classes.

8.4 Summary

The best performing machine learning algorithms were FFNN and LSTM, while the best sentence representation was BERT's classification token. The model which performed the best was FFNN combined with BERT's classification token, and scored a validation accuracy of 73.9%. The worst scoring model, logistic regression with doc2vec, achieved 65.0% accuracy. The recurrent network which had the best performance was LSTM with word2vec, and scored an accuracy of 70.4%. LSTM with word2vec was still only the fourth best model overall. Unfortunately did none of the binary classification models achieved any remarkable results, hence we did not precede with the classification problem.

Other observations worth mentioning are that our baseline model did surprisingly well compared to more complex sentence representations and models. It achieved 68.4% and outperformed 7 other models. Also, both LSTM networks with word2vec and BERT had very similar performances, and were only separated by 0.4% accuracy. This was surprising because of the simplicity of both BoW and logistic regression, and might be a pointer that the problem was not embedding or machine learning algorithm related, but connected to the data.

Chapter 9

Conclusion and Future work

9.1 Conclusion

We concluded that the quality of the data held back the sentence representations and the binary classification models. We were therefore not able to achieve any impressive results regarding the classification problem. The conclusion was drawn from the underwhelming performance of the machine learning models on our simplified binary classification problem. This again shows that the quality of the training data is very important, if not the most essential factor for developing a well-functioning dialogue system.

We have pointed out the two main reasons as to why our binary classification failed. The first reason was the condition, quality and quantity of the data. Due to the dataset being unlabeled, we only managed to hand-label a few samples, and were not able to ensure the quality of those samples. The second problem was the length of the utterances. They were remarkably longer compared to utterances in other datasets (see section 8.1). This made the utterances more challenging to classify. We observed that the reason for this was the amount of information conveyed related to driving school services and personal characteristics occurring in the utterances. We have explained both these occurrences in section 7.1.

The problems with the data were observable via the models that we trained. The models were decreasing their cost functions according to the training data, and converging their performance on the validation set. Unfortunately, the performances were not particularly

impressive for any of the models, and the classification problem was not further pursued. However, this lead us to shifting our focus to analyzing the problem, where we learned the importance of both data quality and labeling in developing a well-functioning classification model.

9.2 Future work

Regarding this specific classification problem and dataset, there are a few things we could have further explored. Hyperparameter tuning of the different embedding- and machine learning algorithms is for instance something that we could have experimented more with and that might have boosted the performance of the models. We could also have experimented with different algorithms for generating static embeddings, like GloVE (an extension of word2vec) or fastText [52, Ch6.8.3 p124]. In addition, we could have experimented more with pre-trained models. Examples of such models are pre-trained word2vec, GloVe and fastText models for Norwegian, or other contextualized embedding models like ELMo or GPT [63, Ch1.4 p6 Ch4.4.4.2 p67]. It would have been interesting to see if any of these could change the visualizations, and if they would have made an impact on the performance.

Regarding the neural networks and potential approaches for training and arranging the networks, a vast range of possibilities exist. Given a second round at the dataset, and with more computing resources and time, we would therefore have tried out different options. We would for example have increased the training duration, tried out different optimization algorithms and explored the effect of deeper networks to see if this would have any effect on the performance.

Based on the results and conclusion we would also like to shed some light on the importance surrounding data for dialogue systems, and in general for any supervised classification problem. Furthermore, we would like to suggest some methods for approaching similar problems.

As mentioned in section 6.2, the dataset lay down the foundation for any machine learning model. For classification models, which dialogue systems rely on, correct and sufficient labeled data is essential. However, the process of creating a proper dataset is very costly and time consuming. Our approach, where collected utterances were hand-labeled, is a viable approach but requires a lot of resources, demands proper guidelines and measurements to ensure the quality of the data.

Another approach for collecting and labeling utterances is the *Wizard-of-Oz* setup, which was used in [48] (see section 5.3). In the Wizard-of-Oz setup, the user thinks it interacts with a dialogue system, but is in reality communicating with a human "wizard". The wizard is provided an interface interacting with a system that handles input from users. Based on the users utterances, the wizard can fetch requested information or straight away answer the utterance [52, Ch24 p550]. In this way the dataset may be collected relativity fast and at a low cost [48, Ch3 p3]. Additionally, the system could be configured to guide the users utterances in certain directions and limit the length of them as well. The wizard's interface could also have a setup for labeling each utterance, guiding the dialogue and replying with pre-written templates, to create a dataset efficiently and consistently.

Commercial solution, like for instance Rasa, is another feasible approach. Rasa recommends to build the dataset over time on real data via the dialogue system. Initially, the system will only be based on a few synthetic training examples. By continuously annotating the incoming messages from real users, the dataset will grow. The new data is used to re-train the dialogue system, which will then continue to improve [1] [5]. In comparison to having to hand-label a large collection of already existing data before one can begin the development of the dialogue system, Rasa provides a framework for continuously gathering and selecting data that is then used to gradually enhance the dialogue system.

Bibliography

[1] 10 best practices for designing nlu training data. https://rasa.com/blog/10-best-practices-for-designing-nlu-training-data/. Accessed: 2022-07-04.

[2] The ai community building the future. https://huggingface.co/. Accessed: 2022-05-15.

[3] Aiml docs. http://www.aiml.foundation/doc.html. Accessed: 2022-05-10.

[4] Andrew ng: Unbiggen ai. https://spectrum.ieee.org/andrew-ng-data-centric-ai. Accessed: 2022-02-16.

[5] Annotate nlu examples. https://rasa.com/docs/rasa-enterprise/user-guide/annotate-nlu-examples/. Accessed: 2022-07-04.

[6] Antall ord i norsk. https://www.sprakradet.no/svardatabase/sporsmal-og-svar/antall-ord-i-norsk/. Accessed: 2022-03-17.

[7] Art. 4 gdpr. https://gdpr-info.eu/art-4-gdpr/?fbclid=IwAR2JVVUBZeklyYg7nki0xBBcl1Y9xUN9PrioeeYFX02MU0Am9J_04T273Fk. Accessed: 2022-07-14.

[8] Chatbots to deliver $11bn in annual cost savings for retail, banking healthcare sectors by 2023. https://www.juniperresearch.com/press/chatbots-to-deliver-11bn-cost-savings-2023. Accessed: 2022-06-30.

[9] Doc2vec paragraph embeddings. https://radimrehurek.com/gensim/models/doc2vec.html. Accessed: 2022-04-01.

[10] Exploratory data analysis. https://www.ibm.com/cloud/learn/exploratory-data-analysis#toc-what-is-ex-ofRUduQ6. Accessed: 2022-03-08.

[11] Exploratory data analysis for natural language processing: A complete guide to python tools. https://neptune.ai/blog/exploratory-data-analysis-natural-language-processing-tools. Accessed: 2022-03-09.

[12] From words to vectors. https://towardsdatascience.com/from-words-to-vectors-e24f0977193e. Accessed: 2022-03-17.

[13] Funbit as. https://funbit.no/. Accessed: 2022-03-02.

[14] General agent design best practices. https://cloud.google.com/dialogflow/es/docs/agents-design. Accessed: 2022-07-04.

[15] A gentle introduction to sparse matrices for machine learning. https://machinelearningmastery.com/sparse-matrices-for-machine-learning/. Accessed: 2022-03-17.

[16] How norway's biggest bank automated 51% of its online chat traffic with ai. https://www.boost.ai/case-studies/how-norways-biggest-bank-automated-51-of-its-online-chat-traffic-with-ai. Accessed: 2022-07-24.

[17] Industrial-strength natural language processing. https://spacy.io/. Accessed: 2022-04-05.

[18] Latent semantic analysis using python. https://www.datacamp.com/tutorial/discovering-hidden-topics-python. Accessed: 2022-08-15.

[19] Limit long message. https://forum.rasa.com/t/limit-long-message/15300. Accessed: 2022-07-04.

[20] Nbailab/nb-bert-base. https://huggingface.co/NbAiLab/nb-bert-base. Accessed: 2022-05-15.

[21] nltk.tokenize package. https://www.nltk.org/api/nltk.tokenize.html. Accessed: 2022-04-05.

[22] Norwegian bokmål. https://spacy.io/models/nb. Accessed: 2022-04-14.

[23] Norwegian transformer model. https://github.com/NBAiLab/notram?fbclid=IwAR21taGRLbtzMwZYpjUmNHhlIiNtFVAcCzsnz-PLhfPFlrKUIOl2ZLe4nKM. Accessed: 2022-05-15.

[24] Pandorabots. https://en.wikipedia.org/wiki/Pandorabots. Accessed: 2022-05-10.

[25] Q12: I've used doc2vec infer_vector() on a single text, but the resulting vector is different each time. is there a bug or have i made a mistake? (doc2vec inference non-determinism). https://github.com/RaRe-Technologies/gensim/wiki/Recipes-&-FAQ. Accessed: 2022-05-13.

[26] Rette - uibs prosjektoversikt. https://www.uib.no/personvern/128207/rette-uibs-prosjektoversikt. Accessed: 2022-07-14.

[27] The seven principles. https://www.uhi.ac.uk/en/about-uhi/governance/policies-and-regulations/data-protection/the-seven-principles/?fbclid=IwAR0LLg0zQy3xN13wMWB9mxE4grdRZ3orq7uAzX65Hq9J2_OLhuJutKpA3gQ. Accessed: 2022-07-14.

[28] sklearn.feature_extraction.text.countvectorizer. https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.CountVectorizer.html. Accessed: 2022-04-07.

[29] Spam filtering using bag-of-words. https://medium.com/swlh/spam-filtering-using-bag-of-words-aac778e1ee0b. Accessed: 2022-04-01.

[30] Stemming and lemmatization. https://nlp.stanford.edu/IR-book/html/htmledition/stemming-and-lemmatization-1.html. Accessed: 2021-10-11.

[31] Text preprocessing for machine learning nlp. http://kavita-ganesan.com/text-preprocessing-tutorial/#Relevant-Papers. Accessed: 2022-05-03.

[32] Topic modeling with latent dirichlet allocation. https://www.baeldung.com/cs/latent-dirichlet-allocation. Accessed: 2022-08-15.

[33] Understanding lstm networks. http://colah.github.io/posts/2015-08-Understanding-LSTMs/. Accessed: 2022-06-28.

[34] Understanding umap. https://pair-code.github.io/understanding-umap/. Accessed: 2022-04-21.

[35] The unreasonable effectiveness of recurrent neural networks. http://karpathy.github.io/2015/05/21/rnn-effectiveness/. Accessed: 2022-04-27.

[36] What are word embeddings for text? https://machinelearningmastery.com/what-are-word-embeddings/. Accessed: 2022-03-23.

[37] Word embeddings. https://www.tensorflow.org/text/guide/word_embeddings#encode_each_word_with_a_unique_number. Accessed: 2022-03-23.

[38] Word2vec embeddings. https://radimrehurek.com/gensim/models/word2vec.html. Accessed: 2022-04-07.

[39] Zuckerberg plans to integrate whatsapp, instagram and facebook messenger. https://www.nytimes.com/2019/01/25/technology/facebook-instagram-whatsapp-messenger.html. Accessed: 2022-06-30.

[40] Eleni Adamopoulou and Lefteris Moussiades. Chatbots: History, technology, and applications. *Machine Learning with Applications*, 2, 11 2020.

[41] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Janvin. A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155, mar 2003.

[42] Yoshua Bengio, Yann Lecun, and Geoffrey Hinton. Deep learning for ai. *Commun. ACM*, 64(7):58–65, jun 2021.

[43] Sigrid Lofthus Bjørndal. Exploring pretrained word embeddings for multi-class text classification in norwegian. Master's thesis, Norges teknisk-naturvitenskapelige universitet, 2019.

[44] George E. P. Box. Science and statistics. *Journal of the American Statistical Association*, 71(356):791–799, 1976.

[45] Petter Brandtzaeg and Asbjørn Følstad. Why people use chatbots. 11 2017.

[46] Daniel Braun, Adrian Hernandez Mendez, Florian Matthes, and Manfred Langen. Evaluating natural language understanding services for conversational question answering systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, pages 174–185, Saarbrücken, Germany, aug 2017. Association for Computational Linguistics.

[47] Nikhil Buduma and Nicholas Locascio. *Fundamentals of Deep Learning: Designing Next-Generation Machine Intelligence Algorithms*. O'Reilly Media, Inc., 1st edition, 2017.

[48] Paweł Budzianowski, Tsung-Hsien Wen, Bo-Hsiang Tseng, Iñigo Casanueva, Stefan Ultes, Osman Ramadan, and Milica Gašić. Multiwoz – a large-scale multi-domain wizard-of-oz dataset for task-oriented dialogue modelling, 2020.

[49] Harry Bunt. Dimensions in dialogue act annotation. 2006.

[50] Jianpeng Cheng, Li Dong, and Mirella Lapata. Long short-term memory-networks for machine reading. *CoRR*, abs/1601.06733, 2016.

[51] James H. Martin Daniel Jurafsky. *Speech And Language Processing*. Third edition draft edition, December 30, 2020.

[52] James H. Martin Daniel Jurafsky. *Speech And Language Processing*. Third edition draft edition, January 12, 2022.

[53] Jan Deriu, Alvaro Rodrigo, Arantxa Otegi, Guillermo Echegoyen, Sophie Rosset, Eneko Agirre, and Mark Cieliebak. Survey on evaluation methods for dialogue systems. *Artificial Intelligence Review*, 54(1):755–810, Jun 2020.

[54] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018.

[55] J.R. Firth. *A Synopsis of Linguistic Theory, 1930-1955*. 1957.

[56] Larry B. Wallnau Frederick J Gravetter. *Essentials of Statistics for the Behavioral Sciences*. CENGAGE, 8st edition edition, 2012.

[57] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016. `http://www.deeplearningbook.org`.

[58] Aurélien Géron. Hands-on machine learning with scikit-learn, keras, and tensorflow. September 2019.

[59] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9:1735–80, 12 1997.

[60] Per E Kummervold, Javier De la Rosa, Freddy Wetjen, and Svein Arne" Brygfjeld. Operationalizing a national digital library: The case for a Norwegian transformer model.

[61] Quoc V. Le and Tomás Mikolov. Distributed representations of sentences and documents. *CoRR*, abs/1405.4053, 2014.

[62] Yang Liu Li Deng. *Deep Learning in Natural Language Processing*. Springer, 1st edition edition, May 31, 2018.

[63] Zhiyuan Liu, Yankai Lin, and Maosong Sun. Representation learning for natural language processing. *CoRR*, abs/2102.03732, 2021.

[64] Leland McInnes, John Healy, and James Melville. Umap: Uniform manifold approximation and projection for dimension reduction, 2018.

[65] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg Corrado, and Jeffrey Dean. Distributed representations of words and phrases and their compositionality, 2013.

[66] Tomas Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751. Association for Computational Linguistics, jun 2013.

[67] Fred J. Damerau Nitin Indurkhya. *Handbook of Natural Language Processing*. Second edition edition, 2010.

[68] Abhinav Rastogi, Xiaoxue Zang, Srinivas Sunkara, Raghav Gupta, and Pranav Khaitan. Towards scalable multi-domain conversational agents: The schema-guided dialogue dataset. 2020.

[69] Jetze Schuurmans and Flavius Frasincar. Intent classification for dialogue utterances. *IEEE Intelligent Systems*, 35(1):82–88, 2019.

[70] Ralf C. Staudemeyer and Eric Rothstein Morris. Understanding LSTM - a tutorial into long short-term memory recurrent neural networks. *CoRR*, abs/1909.09586, 2019.

[71] Greg Corrado Jeffrey Dean Tomas Mikolov, Kai Chen. Efficient estimation of word representations in vector space. abs/1301.3781, 2013.

[72] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[73] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017.

[74] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. Google's neural machine translation system: Bridging the gap between human and machine translation. *CoRR*, abs/1609.08144, 2016.

[75] Steve Young, Milica Gašić, Simon Keizer, François Mairesse, Jost Schatzmann, Blaise Thomson, and Kai Yu. The hidden information state model: A practical framework for pomdp-based spoken dialogue management. *Computer Speech Language*, 24:150–174, 04 2010.

[76] Tom Young, Frank Xing, Vlad Pandelea, Jinjie Ni, and Erik Cambria. Fusing task-oriented and open-domain dialogues in conversational agents, 2021.

Appendix A

Guidance for domain and intent labeling

Guidance for Domain & Intent

The goal of the *natural language understanding component* in the *frame-based architecture* is to extract three things from the user's utterance.

Domain classification: Is this user for example talking about airlines, programming an alarm clock, or dealing with their calendar? Of course this 1-of-n classification tasks is unnecessary for single-domain systems that are focused on, say, only calendar management, but multi-domain dialogue systems are the modern standard.
Intent determination: what general task or goal is the user trying to accomplish? For example the task could be to Find a Movie, or Show a Flight, or Remove slot filling a Calendar Appointment.
Slot filling: extract the particular slots and fillers that the user intends the system to understand from their utterance with respect to their intent.

Multi-output classification:
Classifier-models that output multiple labels. Probably not necessary, but won't hurt.

"The simulator also allows for multiple intents to be active during a given turn."

- *SGT*

"TOP, which pertains to navigation and event search, is unique in that 35% of the utterances contain multiple, nested intent labels. These hierarchical intents require the use of specialized models."

- *Learn to Classify Intents*

T. Solbakken labels:

Domains | Description:

Courses	Any cours, TG, Mrk etc
IntensiveCourses	Intensive courses for any licences
Licences	Any licence
Package price	Any package
General	General questions
Finance	Financing lessons
Payment	Anything regarding paying
Complex	Anything that's seems challenging and should be handled of an employee.
DrivingTest	Anything regarding driving test
Other	Possible features for later, **REVISIT.** *Oppkjøring,*
Request	Customer request beyond chatbot, **REVISIT** *Eger customers*, *unordinary req*,
Toss	**REVISIT**
Advice	Customer asks for any advice
Corona	Corona related messages
Giftcard	Giftcard related messages
OnlineCourse	Anything regarding online courses
Rudskogen/Førerutviklingskurs	Anything regarding Rudskogen or Førerutviklingskurs

Intents | Description:

Intent:	Description:	Action:
alterBill	change bill, => getEmployee	
alterBooking	change booking, => getEmployee	
alterBookingOther	Asks to change booking for another person	
cancelBooking	cancel one or more bookings	
confirme	Customer confirmed current state	
confirmMultiple	(Intent for production) Confirm multiple slots	
deny	Customer deny current state	
denyMultiple	(Intent for production) Customer deny multiple slots	
endOfConversation	Customer thanks or says bye	
freeLesson	Anything regarding Free Lessons	
gatherExperience	Customers provide experian. Slot extraction later.	
gatherInformation	Customers provide relevant information,	Slot extraction later.
getAvailable	Customer asks for available courses/driving lessons	Link to page course page or (licence => getEmployee)
getBill	Customer ask for bill => getEmployee	
getBooking	Show booking (time, location, instructor)	

getBookingOther	Asks for booking regarding another person	
getContactInformation	Gets some information about contact	
getEmployee	Get employee	
getEstimatedTime:	Estimated time of an event	
getInformation	Customer ask for information regarding domain	Link to related page at website
getInformationComplex	Complex req about info	
getLocation	Asks where to meet, office location or service location (Norheimsund, Sotra etc)	
getPaymentMethode	Asks about payment methodes.	
getPractical	Practical information about current domain	
getPrice	Asks for price	Link to price page, and section at page.
getPractical	Customer ask practical Q, => getEmployee	
getStarted	When a customer say they want to get started, or something regarding ‘Prøvetime’.	
getTime	Customers request time-slots for the specific domain. After dinner? When and for how long.	
None		
Other	**Revisit**	

postBooking	Customer wants booking information after booking is completed	
reqBill	Customer req the bill	
reqBooking	Customers want to book. All domains (license/driving lesson, course, package, intensive course).	Slotts might differ.
reqCompetition	Customer replays to competition	
reqGiftcard	Customer request giftcard	
reqMultipleBooking	Book multiple events, => getEmployee	
reqPrice	More custom req for price => getEmployee	
suggestBokking	Customer is wondering to book	Suggest booking, link to book-page
suggestLocation	Customer want to propose location	
suggestTime	Customer suggest time	

Chatbot specific intents

Intents that are not in the dataset but might be relevant in production.

challenge	Ask about chatbot
getOptions	Ask about options for current domain
getEmployee	Customer req a human

Licence class

This feature categorieze licence classes, might be useful.

A (Tung)	Motorsykkel lett
A1 (Lett)	Motorsykkel mellom
A2 (Mellom)	Motorsykkel tung
AM146 (Mop)	Moped
B	Personbil
B96 (4250kg)	Personbil m/tilhenger
BAut	Personbil (automat)
BE (7000kg)	Personbil m/tilhenger
Annet	
None	