OULUN AMMATTIKORKEAKOULU

Elina Kuuluvainen

CLASSIFYING NEWS ARTICLES BASED ON USER NEEDS USING TRANSFER LEARNING AND DEEP NEURAL NETWORKS

A multi-class approach combining BERT with non-textual features

CLASSIFYING NEWS ARTICLES BASED ON USER NEEDS USING TRANSFER LEARNING AND DEEP NEURAL NETWORKS

A multi-class approach combining BERT with non-textual features

Elina Kuuluvainen
Master's Thesis
Autumn 2023
Degree Programme in Data Analytics
and Project Management
Oulu University of Applied Sciences

ABSTRACT

Oulu University of Applied Sciences
Degree Programme in Data Analytics and Project Management

Author(s): Elina Kuuluvainen
Title of the thesis: Classifying news articles based on user needs using transfer learning and deep neural networks
Thesis examiner(s): Ilpo Virtanen
Term and year of thesis completion: Autumn 2023 Pages: 86

This thesis describes the procedure for developing a machine learning model that can categorise news articles based on audience insight experts' assessments of user needs. This study aims to determine if it's possible to employ a machine learning algorithm for identifying user needs within the context of Yle News. The research method involves fine-tuning a pre-trained large language model, BERT, by making use of both textual and non-textual features of news articles. The data is collected from Yle's data warehouse and combined with manually labelled user needs classes. Python, along with the PyTorch and Hugging Face Transformers libraries, is used in implementing the model. The results show the model can effectively categorise articles into user needs groups with a relevantly appropriate level of accuracy. However, limitations arise from the small dataset size and its uneven distribution across various user needs categories. Based on the results, the trained model might not be well generalisable to unseen data. The research concludes by suggesting avenues for improvement and application, such as acquiring a more balanced dataset while exploring alternative deep learning models. In summary, this study showcases how machine learning algorithms hold potential for classifying news articles based on user needs; nevertheless, further refinement and development are required for future use cases.

Keywords: Neural Networks, News Articles, Classification, Natural Language Processing, BERT

TABLE OF CONTENTS

ABBREVIATIONS AND ACRONYMS

AI	Artificial Intelligence
AR	Autoregressive
BERT	Bidirectional Encoder Representations from Transformers
BPE	Byte-Pair Encoding
CPU	Central processing unit
CSV	Comma-Separated Value
DL	Deep Learning
DNN	Deep Neural Network
FNN	Feedforward Neural Network
GPT	Generative Pre-Trained Transformer
GPU	Graphics processing unit
LLM	Large Language Model
ML	Machine Learning
MLM	Masked Language Modelling
MLP	Multi-Layer Perceptron
MPS	Metal Performance Shaders
NLP	Natural Language Processing
NN	Neural Networks
RNN	Recurring Neural Network

1 INTRODUCTION

This research is done for the Finnish Broadcasting Company, later referred to as Yle, News and Current Affairs Unit. Working as an audience insight analyst at Yle News Lab, a future and development division of Yle News and Current Affairs, I was introduced to a real-life problem in the organisation's ability to analyse published content. In this problem, I saw an opportunity to combine our team's understanding of user needs with a personal interest in learning more about Machine Learning (ML) technologies.

1.1 Background and Motivation

Yle is a public service medium and therefore, in principle, meant to serve the whole public of Finland. However, the latest Reuters Digital News Report reveals that while most Finns think that having a public service media is important to both society and for them individually, 40% of people do not find Yle at all personally important to them, and the decline in youth engagement is especially critical for the public service's future. Young people, after growing up with a variety of social media platforms, say they trust influencers more than traditional journalists. If these new generations do not at any point develop a relationship with the public news service, they will contribute to the growing percentage of "Not at all" answers when asked "How important, or not, are publicly funded news services such as Yle to you personally or for society?". Maintaining the range of public service reach is crucial for future legitimacy, particularly among young people. (Newman et al. 2023, 10,45.)

To be an engaging, relevant medium, it is important to oversee and manage the bigger picture of all content that is published. Yle categorises its audio and video contents, both in broadcast and digital channels, based on the user needs they meet. The categorisation is part of portfolio management work, which ensures a balanced content offering. The portfolio is a tool that Yle uses to serve a range of audiences and media needs. Digital news articles, however, are a huge part of that portfolio that cannot be analysed in the same way at the moment. There are almost a thousand news articles published at Yle.fi each week that have not been analysed based on what needs they meet. As they say, one cannot manage what is not measured.

The motivation behind this study is to see if ML could help with measuring what is published to better manage the mass of online news articles produced every day. Getting a better overview of

what is published as news articles could lead to more informed decision-making in journalism and thus improve the overall quality and balance of the daily news offering. These factors play into making Yle News as a service more compelling to audiences.

Because the publishing volume is so large, doing a large content analysis manually does not seem to be a viable option for understanding the division of articles according to user needs. Therefore, the research question is whether it is possible to use a machine learning algorithm to identify the different user needs in the context of Yle News.

The inspiration for this came from Smartocto (2021), a company offering an editorial analytics system developed by Dmitry Shishkin. While working for the BBC World service, Shishkin worked on and further developed an audience-centric content model based on news user needs.

1.2 Aims and Limitations of Study

As published in the Smartocto white paper “Actionable User Needs” (2021, 5, 15), the BBC World Service began examining the balance of its news coverage back in 2016. Their findings were that 70% of content at one station fell into one category, "update," while accounting for only 7% of traffic, and this caused them to critically rethink how they produced content. These findings of overproduction on update articles were later found to be the case in most newsrooms it was tested in, and the user needs model created back in 2016 was quite universally applicable.
By applying a similar logic to a machine learning algorithm, Yle could also analyse large sets of articles from the users’ needs’ perspective. It is evident by merely browsing Yle.fi that most of the articles Yle News and Current Affairs publishes, as well as in the Smartocto studies, fall into the category of keeping the reader up to date, but the goal is to visualise exactly how much of a majority it is.

If it is found that the algorithm manages to successfully categorise news articles, the model could be further developed into a tool for news desks to use in continuous monitoring of publishing volumes and public engagement. Comparing publishing volumes with reach and engagement in different user needs categories could be the information that works as a driver for change in newsrooms when planning what content to publish and in what form.

The scope of this study is to train a simple classification model using natural language processing (NLP) combined with non-textual data.

The motivation is to categorise news articles published at Yle.fi, which in this case refer to articles from the Yle News and Current Affairs unit.

While Yle News and Current Affairs is the largest organisational unit by publishing volume, it is not the only one producing content. Other departments like Sports and Events and Creative Content and Media also produce articles that are available at Yle.fi and the Yle application. Their content involves, for example, sport updates and articles on nature, science, lifestyle, and entertainment. Classifying those articles with new, unseen data is not within the scope of this study because of differences in unit objectives and areas of focus. The aim is to investigate the possibility of utilising an ML tool's outputs for Yle newsroom decision-making specifically. This is a reference to contents that fall under the responsibility of the Yle News and Current Affairs editor-in-chief. While non-newsroom-produced articles are not the focus in possible future applications, all available online articles are used as teaching material for the model. This is done to increase the diversity of user needs in the training dataset.

Svenska Yle, who provides content to digital Yle in Swedish, is excluded from the study dataset and scope because the focus is on using Finnish and using large language models pre-trained with Finnish.

The dataset labelling was done by the Yle News Lab Audience Insight team members, and it was not based on a journalistic decision.

While this thesis document is completed in the autumn of 2023, most of the building of the solution was timed a year prior, in the autumn of 2022, and the gathering of the data was earlier in 2022. Therefore, the solutions are mostly built on the data, technologies, and tools available at that time.

1.3 Structure of Thesis

The first part of the thesis is an overview of the research background. First, the background explains what users' needs and motivations behind news media content refer to in this case and how they are applied in this study. This is followed by an overview of machine learning, containing details of relevant theoretical features about neural networks and natural language processing. The second part comprises a description of building an example solution and an analysis of the results. The second part ends with conclusions about the study and possibilities for future applications.

2 RESEARCH BACKGROUND

This chapter presents the background information for this research, beginning with a brief introduction to user needs classification in the organisation and how the needs are interpreted in this study. A description of the data used in this case concludes the chapter.

2.1 News and User Needs

At Yle, understanding user needs and motives is crucial for effective portfolio management. Insights into media usage motives and needs guide customer-centric decisions.
Four primary media usage motives have been identified, each linked to fundamental human need:

TABLE 1. Media usage motives and their corresponding human needs

Motive Category	Linked Human Need
Social	Connecting to Others
Information	Understanding the World
Emotional	Regulating Mood
Identity	Self-Actualization

These motives have been further adapted into four categories for portfolio management at Yle News and Current Affairs, guiding the management of audio and video products in both broadcast and digital formats. These can be roughly translated as follows:

TABLE 2. Existing portfolio categories and media usage motives

Category Name	Associated Motives
Fact for Keeping Up to Date	Information, Social
Topic-Driven Fact	Information, Identity
Fact with an Experience	Identity, Information
Entertaining/Community-Creating Fact	Social, Emotional, Identity

Even though the Actionable User Needs model (Smartocto, 2021), which originated in the BBC, was found to universally apply across newsrooms, it was thought best to use the Yle News and Current Affairs categorisation as a base instead in this research. This was mainly because of an existing categorisation of similar nature for content management. The Smartocto model from 2021 comprises six categories and would not be easily paired with the categorisation that was already being used in the organisation. Therefore, the Yle News Lab Audience Insight team developed four distinct user needs categories, corresponding to the portfolio management categories.

TABLE 3. User needs for news articles

User Need Category	Description
Need for Information	Articles that keep the reader up to date focusing on providing current information
Need for Understanding	Articles offering in-depth insights and understanding, like background information to a topical phenomenon with points of view and analysis
Need for Personal Connection	Articles encouraging self-reflection, learning, and identity. For example, articles that explain a phenomenon through people or a person.
Need for Connecting to Others / Need for Experiences	Articles with communal aspects or about topical phenomena. For example, interactive articles, like games, quizzes and chats, or articles that contain engaging media clips of current phenomena and / or event.

2.2 Data Description

The data used in this study was obtained from Yle, where, as an analyst in an audience insight team, I am privileged with access to relevant, detailed information on published news articles. This research uses three types of data on news articles: automatically generated metadata about news articles (such as their title, used keywords, exact time of publishing, format of the article), the contents of the articles, and a limited dataset with articles linked to the user needs they meet, which was specifically manually created for this research. All this data is useful for understanding the relationship between news articles and the needs they serve.

Yle uses Amazon Web Services (AWS) as its cloud computing platform. For data management and storage, a variety of solutions are used in different stages of data management. All the relevant data for this research is stored in Amazon Redshift, a data warehouse that enables the analysis of large datasets. It can be directly accessed with SQL queries, for example, when an AWS user account belongs to an appropriate IAM (Identity and Access Management) user group with relevant

cluster access policies. At Yle, selected sets of data can also be accessed and analysed with limited data models put together for business intelligence (BI) tools such as Tableau and Power BI. This information in its visualised report form can, at its lowest security level, be accessed with any Yle employee AD account identification, but exploring the data models' content in detail is at appropriate times restricted to relevant roles and team members by applying user and group-level access rights to data and models in business intelligence environments.

The editorial systems used to manage and publish articles are the primary source for the metadata about the articles. The publishing systems generate some data automatically, such as the precise publication time, while the journalists manually add or edit other parts when publishing, for example, by choosing appropriate keywords or adding missing author information.

The dataset with a limited number of articles labelled with the user needs they meet was specifically manually created for this research case by the Audience Insight Team members of Yle News Lab. The dataset was initially created as a Google Sheet in Yle's Enterprise Google Drive, but the data was then used as a local comma-separated value (CSV) file when linked to a larger set of data in the pre-processing of data for the analysis.

3 MACHINE LEARNING

Machine learning (ML) is a powerful tool that makes use of artificial intelligence (AI) technologies. Even though often referred to as AI itself, machine learning is merely a subfield of artificial intelligence, one that is focused on developing computer programmes that learn from data without being explicitly instructed what to do. The learning is driven by algorithms and statistical models that allow the machine to analyse and predict outcomes from data patterns. Machine learning is now an important part of business and research almost everywhere.

Even though the dawn of machine learning dates to the 1940s and 1950s, its powerful everyday applications needed to await access to less-expensive computing power and the better availability of data. Along with these developments, machine learning has taken impressive leaps in the past decades.

The introduction of Deep Learning (DL) has made it the dominant approach in machine learning. DL uses Neural Networks (NN) with multiple layers to understand complicated, hierarchical data structures. DL has achieved notable success in a variety of fields, including image recognition and natural language processing. (LeCun et al. 2015,1.)

For example, the recent release of ChatGPT, a sophisticated language model powered by Deep Neural Networks (DNN), has significantly elevated the field of text-based data analysis, and broadened the scope for utilising ML-driven tools across various industries. Overall, machine learning is currently making significant progress and finding practical uses in almost every aspect of human life. This chapter will provide a brief introduction to the fundamental principles and concepts of ML, focusing on those relevant to this use case.

3.1 Learning Scenarios for Models

Learning in ML refers to training a model using a set of data, which enables it to make reliable and precise predictions on new data. This involves the algorithm's ability to identify patterns within the data and subsequently adjust the model parameters accordingly.

There are three primary learning scenarios for machine learning models: supervised learning, unsupervised learning, and reinforcement learning (Chandramouli et al. 2018, chapter 1.5).

In supervised learning, also known as predictive learning, a computer predicts what class an

unknown object belongs to based on what it knows about similar objects. Labelled training data containing past information is given to the model as input, and the machine creates a predictive model based on the input data. This model can then give an output value to a new sample it is introduced to. Generally, there are two types of modelling in supervised learning, depending on the desired output variable. There is classification when the output is a discrete variable, such as a list with a category or a label. For continuous variables, such as numbers, the model is a regression model. Instead of distinct labels, regression might predict temperatures, prices, and revenues, for example. (Chandramouli et al. 2018, chapter 1.5.)
Most people encounter supervised learning models being used in their day-to-day life, for example, classifying emails as spam or non-spam, recommending content on streaming platforms, and giving credit-risk assessments on loan applications.

Unsupervised learning, on the other hand, deals with unlabelled input data and aims to uncover hidden patterns within it. This is widely used, for example, in identifying meaningful trends, structures, and groupings in results. Tasks such as clustering, dimension reduction, anomaly detection, and mining for association rules fall under unsupervised learning. (Sarker, 2021,4.) Clustering is likely to occur when online stores suggest items that are often bought together. Anomaly detection is used, for example, when banks are alerted to automatically shut down bank cards when the model flags transactions as differing from the norm.

Reinforcement learning means the model learns how to perform actions in a specific environment to achieve a predetermined aim. The machine aims to improve its performance by doing the task based on reward and penalty. When a sub-task is accomplished successfully, a reward is given. When the opposite occurs, there is a penalty. Applications of reinforcement learning can be seen in robotics, game play, and navigation. A contemporary example of reinforcement learning is self-driving cars. (Chandramouli et al. 2018, chapter 1.5.)

Besides the three primary scenarios, there are a couple of other approaches worth mentioning: semi-supervised learning and transfer learning.
Semi-supervised learning is a type of learning that uses both labelled and unlabelled data and can thus be defined as a hybridisation of the above-mentioned supervised and unsupervised methods. This approach becomes valuable when there is a large amount of unlabelled data and only a limited amount of labelled data to work with. The main goal of a semi-supervised learning model is to make predictions that are more accurate than those made with just the labelled data from the model.

Some areas where semi-supervised learning is used are machine translations, fraud detection, and data labelling. (Sarker, 2021,4.)

Transfer learning is a form of machine learning that aims to enhance the performance of a learner in one field by applying the knowledge previously gained from a different, but related, field. The goal is to achieve efficient learning while minimising the reliance on extensive labelled data specific to the task at hand. Transfer learning proves beneficial when access to labelled data for the target domain is excessively costly, time-consuming, or impractical. Because of the wide application possibilities and access to large pre-trained models, transfer learning has become a popular and promising area in machine learning. (Zhuang et al. 2020, 1.)

According to Géron (2018, chapter 2), it is typically not a good idea to train one from scratch for large deep neural networks. Instead, one should always try to find an existing neural network that accomplishes a similar task to the one at hand. It will not only speed up training considerably but will also require much less training data. For example, suppose one has access to a large model that was trained to classify pictures into 100 different categories, including animals, plants, vehicles, and everyday objects. If the task at hand is to train a model to classify specific types of vehicles, one should consider that these tasks are very similar and should try to reuse parts of the first network.
This study uses transfer learning by harnessing the knowledge a large language model (LLM) has already learned. The fine-tuning process allows the LLM to transfer the general knowledge it gained from pre-training to this specific task, improving its performance on the task at hand.

3.2 Natural Language Processing

Natural Language Processing (NLP) is a field of ML that aims to enable machines to understand spoken and written languages.

NLP has emerged to play a key role in people's day-to-day lives, as it is a vital component of multiple applications and services that make interactions with technology easier. Currently, it might be difficult to imagine a life without it. For example, imagine a world where search engines did not understand language but just searched for given sets of characters. Googling for the right answer to a question would be quite a different experience. Besides search terms, NLP is present, for example, in grammar and spell checkers, email spam filters, language translators, customer service

chat bots, social media content recommendation and moderation, and voice assistants like Siri or Alexa.

Working with natural language is challenging for machines because of the ambiguities present in language. Machines face the task of accurately interpreting and clarifying the intended meaning in accordance with the context at hand. Besides context and meaning, every language has its own grammar rules, syntax patterns, and vocabulary, which further complicate matters, not to mention dialects, slang, and informal forms of language.
For instance, in these two sentences, "Bats are flying mammals" and "Right off the bat," the word "bat" carries two distinct meanings. To correctly interpret or disambiguate this term within its given context is crucial.

Speech recognition, part of speech tagging, named entity recognition, syntactic parsing, semantic analysis, and discourse analysis are just some of the language processing tasks that NLP systems can do (Prakash, 2022).

Even though the surge of everyday language applications might seem quite recent, this field of AI too has been evolving for quite some time.
The different approaches to NLP from earliest to newest can be summarised as symbolic NLP, statistical NLP, and neural NLP, as explained by Geetha M (2021).
In the early days, many language systems were designed using symbolic NLP. This approach involves using hand-coded rules and dictionaries to design language systems. It is still used where there is insufficient training data or for pre-processing and post-processing tasks.
Statistical NLP was taken into use in the 1980s and mid-1990s. It is an approach where machine learning algorithms are used to automatically learn rules from large amounts of real-world data. It relies on statistical inferences to generate more accurate results.
Modern neural NLP, on the other hand, uses DL techniques and neural networks to handle complex sequence-to-sequence transformations. It reduces the need for extensive feature engineering and captures the semantics of words through word embeddings.

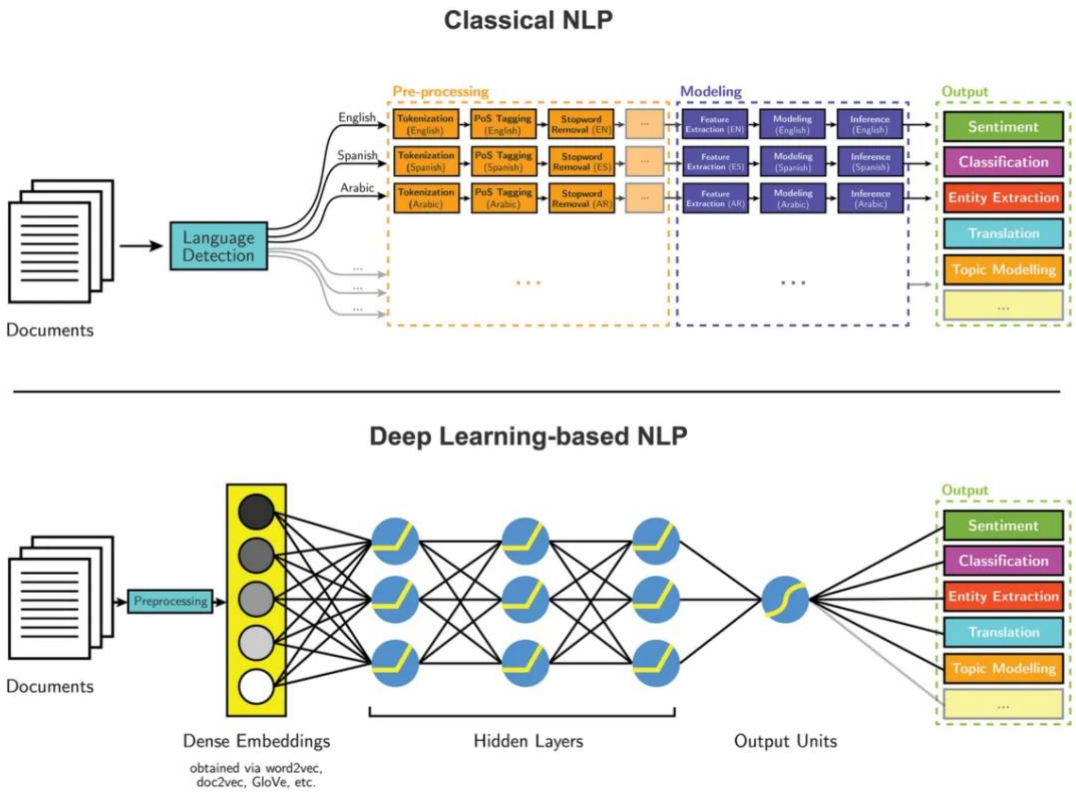

FIGURE 1. Classical NLP and Deep Learning-based NLP architectures (Yang & Liu 2021)

Overall, all NLP can be divided into two parts: natural language understanding and natural language generation (Geetha M 2021).

Natural language understanding is the process of making systems understand a natural language input via text or speech. This understanding is gained through numerous steps of analysis.
Lexical analysis is the process of breaking down text into paragraphs, sentences, and words. It involves analysing the structure and meaning of words. Syntactic analysis involves studying grammar and how words are arranged to establish relationships between them. Semantic analysis aims to determine the meaning of words and the overall meaning of a text.
Disclosure analysis is a step that considers the context of the text. The meaning of a sentence is carefully examined before reaching a conclusion. Pragmatic analysis focuses on language use and interpretation in situations with an emphasis on communication. Natural language understanding is what this study utilises.

Natural language generation, on the other hand, is the process of producing meaningful phrases and sentences in the form of natural language.
Language generation steps involves discourse generation, which refers to transforming input into an output represented as a content tree. In sentence planning, the focus is on converting the

generated text into a linear form according to grammar. Lexical choice means selecting words in the generated text. Sentence structuring ensures that sentences are formed based on syntax rules. Morphological generation is the final stage where possible structure corrections are made to ensure there are no discrepancies, for example, in tense or gender. Sophisticated, large language models (LLMs) employed as chat-based tools, like the popular ChatGPT, are a great current example of the successes in NLG.

To understand LLMs, relevant concepts like the Transformer architecture, attention mechanism, and tokenization will be introduced later in this thesis.

3.3 Classification

As briefly introduced in a prior chapter, classification is a supervised learning technique used in machine learning to categorise data into classes or groups. The goal is to teach a model to predict the class of an object based on its features. Some examples of classification include image classification, sentiment analysis, disease prediction, the prognosis of game outcomes, and forecasting natural disasters. (Chandramouli et al. 2018, chapter 1.5.)

A classification model can be one of these variations: binary, multiclass, or multi-label.
Binary classification means that the algorithm classifies the processed object into one of two categories. For example, it determines whether a patient should be classified as having "disease" or "no disease" based on a series of tests for a specific illness.
With multiclass classification, the algorithm can assign the processed object to one of several categories. However, only one category can be assigned to each object. For instance, when classifying trees detected in an image, an object could be a birch tree, a pine, an oak, or a willow, but not any of the two at the same time.
In multi-label classification cases, the algorithm is expected to provide a list of categories or labels to which the object belongs. For example, a social media post could be simultaneously classified as being about sports, technology, and family.
All three types of classification aim to determine the class or classes to which an instance belongs based on characteristics.

In all these classification scenarios, there is an option to select from a variety of algorithms for the task. Some popular techniques include Logistic Regression, Support Vector Machines (SVMs), Decision Trees, Random forests, and K-nearest neighbour, just to name a few. Logistic Regression

is a statistical method used for predicting binary outcomes. SVMs, on the other hand, are supervised learning algorithms that find the boundary that separates different classes in the data. With Decision Trees, predictive modelling is done by creating a tree-like structure that represents the relationships between variables, and a Random Forest is an ensemble that combines multiple decision trees. K-nearest neighbour classifies a data point based on the majority class of its k nearest neighbours. (Gupta 2023.)

Neural Networks (NN), however, have emerged as a popular alternative to these various standard classification methods, especially for complex tasks. Therefore, these briefly introduced technologies are not gone into in any further detail, but what follows next is an introduction to Neural Networks.

3.4 Neural Networks

Neural Networks are models that take inspiration from the structure of biological neurons. They are arrays of simple computer functions that are highly interconnected and strongly constructed on brain structure.
The technology has been around for decades, but only quite recently have computing resources and data availability made it possible for NNs to rise as the algorithm to outperform others in many machine learning classification tasks. (Géron 2018, chapter 1.)

Currently, with the availability of open-source machine learning libraries like TensorFlow, Keras, or PyTorch, one can create a neural network, even a complex one, with just a few lines of code. Understanding the mathematics behind them and deep learning can, however, help understand what happens in the network. This is important for architecture selection, deep learning model fine-tuning, hyperparameter tuning, and optimization (Dasaradsh 2020).

The fundamental building block in NNs responsible for processing inputs and producing outputs is an artificial neuron called a node. Each node receives various inputs through connections and transfers them to adjacent nodes. The nodes are arranged and organised into linear arrays known as layers. (Alaloul & Qureshi 2020.)

These models comprise interconnected nodes that analyse information and generate results. The connections between nodes are represented by arcs with varying weights, determining the strength of the relationships. Input units receive data, which dedicated output units then process. The

network's behaviour is defined by its architecture, data encoding method, and training algorithm. The neuron model provides an understanding of how each unit generates an output based on its inputs.

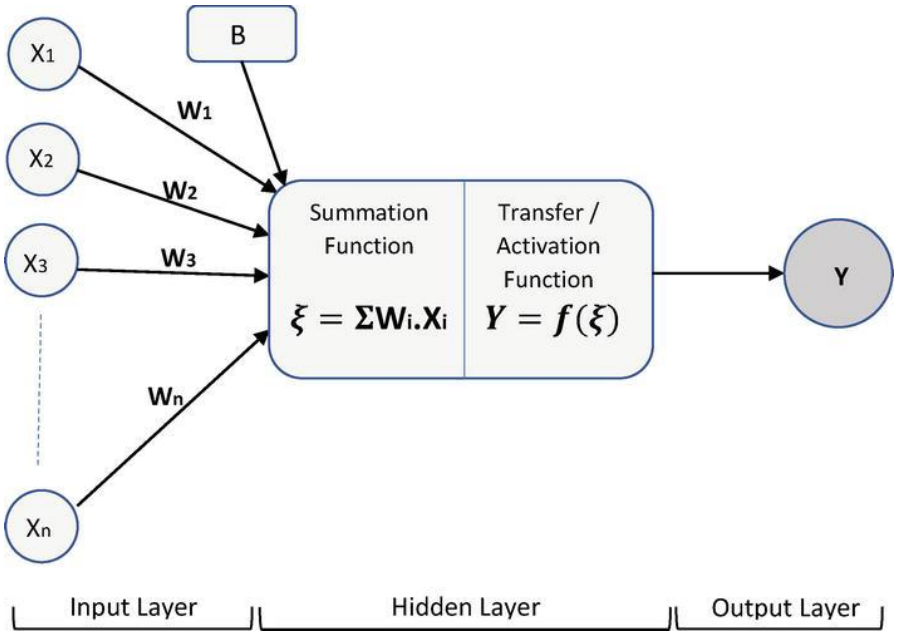

FIGURE 2. General neural network structure (Alaloul & Qureshi, 2020)

The simplest neural network, a perceptron, performs forward propagation, which passes data through the neural network, as described in the following three steps by Dasaradh (2020). A perceptron is a single-layered NN with a single output.

For each input, the process begins by multiplying input values x_i with weights w_i and summing the results. Weights represent the strength of the neuron-neuron connection and determine how much input affects output. The input x_1 will have a greater impact on the output if w_1 is greater than w_2. The summation function notation ξ used in figure 2 is written in the formulas below as the summation sign Σ to align with more conventional notation.

$$\Sigma = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

The input and weight row vectors are $x = [x_1, x_2, \ldots, x_n]$ and $w = [w_1, w_2, \ldots, w_n]$, respectively. The summation equals the dot product by:

$$x \cdot w = (x_1 \times w_1) + (x_2 \times w_2) + \cdots + (x_n \times w_n)$$

Thus, the summation is equal to the dot product of the vectors x and w.

$$\Sigma = x \cdot w$$

The next step is to sum multiplied values with bias b to get z. In most cases, bias (offset) is needed to move the activation function to the left or right to generate the desired output values.

$$z = x \cdot w + b$$

The final step is to pass z to a non-linear activation function.
By adding non-linearity to the output of the neuron, the model can learn from its mistake and change the weights of the neurons during the backpropagation process. Activation functions also significantly affect neural network learning speed.
Perceptrons have a binary step function as their activation function, but in this example, ReLU (Rectified Linear Unit) is introduced in its place because that is what the model in the study uses. This is how the predicted value Y is formed:

$$Y = ReLU(z) = max(0, z)$$

The choice of activation function can vary based on the specific architecture and task at hand. Common choices include the sigmoid, hyperbolic tangent (tanh), and softmax functions, among others.

The network topology of an NN refers to the arrangement and organisation of neural nodes. It includes factors such as the number of hidden neurons, the number of hidden layers, the data flow, the interconnections between neurons, and the specific transfer functions.

3.4.1 Feedforward Networks and Feedback Networks

As explained in Data Processing Using Artificial Neural Networks (Alaloul & Qureshi 2020), the links between nodes in a NN can be broken down into two types: feedforward networks and feedback networks.

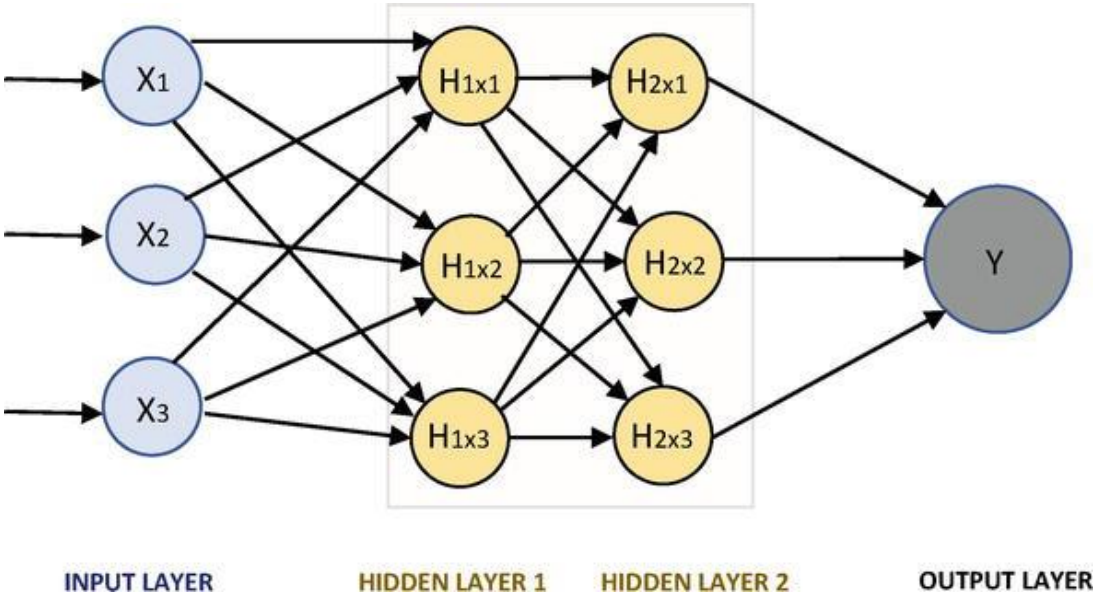

FIGURE 3. A generic Feedforward Network (Alaloul & Qureshi 2020)

Feedforward networks (FNNs) are designed to operate only in one direction and do not possess any form of reverse cycle. Due to this inherent quality, their transmitted signals progress solely within this predetermined path, resulting in an inactive state for the network.

Feedforward networks are often referred to as Multi-Layer Perceptrons (MLPs). They are most effective for tasks that involve a static mapping between inputs and outputs, like classification or regression. This is due to the absence of feedback loops in the feedforward network, which limits its ability to process data in only one direction. (Peixoto 2020.)

In feedback networks, on the other hand, nodes possess backward connected loops, wherein the output of the nodes can serve as input to either the same level or previous nodes. This differs from a feedforward network, as feedback networks are dynamic in nature. In such networks, signals are transmitted both in forward and backward directions. Recurrent Neural Networks (RNNs) are a specific type of feedback network where connections between nodes form a directed graph along a temporal sequence. This allows them to exhibit temporal dynamic behaviour and use their internal state (memory) to process sequences of inputs. (Alaloul & Qureshi 2020.)

RNNs are better suited for tasks that necessitate a dynamic mapping between inputs and outputs, such as sequence-to-sequence tasks like language translation or image captioning (Peixoto 2020).

Deep Learning is a concept that aims to utilise deep neural networks (DNNs) consisting of numerous layers and units. DL has significantly advanced from traditional NNs and serves a vital role in tackling intricate tasks like image classification, image recognition, and speech identification.

DNNs are a special kind of neural network. They have many hidden layers of neurons that are all connected to each other and process data in a way that is hierarchical between the input and output layers. This allows for a more complex and accurate learning process, as the data is processed through multiple layers, each of which is responsible for extracting specific features from the data. (Sarker 2021.)

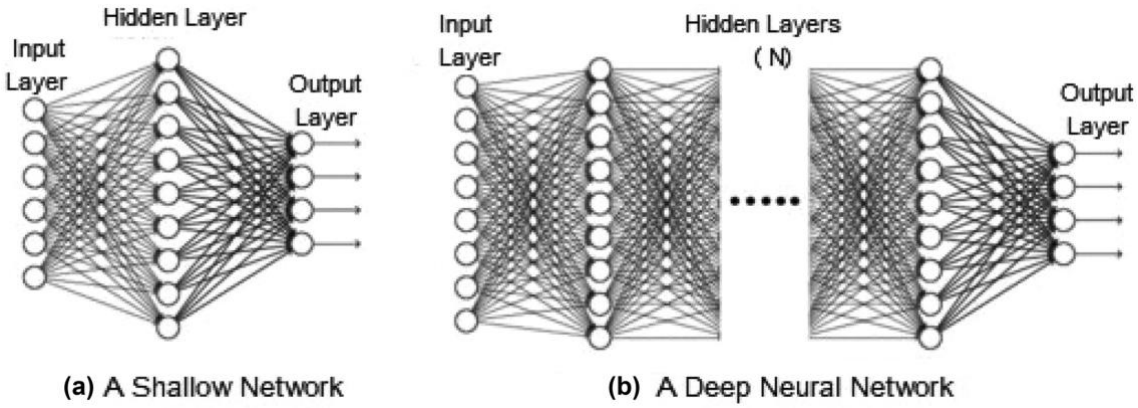

FIGURE 4. The distinction between a shallow network and a deep neural network (Sarker 2021)

As summarized by Sarker (2021), DL triumphs over a well-known obstacle faced by traditional ML because it is more flexible and adaptable to different data types. Deep learning's ability to handle unstructured raw data and automatically select the most important features while disregarding the non-important ones sets it apart from other learning alternatives. These types of networks can manage large datasets while upholding precision.

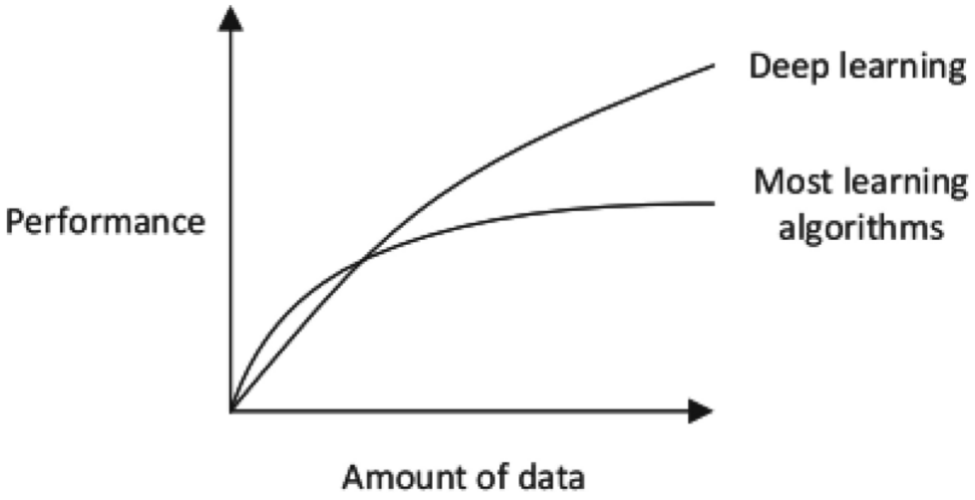

FIGURE 5. Deep Learning performance development over a large amount of data compared to most learning algorithms

Convolutional Neural Networks (CNNs) are a deep learning algorithm inspired by the structure of the human brain's visual cortex. They perform particularly well in image recognition tasks. CNNs detect low-level image features and gradually assemble them into higher-level features, resulting in highly accurate image recognition capabilities. (Peixoto 2020.)
Because image recognition is not within the scope of this study, the intricacies of a CNN architecture are not explained in more detail.

3.4.2 Transformers

After the general types of neural network architectures have been described, it is time to introduce the disruptor: the Transformers, a NN architecture developed specifically for Natural Language Processing (NLP). This architecture was first shown in the Attention is All You Need! paper by Google (Vaswani et al. 2017). Transformer variants dominated popular performance leader boards in almost every natural language processing task, proven to surpass conventional neural networks. Recent transformer-like architectures have become the state of the art in the computer vision field as well (Hristov 2022).
The Transformer architecture is a form of DL model due to its utilisation of layers, non-linear activation functions, and optimisation over a significant number of parameters.

The Transformer model is different from the FNN, CNN, and RNN structures that have already been introduced. As described in the original paper (Vaswani et al. 2017), it uses a technique called self-attention to make connections between words or syllables that come after each other. However, a Transformer model architecture does incorporate Feedforward Neural Networks (FNNs) in every layer of its design to handle information at each position in a sequence.

The concept of self-attention means that every word within a sentence obtains a weight according to its significance compared to other words.
The significance of a word is calculated by transforming each input sequence into queries, keys, and values. Each position in the sequence determines an attention score in conjunction with all other positions, indicating how much attention it should pay to them. These attention scores are used to weight the values, which are then added together to produce the result. Self-attention assists in the capture of long-term dependencies and improves the model's ability to comprehend and generate coherent and contextually relevant predictions. (Vaswani et al. 2017.)

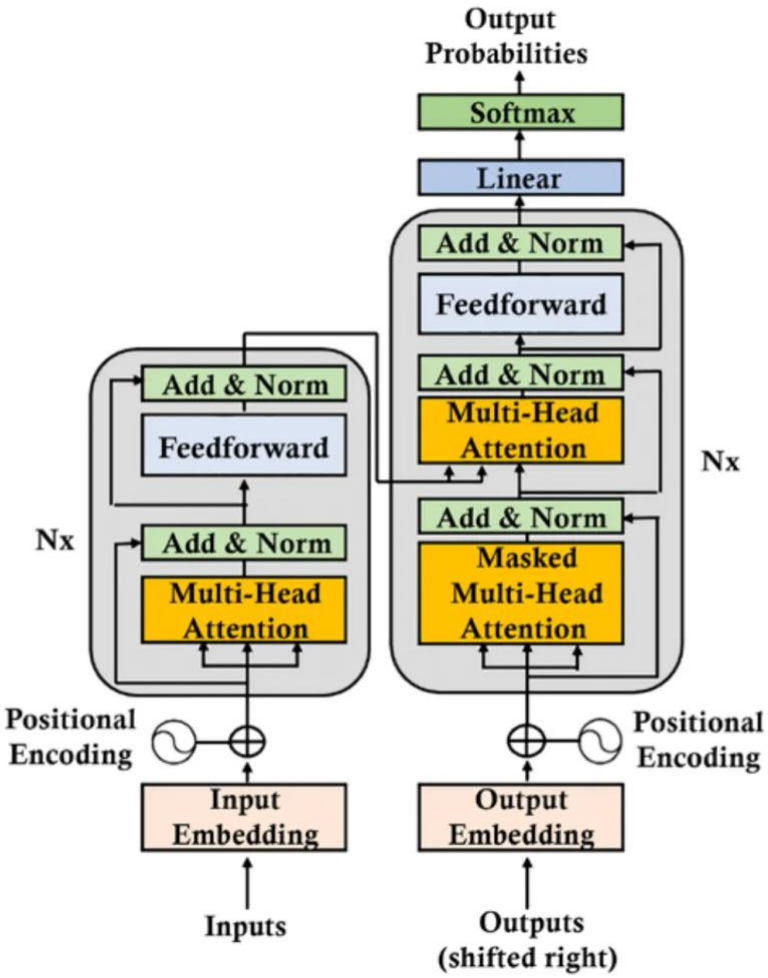

FIGURE 6. The Transformer model structure (Vaswani et al. 2023)

The structure of the Transformers model is next described based on Attention is all you need! (Vaswani et al. 2017).

The Transformer consists of an encoder and a decoder, both of which are made up of multiple layers of self-attention and feed-forward neural networks. The encoder takes in the input sentence and produces a sequence of hidden representations, while the decoder takes in the encoder's output and generates the final output sequence.

The encoder begins by taking in text that is broken down into units, words, or sub words, called tokens. These tokens are then converted into vectors using word embedding in the input embedding layer. To account for the position of each word in the sequence, a position-encoding layer is used. After that, there is a self-attention layer that captures the relationships and importance of each word to establish context. Following self-attention, a fully connected Feedforward Network (FFN) layer follows. This consists of two layers. What the FNN does is use a Rectified Linear Unit (ReLU) activation function to add non-linearity and help the model learn from the data. The FFN structure also allows for parallel processing of each position in the sequence, making it

computationally efficient. Each sublayer in the encoder, whether self-attention or Feedforward, is then followed by layer normalisation to ensure values remain within a manageable range for stable and faster training.
The multi-head attention mechanisms perform different tasks in each of the N=6 layers that make up the encoder stack. The embedding sublayer is only present at the bottom level.

Based on the input that the encoder has processed, the decoder is responsible for producing the output sequence. The structure of the decoder layer is like that of the encoder, with embedding and positional encoding layers at the bottom of the stack. In addition, the decoder has a third main layer called masked self-attention. This mechanism hides or masks the following words in the provided sequence, thus allowing the Transformer to predict the next word without seeing the sequence that follows the word in question.
The model creates an output sequence but handles each element individually in parallel. The output therefore looks like a list of elements, where each y represents the predicted output at a particular position in the sequence.

$$Output\ sequence\ =\ (y1, y2 \ldots yn)$$

The last two layers, linear and softmax, will give scores to the next likely items in the sequence. These scores are then turned into probabilities that are added to the output sequence.

3.5 Aspects to Machine Learning Applications

According to Domingos (2012), ML applications consist of three components: representation, evaluation, and optimization.
Representation is the act of transforming data into a format that is more easily understandable for the learning algorithm. Evaluation is the process of comparing the output of a learning algorithm with what was intended. Optimisation involves adjusting the weights and parameters of a learning algorithm to maximise its accuracy.
This chapter covers these different components in a little more detail while introducing the relevant aspects of each component that are part of training a DNN or fine-tuning a Transformers-based LLM for a multi-class classification task.

3.5.1 Data Representation and Pre-Processing

The representation aspect plays a critical role in NN models since it determines how data is encoded for the model's processing.

Pre-processing of numeric values includes normalisation. ML models need data normalisation to standardise numeric column values because their algorithms use mathematical calculations that are sensitive to the scale of the input data. The model's predictions can be biased if different numeric features, like temperature and price, have different ranges from one another.

By normalising the data, all features are similarly scaled without distorting their ranges. This allows the model to value each feature equally during learning. Larger features may otherwise dominate learning and model predictions. Normalising boosts model accuracy and performance. (Jaitley 2019.)

Many machine learning algorithms cannot handle categorical variables, such as names or labels, and they need to be encoded into numeric ones. There are many methods available for encoding such values. At its simplest form, encoding can look like assigning categorical values a unique number that the model can work with. Label encoding is used for nominal data where no meaningful order exists among the categories, and ordinal encoding is used for ordinal data where a meaningful order among the categories exists. According to Jarapala (2023), a method called one-hot encoding is most used for encoding categorical variables. It refers to the method of creating a binary column for each unique category in the variable. If the value matches the category in a column, it is assigned a 1; otherwise, it is a 0. By employing this encoding method, algorithms can treat these columns as features without assuming any natural order among categories that may not exist.

The disadvantage of one-hot encoding is that it might result in a high-dimensional dataset with categories that have many unique values. This might lead to issues with memory on a large dataset.

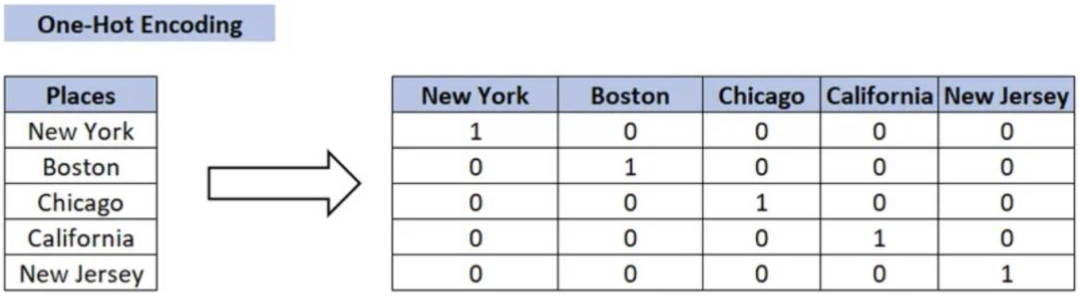

FIGURE 7. Example of one-hot encoding (Jarapala 2023)

For textual values in NLP, tokenization is a part of data preparation where textual information gets converted into a specific format, tokens, that the model can handle. The input embedding layers of Transformers-based LLMs show these tokens in a vector space, which shows how they are semantically connected. Text tokenization means breaking down text into smaller units. Those units are thereafter translated into numeric ID's that machines can understand. The way this is done depends on the model and its training.

A popular approach is to use sub-word-based tokenization, that aims to find a balance between breaking text down into characters and breaking it down into whole words. A character-level split would often end up in long sequences where individual tokens would be less meaningful. With word-level tokenization, on the other hand, less frequently encountered words would not be handled well. In an extensive vocabulary like this, different meanings would be assigned to words that are like each other. With sub word tokenization, when consecutive characters appear often enough, they are formed into a single token and added to the models' vocabulary. A word like "thing" might be an example of such in the English language. Less frequently occurring words are broken into smaller, more meaningful sub word units to ensure that the model can handle a range of words that weren't encountered during training. As opposed to word-level, the words "thing" and "things" would not be assigned different meanings after tokenization because the base of the word "thing" is recognised as a commonly used sub word. The final result of the tokenization depends on the training algorithm, the specifics of the training data, and the number of merge operations that are performed during the tokenization training. Popular sub word tokenization algorithms include WordPiece, Byte-Pair Encoding (BPE), Unigram, and SentencePiece. BPE is used in language models like GPT-3, and WordPiece, which is an extension of BPE, is the approach in BERT and its variants. (Khanna 2021.)

BPE was first introduced in the article “A new algorithm for data compression” (Gage 1994).
As Khanna (2021) explains it, BPE is applied in NLP by first splitting all text into characters and representing them with a numeric ID, a token. Then it looks for all possible scenarios where two tokens are found next to each other in the text, and the pair that is most often found together is merged into a single token that is added to the list of tokens in the text. Merging pairs that are often found together is key to the process of the BPE because, at its core, it is a data compression algorithm. The goal is to represent the text using the fewest number of tokens possible by repeatedly pairing and merging tokens. After multiple iterations of pairing and merging, common words are often represented with a single token, and less frequently encountered words are split into two or more sub words.

When looking at the example word “internal” and BPE tokenization, if BPE has encountered the sequences “in”, “ter”, and “nal” frequently during training, it could potentially split the word into the following tokens: [“in”, "ter”, “nal”].
However, depending on the training data or the number of merge operations used, BPE might also split the word differently, such as: [“inter”,”nal”] or [“in”,”ternal”].
In some cases, if none of the sub word sequences within "internal" were frequent enough in the training data, BPE might even split it into single characters.

WordPiece tokenization was first outlined in “Japanese and Korean Voice Search” (Schuster & Nakajima 2012). According to the Hugging Face Course (2022), the distinction between WordPiece and BPE is that when deciding what to merge, WordPiece looks for the token pair that is most often paired only with each other. At each iteration, WordPiece scores the token pair that is most frequently paired together by comparing the frequency of the pair against the frequencies of the individual parts of the pair combined. The scoring formula for deciding which pair to merge can be represented as follows:

$$score = \frac{freq\_of\_pair}{freq\_of\_first\_element \times freq\_of\_second\_element}$$

As an example, the frequency of occurrence of the word “friendship” is evaluated against the combined frequencies of the individual words “friend” and “ship”. The pair is merged only when “friendship” occurs more often than the word “friend” and the suffix “-ship”.

3.5.2 Model Evaluation

Evaluating the performance of a machine learning model is part of the training process. It involves testing the accuracy of its predictions and classifications. By comparing the model's results to known truths, it is possible to pinpoint areas that need improvement and affirm successful learning.

Evaluations of prediction accuracy are done continuously during training to calculate loss values for training and validation.
Loss is a quantitative measure of how far the model's predictions are from the actual values. It's a function that the algorithm aims to minimise during the training process.
Calculating loss provides a tangible metric to understand the model's performance and guide the optimisation process to achieve better accuracy. (Riva 2021.)

Performance metrics are the tools used to measure the performance of the model in various aspects. In this study, accuracy (formula 8) and a weighted F1 score are the evaluation metrics used. The F1 score is a more nuanced metric than the mere accuracy of predictions, and by (Keldenich 2021), it is one of the most used metrics among data scientists.

$$Accuracy = \frac{Number\ of\ Correct\ Predictions}{Total\ Number\ of\ Predictions}$$

Accuracy, as formula above explains, calculates the proportion of correctly predicted classification labels out of the total predictions.

$$Precision = \frac{Correct\ results\ in\ a\ class}{All\ predictions\ as\ class}$$

$$Recall = \frac{Correct\ results\ in\ class}{All\ that\ should\ have\ been\ identified\ as\ class}$$

$$F1\ Score = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

The F1 score is the harmonic mean of two metrics, precision, and recall, as depicted in prior formulas. A weighted F1 score calculates metrics for each label and finds their average weighted by support (the number of true instances for each label). This is useful for unbalanced datasets,

because when there are many items in one class and fewer in others, a good performance with non-weighted accuracy might mean that the model accurately categorises the largest class items to a notable degree while performing badly with the smaller classes (Kundu 2022).

In other words, accuracy simply indicates the frequency of correctness, while the F1 score provides insights into how often precision was correct when it was believed to be correct and how often it was incorrect when it should have been correct.
Accuracy performs effectively in situations where there is a balance, with each outcome being equally common and significant. However, when certain outcomes are less frequent or carry more importance than others, the F1 score can offer a better evaluation of the models' performance (Gurusamy 2018).

Loss values are the key measures that indicate if the model is possibly under- or overfitting.
Overfitting, a common challenge in ML, refers to a situation where a model performs well on the training data but poorly on unseen or validation data. This means that the model ends up being too complex during training, resulting in a loss in its generalisation capability. This is the situation where the model has learned too much about the specifics of the training data, so much so that irrelevant noise is included, and the learnings cannot be applied to unseen data. (Brownlee 2016.)

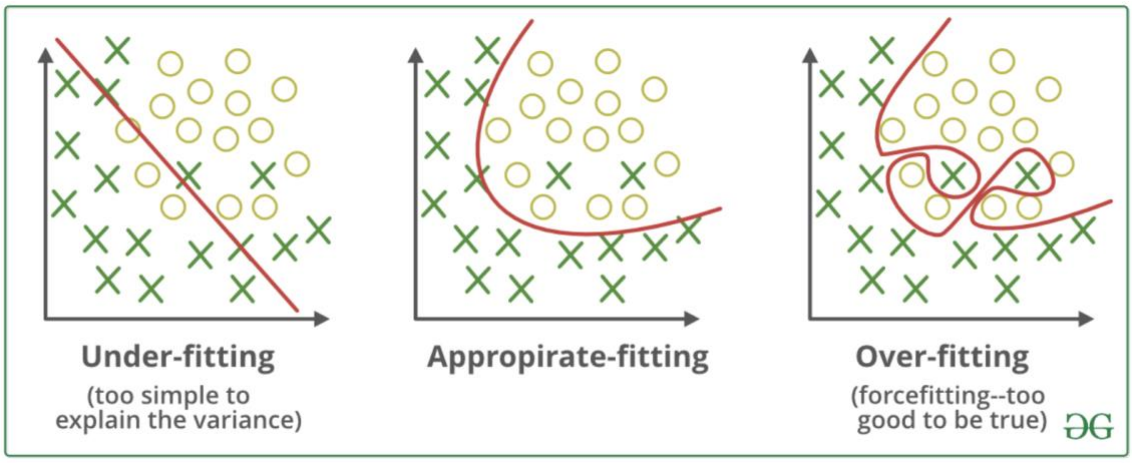

FIGURE 8. Overfitting and underfitting of ANN models (Géron 2019)

A clear indicator of overfitting is when there's a significant discrepancy between training and validation performances. The point in training iterations where training loss values keep declining but validation loss hinders, or rises is often the breaking point at which the model is getting too complex for generalisation. Early stopping is a technique to mitigate overfitting (Brownlee 2019). The training can be continued up to a certain number of iterations until new iterations improve the

model, as depicted in figure 9 After that point, the model's ability can exhaust itself as it begins to overfit the training data.

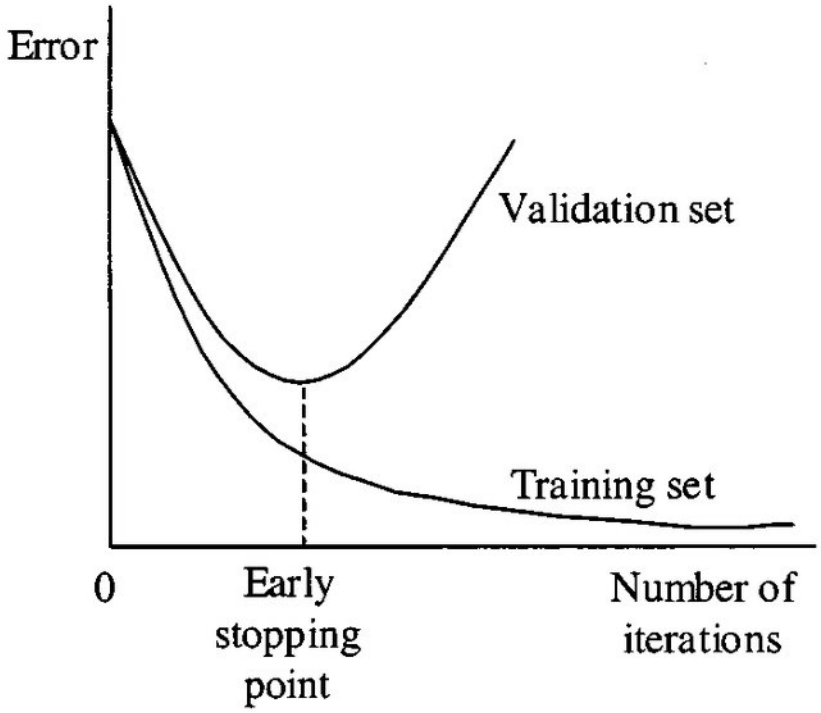

FIGURE 9. Overfitting and early stopping (Gençay & Qi, 2001)

3.5.3 Model Optimization

Optimising a neural network for the best possible performance is performed iteratively during training. This means that the steps of forward pass, loss calculation, backward pass, and weight updates are done repeatedly until the loss reaches a minimum value or a certain number of times is reached. Backpropagation adjusts weights to minimise prediction error and improve model predictions over time. (Gad 2022.)

AdamW is a common optimization technique in Transformers. The "W" in AdamW stands for "Weight decay fix." Traditional weight decay can shift the optimal positions of weights, but with AdamW, the weight decay is decoupled from the optimization steps, making it a better fit for adaptive learning rates (Loshchilov & Hutter 2019).

Optimising can be explained by referring to the process of training a model to try to find a location in the wilderness without being able to see. The optimizer acts as the guide, telling which way to go and how big of a step to take, all with the goal of getting to the final location as fast and efficiently as possible. DNNs, including transformers, are complex models (like intricate terrains). So,

choosing the right guide, in this case, the right optimizer, is crucial to navigating them well. As while travelling in the wild, one might get tired (compared to a model becoming excessively intricate or tailored to the data it has encountered), "weight decay" serves as a guideline that advises taking small steps to prevent exhaustion. However, with some optimizers like the original Adam, this can result in misguided decisions. The modification denoted by the "W", in "AdamW" signifies an improvement that addresses this issue, ensuring that one is not led astray.
The idea of being decoupled from optimisation steps means that taking steps over time (weight decay) doesn't conflict with the strategy of navigating to where one's headed. So even if one is conserving energy by taking smaller steps, the guide (AdamW) is still effectively guiding towards the destination.

When the model's weights are adjusted, the hyperparameter learning rate plays a role in deciding how much adjustment should be made to the model based on the error. Instead of adjusting the weights by the full amount, the learning rate value is used to set the proportion of the error to be included in the weight shift. Choosing the right learning rate can be challenging, as going too low might result in prolonged training with no progress, whereas choosing a value that's too high could lead to rapid but suboptimal weight learning or instability. (Brownlee 2019.)

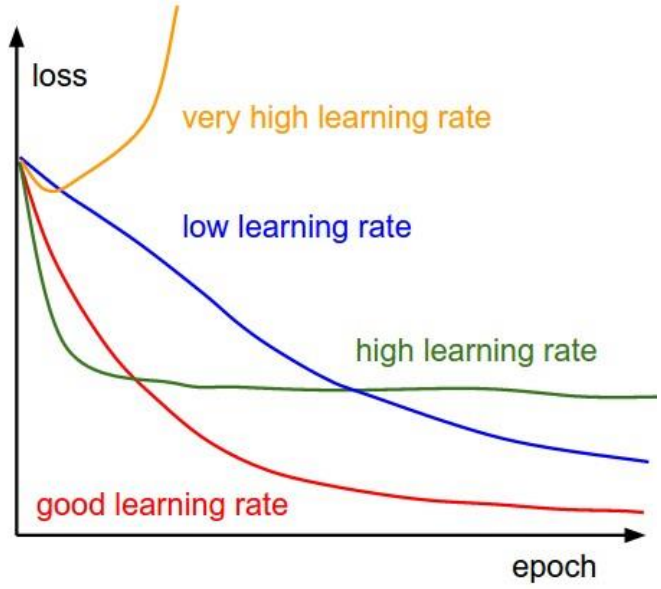

FIGURE 10. The effects of different learning rates on loss during training (Stanford University)

As described by Li (2022), adaptive learning rates with schedulers offer the benefits of starting with a higher learning rate and gradually reducing it as the training progresses. Learning rate schedulers are used to adjust the learning rate during training based on the training progress. To get the weights suitably adjusted early in the training, the learning rate is first set to a higher value. As

training progresses through iterations, the rate is gradually diminished to a smaller value for fine-tuning these weights to get better accuracy.

Adaptive learning rates can be thought of as the guide being able to adjust their strategy based on how the terrain feels. When feeling confident, they might encourage travellers to take bigger leaps. When they're uncertain, they might advise one to proceed cautiously.

3.6 Large Language Models

Thanks to breakthroughs in machine learning techniques, notable progress has been recently made in NLP. Large Language Models (LLMs) are cutting-edge text processing and generation systems with the goal of interacting meaningfully with human languages (Y Arcas 2022, 183-197). Recently, LLMs have developed to such maturity that they exhibit impressive levels of accuracy when performing various language-related tasks.
Large Language Models are advanced AI systems that can understand and produce language that sounds a lot like human speech. They have already been trained with large amounts of text data and unsupervised learning techniques. They are typically (but not always) derived from the Transformer architecture (Ozdemir 2023, chapter 1).

LLMs are invaluable in the development of tools for answers, text summarization, sentiment analysis, and language translation.
They are often developed by organisations or researchers with substantial computing resources and expertise on the subject matter. Openly available LLMs have played a key role in making AI more accessible to both developers and individuals without specialised knowledge, allowing them to create sophisticated computer interactions. Unless someone is willing to invest substantial time and effort into building their own model, these pre-trained models render the process unnecessary. Instead, readily adjustable models, like BERT, can be utilised and adapted to suit different needs, as they are mostly open for use and can be fine-tuned for specific tasks.

The success of LLMs and Transformers is due to the combination of several ideas. Most of these ideas have been around for years but were also actively researched around the same time. The combination of near-simultaneous breakthroughs with mechanisms such as attention, transfer learning, and scaling up neural networks, which provide the scaffolding for Transformers, resulted in a major impact on LLM development. (Ozdemir 2023.)

The progress made in AI and NLP with the easy accessibility of LLMs paired with these models' usability has played a significant role in their widespread adoption and increasing popularity across different natural language processing applications and AI development in general.
The following is an introduction to some notable Transformers-based LLMs in summary. While they often have a lot in common, there are some variations, for example, in pre-training objectives, training data and scale, fine-tuning approach, and task performance.

3.6.1 BERT

In 2018, Google published a pre-trained NLP model called Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al. 2019). BERT was the first model to bring bidirectional attention to transformer models, introducing a two-step structure consisting of pre-training and fine-tuning.

During pre-training, BERT goes through a massive corpus of unlabelled text. For pre-training, BERT uses two techniques: Masked Language Modelling (MLM) and Next Sentence Prediction (NSP). In MLM, BERT hides words at random and guesses what their tokens are based on the context, which lets it get representations for words that go both ways.
NSP trains BERT to predict if a second sentence follows the first.
During fine-tuning, additional downstream tasks are included in the BERTs training process. Using a pre-trained BERT model as a foundation, a model can be fine-tuned for various NLP tasks such as text classification, named entity recognition, sentiment analysis, and more. This adaptability allows BERT to enhance its performance for specific NLP tasks.

BERT's architecture uses only the encoder part of the Transformers network and uses it in such a way that multiple encoder networks are stacked one after the other. The encoder stack comprises multiple encoder layers in both variations of BERT: BERTBASE with twelve encoder layers and BERTLARGE with twenty-four.
Additionally, the dimensions of the BERT models are larger than those of the original Transformer model introduced previously. In conclusion, through bidirectional attention and pre-training on extensive amounts of data, BERT learns contextualised representations that can be further improved through fine-tuning for various NLP tasks.

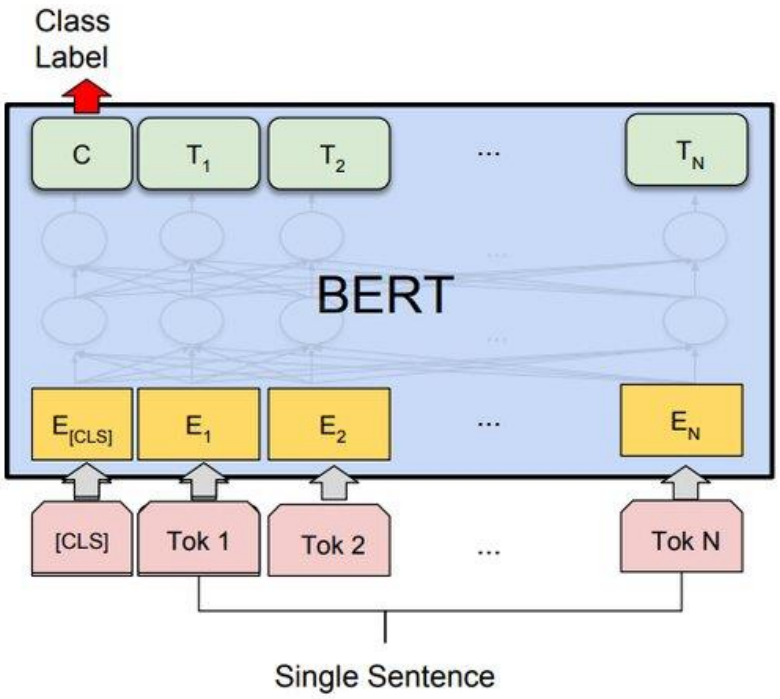

FIGURE 11. BERT model and a single sentence classification task (Devlin et al. 2019)

Bidirectional language modelling is a type of language modelling that uses both forward and backward passes through the text. This means that the model considers not only the words that come before the current word but also the words that come after it. (Rothman & Gulli 2022, chapter 3.)

When predicting the meaning of the word "always", BERT considers both "the grass is" and "greener" that follow it. This bidirectional approach helps the model understand the full context of the sentence and make more accurate predictions. It is especially useful for understanding the context of a sentence by considering the words that came before and after the sentence. This helps the model understand the meaning of words better while also improving accuracy in tasks like sentiment analysis, text classification, and machine translation.

The BERT approach has been modified into multiple variations, including RoBERTa, ALBERT, DistilBERT, and ELECTRA.

RoBERTa, short for Robustly optimised BERT, enhances the training methodology of BERT.

It achieves this by pre-training on a much larger scale with one thousand times more data and processing capacity. The Next Sentence Prediction (NSP) task is eliminated in RoBERTa's pre-training, and dynamic masking is implemented to alter the masked token during training epochs.

Furthermore, the training procedure is optimised by using larger batch training sizes. It is worth noting that RoBERTa uses a massive 160 GB of text for pre-training, including the 16 GB from

Books Corpus and English Wikipedia used by BERT. Despite some minor differences in sentence padding and beginnings and endings, RoBERTa's overall implementation remains relatively similar to BERT's. (Jain 2020.)

ELECTRA, which stands for "Learning an Encoder that Classifies Token Replacements Accurately", is a technique that aims to improve the BERT model through pre-training. This approach allows for superior performance on tasks while utilising fewer computational resources. Unlike BERT, ELECTRA introduces a task during pre-training where it detects replaced tokens in a given sequence. To accomplish this, ELECTRA utilises two Transformer models: a generator and a discriminator.

In the training phase, random tokens in the input sequence are replaced with [MASK] tokens. The generator then predicts the values of these disguised tokens. For the discriminators input sequence [MASK], tokens are substituted with generator predictions. The discriminator's role is to determine whether each token in the sequence is an original or a substitute.

The loss of the discriminator is determined by considering all the tokens in the sequence, unlike in MLM, where only the loss of masked tokens is calculated. This approach enhances performance. By discarding the generator model after pre-training, ELECTRA reduces memory and storage requirements, making it more practical for training language models on a single GPU. (Rajapakse 2020.)

3.6.2 XLNet

XLNet is another type of approach to large language models worth introducing. It gained popularity after outperforming BERT in around 20 NLP tasks, often by large margins. In terms of architecture, XLNet is quite comparable to BERT; the main difference lies in their approach to pre-training. XLNet is set apart from the above-mentioned models by its unique training objective called Permutation Language Modelling (PLM), which retains the benefits of AR language modelling while also gaining insights from the bidirectional approach.

Permutations refer to the organisation of objects in a specific sequence. When it comes to pre-training in XLNet, permutations are used to modify the arrangement of tokens within a sentence. Every imaginable permutation indicates a unique order for factoring the tokens. XLNet aims to comprehend information from both aspects of a token by predicting it with respect to the preceding tokens in each possible order, considering all potential word orders in a sentence. This empowers XLNet to grasp contextual meaning in both forward and backward directions while retaining the benefits of autoregressive language modelling. (Xiao, 2020.)

Referring to the example of “The grass is always greener”, XLNet in pre-training considers every possible order of the words in the sentence to predict the context, for example: “The grass is always greener", "The greener is always grass", "Always the grass is greener", and so on.

3.6.3 GPT-3 and GPT-4

GPT-3 (Generative Pre-Trained Transformer 3), a language model developed by OpenAI, was released in June 2020 (Brown et al. 2020). One of the key features of GPT-3 is its large size, boasting an impressive 175 billion parameters. This places it among the largest language models ever constructed. BERT-Large’s parameter amount is merely 2% of that at 340 million (Devlin et al. 2019).

GPT-3 is the third iteration in the series of Generative Pre-Trained Transformers (GPT) language models created by OpenAI and powers the popular free ChatGPT tool. GPT-3 received praise for its impact on NLP powered by AI. Its remarkable capability to produce text that closely resembles human writing, along with its versatility in performing language-related tasks, distinguishes it as a game changer in the field (jorge 2023).

While BERT is bidirectional in nature, GPT-3 is autoregressive (AR), which refers to generating text one word at a time by predicting the next word based on the words that have come before it (Ozdemir 2023, chapter 1). In other words, an autoregressive model would predict each word one by one based only on the preceding words.
When compared to the same sentence as mentioned before in the case of BERT, an autoregressive model like GPT-3 would predict "always" without considering the words that come after it, such as “The grass is”.

According to Brown et al. (2020), one impressive thing about GPT-3 is that it can do natural language processing (NLP) tasks without needing a lot of fine-tuning. It can effortlessly handle a wide range of NLP duties, including text completion, language translation, question answering, summarization, and programming.

Another advantage of using GPT-3 is its few-shot or zero-shot learning ability. Few-shot learning and zero-shot learning are techniques in machine learning that are commonly used in the field of natural language processing.

Few-shot learning refers to a method where a model is trained with very little data, even fewer than 20 examples per category, to recognise new categories in classification. This approach is particularly helpful in situations where it's challenging or costly to obtain large amounts of labelled data. Zero-shot learning, on the other hand, involves training a model to identify categories that it hasn't encountered during training, but the model is simply given a description of a class (Chandhok 2020).

Just as humans can recognize new objects they have never seen before if given some information or description about them, zero-shot learning aims to teach machines to do the same. For instance, if one has seen a bear but never seen a panda, one might still be able to recognise a panda from a description or by associating it with a bear.

Both techniques hold promise for enhancing the precision and efficiency of natural language processing tasks, like text classification and language translation.

OpenAI has made accessing GPT-3 user-friendly by providing an API that doesn't demand advanced software skills or in-depth AI knowledge. Almost anyone can make use of this powerful tool without too much complexity.

As the field of ML is quickly evolving, some major developments were published during the process of documenting this study. The latest release in the GPT-family, OpenAI released GPT4 in March 2023, along with a paper titled "Sparks of Artificial General Intelligence: Early Experiments with GPT-4" (Bubeck et al. 2023). GPT4 is said to be the most sophisticated LLM out there yet.

GPT-4 stands out as a language model that surpasses its predecessors in comprehending and producing natural language text. What makes it even more impressive is its ability to process images as inputs alongside data. GPT-4 has achieved a level of performance on academic exams that's comparable to that of humans, surpassing both current language models and state-of-the-art systems. It excels at understanding user intent and exhibits proficiency across languages. It is important to note that GPT-4 may still have limitations, occasionally making mistakes or providing incorrect information. As of the time of writing this thesis, GPT-4 is not publicly available yet, but there are a few ways to access it for free or at a cost, either by signing up for an account or purchasing credits, for example.

4 BUILDING A SOLUTION

The objective of this study is to fine-tune a pre-trained large language model combined with non-textual features for a specific task. This is done by putting together a deep neural network that can fine-tune a model for classification with supervised learning techniques. The pre-trained model used in this case has been trained with unsupervised methods. This is how transfer learning comes into the picture.

This chapter explains the process of building an example solution. Firstly, the tools and environments used in the research are introduced, followed by explanations on how the data was collected and formatted for the neural network. The features of the DNN are then introduced, and lastly, this chapter covers the training process and validation of the model's outputs.

4.1 Tools and Environments

The training and validation of the model are run locally on an Apple MacBook Pro with an Apple Silicon M2 chip. Recently, Apple introduced Mac's new Metal Performance Shaders (MPS) backend for GPU training acceleration (Apple Inc. 2023). This MPS backend extends the PyTorch framework, providing scripts and capabilities to set up and run operations on a Mac graphics processing unit instead of the CPU. According to the PyTorch documentation (2023), the new MPS device maps machine learning computational graphs and primitives on the MPS Graph framework and tuned kernels provided by MPS, which makes training models faster than if everything were done on a CPU alone.

4.1.1 Jupyter Notebook

Almost all data transformation and model training are done in a single, local Jupyter Notebook file. Jupyter Notebook is provided by Project Jupyter, a non-profit, open-source project. Jupyter Notebook is an interactive development environment for notebooks, code, and data. Its flexible interface allows users to configure and arrange workflows in data science, scientific computing, computational journalism, and machine learning. Jupyter Notebook is running a kernel that takes the code that is written, runs it, and then shows the result. This kernel can understand different programming languages, but Python is the most popular.

4.1.2 Python

Python is a high-level, general-purpose programming language. Much of its popularity can be credited to its versatility, particularly in machine learning, data analysis, and text processing, since Python offers a wide range of well-documented libraries and frameworks for ML. Python's code readability makes it a great choice for prototyping purposes. Since Python is interpreted at runtime, it allows for quick iterations in testing code and debugging. Previous experience with Python made it an easy choice for this research.

4.1.3 NumPy

NumPy is a fundamental toolkit for computation in Python. It offers tools for handling large multi-dimensional arrays and matrices, as well as a range of mathematical functions to manipulate these arrays.

4.1.4 Pandas

Pandas is a popular and sophisticated Python data analysis and manipulation toolkit. It includes data structures such as data frames and series for working with and analysing structured data. Pandas offers an array of functions for quickly generating pivot tables and performing cleaning and transforming operations on the data.
Many ML libraries within the Python ecosystem are designed to work with Pandas data structures. This makes it easier and more straightforward to transition from data pre-processing to modelling.

4.1.5 PyTorch

PyTorch is a deep learning framework for Python, primarily used in computer vision and natural language processing. PyTorch's strengths lie in its flexibility, which allows the integration of new data types and algorithms. The framework is known for its efficiency and scalability, as it minimises calculations and supports hardware architectures.

PyTorch provides tensors, which are data structures similar to arrays or matrices. The difference is that they operate on GPUs, which significantly enhance computational speed, making it possible to run operations on large datasets.
At the time of the initial training, the platform for training was an Apple Silicon Mac, which then supported PyTorch, but solutions for others like TensorFlow were not yet available.

4.1.6 Hugging Face Transformers

Hugging Face's Transformers is a popular library that makes state-of-the-art machine learning models easily available. With just a few lines of code, it is possible to download, use, and fine-tune a pre-trained large language model, like BERT. Alongside models, Transformers provides tokenizers, which convert text data into a format that the models can understand.

4.1.7 Scikit-learn

Scikit-learn, also referred to as sklearn, is a widely used and versatile machine learning library for Python. It provides efficient functionality for data mining and analysis. Scikit Learn is designed to work with NumPy and SciPy, two Python libraries used for scientific research and numerical computing, by building on their basic structure.

4.2 Methods and Data

This chapter introduces the selected method, model, and process of collecting and formatting the data for the model fine-tuning.

4.2.1 Selected Method

Even though in life it might not be easy to determine what one specific need is at play at once, a simple approach like that was thought to be the best way to begin this experiment. Therefore, a classification method was chosen rather than trying to build a regression model that, for example, would predict the portion of needs met on a scale or a matrix. The latter option was considered for a while, but labelling needs on a scale was quickly found to be too complicated to

perform, even for the team labelling the teaching material. Because it is desired to identify more than two categories of user needs, the fitting option was to go with a multi-class classification model. And like the user motive and need categories used in the organization's portfolio work, the aim is to define only one possible outcome category per sample. This means that multi-label classification was ruled out.
The chosen method was to use a pre-trained large-language model and fine-tune it for this specific task.

4.2.2 Model Selection

Finding the most suitable, well-performing, and easily available LLM for fine-tuning was the first step of this study. BERT was chosen for numerous reasons, one of which was the availability of language-specific models. The FinBERT model from TurkuNLP has been pre-trained on Finnish text. When working with text data for a classification task, using a model specifically designed for that language can result in improved performance (Virtanen et al. 2019). This is because the model would already have knowledge of the subtleties, idioms, and structures of the language. FinBERT has been pre-trained for 1 million steps on over 3 billion tokens (24B characters) of Finnish language culled from news, online debate, and web crawls. In comparison, multilingual BERT was trained using Wikipedia texts, with the Finnish Wikipedia material accounting for around 3% of the total quantity used to train FinBERT.
According to the paper "Multilingual is not enough: BERT for Finnish (Virtanen et al., 2019), when fine-tuned for Finnish natural language processing tasks, these features enable FinBERT to beat not only Multilingual BERT but also all previously suggested models.

The implementation period of this research was done in the autumn of 2022. At that time, BERT and its variants had achieved state-of-the-art performance on numerous NLP benchmarks. Fine-tuning BERT for specific tasks was said to have often led to significant improvements in performance metrics.
The design of BERT's Transformers architecture caters to tasks that require understanding the context of a word from both directions, from left-to-right and right-to-left, making it a great option for situations where context plays a key role. In contrast, some other models of the time might be more general, such as GPT-3, and primarily best suited for tasks that require generating text in a left-to-right manner.
Although BERT is mainly designed for working with text, its structure can be adapted to handle other types of data by following BERT with an MLP to combine BERT embeddings (which represent

text) with numerical and categorical data. This hybrid approach allows the model to make predictions using a broader range of information.

Even though the text content of the articles was thought to play a significant role in the classification of the user need, the main text content alone was not considered to be enough for an accurate classification. The Smartocto white paper (Smartocto, 2021) also mentions a whole list of parameters used in the approach implemented for the BBC World Service and others thereafter. Various features define and describe articles, such as which team published it, what are the keywords, how long is the article content, what time it was published, does it contain pictures or other media, etc. Combining these other factors into the classification meant that a multi-layer perceptron seemed like the best approach for this research case. With this architecture, textual contents are first processed with BERT, and the non-textual features are combined into the model as additional layers for classification. Figure 12 is a representation of a such implementation introduced in the paper "Multimodal-Toolkit: A Package for Learning on Tabular and Text Data with Transformers" (Gu & Budhkar 2021).

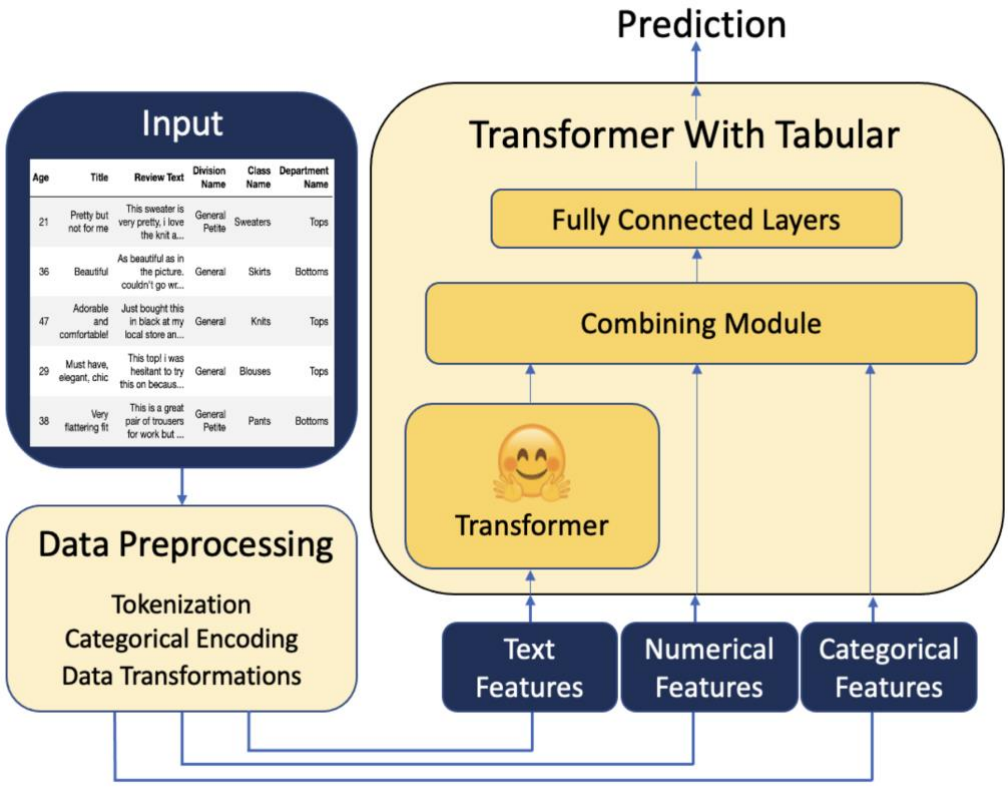

FIGURE 12. The framework of Multimodal-Toolkit (Gu & Budhkar 2021)

4.2.3 Data Collection

The data used in training and validation was collected from Yle's data warehouse, Amazon Redshift, with an SQL query. The initial dataset was then expanded to include the training labels for classification. The classification was done based on Yle News Lab's Audience Insight Team's analysis of each sample.

Labelling user needs for articles turned out to be a difficult task because of a lack of definite rules that determine the user need class for each article, even though the organisation's user motive categories were used as a guideline. During the labelling process, the different class names and their definitions were discussed and rewritten numerous times. In the final rounds of labelling, the number of classes was diminished to three in order to come up with an efficient labelling system that produced seemingly enough training material for the model.

TABLE 4. Final classification labels that were formed by combining two smallest classes into one

Original Need Category	Condensed final categories for the model
Need for Information	1. Information
Need for Understanding	2. Understanding
Need for Personal Connection	3. Other: emotional, social/community, identity/self-reflection, learning
Need for Connecting to Others / Need for Experiences	3. Other: emotional, social/community, identity/self-reflection, learning

The last class is a combination of the final two classes, broadened to include anything else not fit for the first two. This was done because in the labelled dataset, there was such a small number of samples in the last two classes, the fourth one particularly, that it was considered beforehand to be impossible for a machine to learn from.

The first dataset for labelling was created with a randomised query, selecting random articles from within a given timeframe, but it was soon realized that to get even a little bit more balance between the different classes, manually selecting more rows that fit into some of the more underrepresented

classes was required. This was done by adding rows that represented some of the most popular content by article page views at that time.

The labelled dataset was created using a Google Sheet, which was later downloaded as a local CSV file for processing. CSV was chosen as a format because it is an easy-to-use, popular format for data storage and exchange. Unlike many other formats, CSV files can be opened and viewed in simple text editors, making it easy to inspect the data. CSV files have a simple structure, making the generation and parsing of them easy.

4.2.4 Data Description

The dataset contains detailed information about news articles published on Yle.fi, including meta-data (like publishing time and word count) and content details (like title, lead text, and main text content).

TABLE 5. Contents of the dataset

Field Name	Description
article_yle_id	A unique identifier for the article
title	The title or headline of the article
organizations	A comma-separated list of organisations associated with the article, where organisation refers to a specific journalistic team, such as politics or culture or a specific region in Finland.
pagetype	The type of page determining its contents, e.g. “newscontent-sport”, “newscontent-news”, or blank for content not from the news department
lead_text	A lead or introductory text of the article
concepts	A comma-separated list of keywords associated with the article
contenttype	The content type of the article, e.g. “article” for a basic article, “card_story” for Instagram-story-like content or

	"livefeed" for an update-based format for a breaking news story
media_type	Describes the type of media associated with the article. "Image", "video"
media_caption	A caption or description of the associated media, if any
text_content	The main body, or content, of the article
words	The word count of the article
characters	The character count of the article
Weekday of published_time	The day of the week when the article was published
Hour of published_time	The hour of the day (in 24-hour format) when the article was published
all_updates	Indicates the number of updates or edits made to the article after its initial publication.
User need	The purpose or need the article fulfils for the reader is the target for predictions.

Because the labelling was not a straightforward but a time-consuming task for the team, the size of the final dataset for this case remains modest: it includes 583 articles altogether. The representation for each class remains unbalanced, but that was to be expected, as the expectation is that there would be even less of a balance in real life.

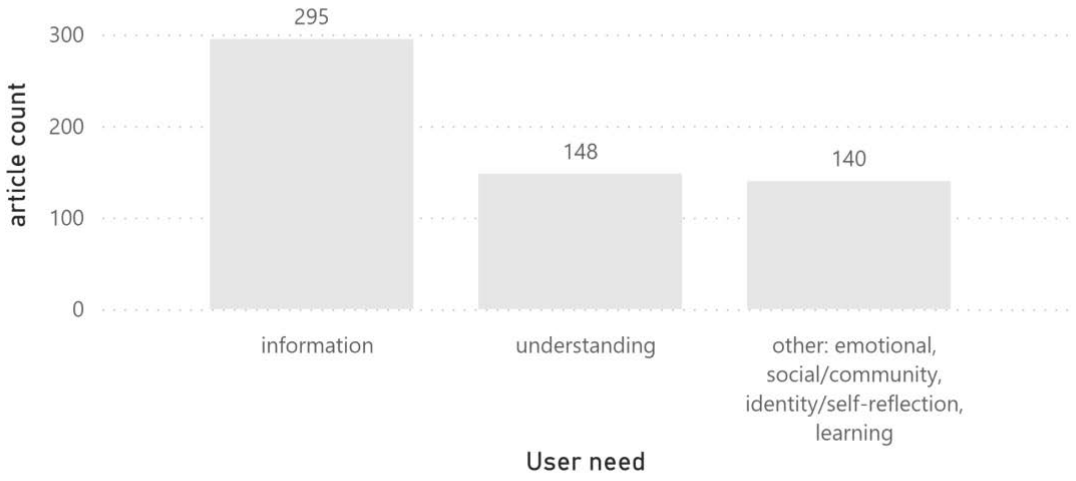

FIGURE 13. The division of samples in classes

4.2.5 Pre-Formatting Data

The pre-formatting steps involve importing the data, cleaning, formatting, and encoding the data.

Firstly, the data was loaded onto a Pandas data frame from the CSV file. When loaded into a data fame, the data is stored in memory, allowing for fast operations.
A Pandas data frame ensures an environment that keeps the data structure in place during preparation operations. All data manipulation hereafter was done with the Pandas data frame.

Large datasets often contain inconsistencies or missing values that make analysis or modelling challenging. Therefore, the next step was to clean the contents. If some fields were blank in the dataset, the blanks were replaced either with a number 0 or a categorical value of “other”.

All the numerical columns in the data frame are modified to make their distribution closer to a normal, Gaussian distribution. This step is taken because when all numeric input variables follow a similar scale, machine learning algorithms tend to perform better (Alam 2020). The normalisation was done using Quantile Transformer, which provides a non-linear transformation to ensure that the probability density function of each feature will be mapped to a desired output distribution.

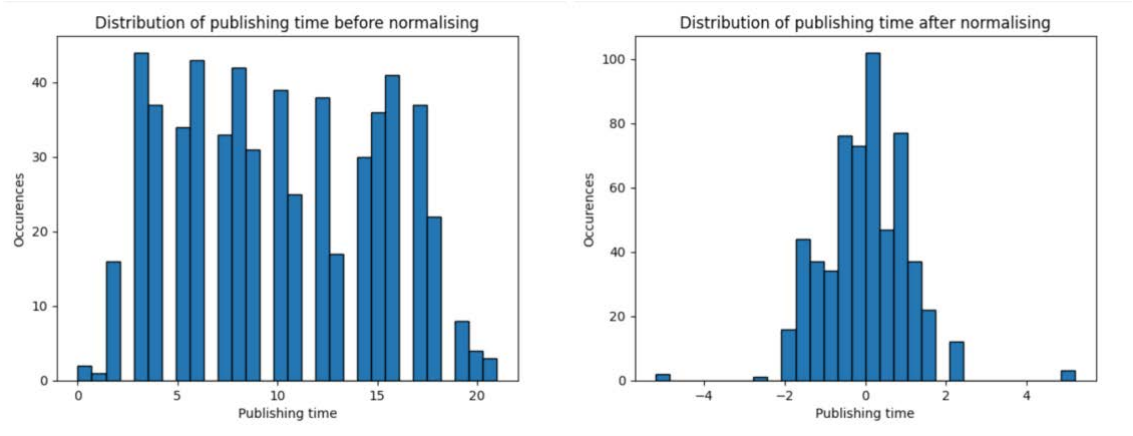

FIGURE 14. Publishing times before and after normalizing

Next, the different columns in the data frame were categorised into different types of data. This is done to make operations on different kinds of data as easy as possible. The categories of columns that describe the features of the article are textual columns, numerical columns, and categorical columns. Additional categories are created for columns that are required for the classification task: the target column of user need and the data type of train or test. The numerical columns contain values that can have an unforeseen number of unique numerical values in a range. The categorical columns contain a predefined set of unique values that one article can be classified as.

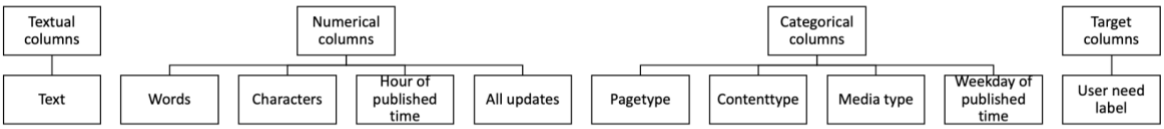

FIGURE 15. Classification of columns in the dataset

The target labels were encoded next. The label encoding produces a mapping of classes with numerical values, which are stored in a Python dictionary data structure:

```
{'information': 0,
 'understanding': 1,
 'other: emotional, social/community, identity/self-reflection, learning': 2}
```

The categorical values were then encoded into a binary matrix using one-hot encoding. After this step, instead of there being four categorical columns as described in figure 15, there ended up being a column for each of their unique values combined, resulting in 18 binary columns. Other methods for encoding categorical variables were tested during the process of this research, but one-hot encoding, however, seemed to lead to the best results in model performance.

```
col_info = {}

col_info['text_cols'] = ['text']
col_info['num_cols'] = ['words','characters','Hour of published_time','all_updates']
col_info['cat_cols'] = ['pagetype_newscontent-news',
 'pagetype_newscontent-sports',
 'pagetype_other',
 'contenttype_article',
 'contenttype_article_feature',
 'contenttype_card_story',
 'contenttype_livefeed',
 'contenttype_mobile_feature',
 'media_type_image',
 'media_type_none',
 'media_type_video',
 'Weekday of published_time_Friday',
 'Weekday of published_time_Monday',
 'Weekday of published_time_Saturday',
 'Weekday of published_time_Sunday',
 'Weekday of published_time_Thursday',
 'Weekday of published_time_Tuesday',
 'Weekday of published_time_Wednesday']
col_info['target_cols'] = ['label']
col_info['split'] = ['data_type']
```

FIGURE 16. List of column types after one-hot encoding

Weekday of published_time_Friday	Weekday of published_time_Monday	Weekday of published_time_Saturday	Weekday of published_time_Sunday	Weekday of published_time_Thursday	Weekday of published_time_Tuesday	Weekday of published_time_Wednesday
1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0
1.0	0.0	0.0	0.0	0.0	0.0	0.0
0.0	0.0	0.0	1.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	0.0	1.0	0.0

FIGURE 17. Example of 5 sample rows displayed by one-hot encoded weekday of publishing

The following step was to prepare the textual values for analysis. The different text columns were combined into a single column for sequence analysis. In BERT, the input text has a sequence length limitation. The original BERT models are designed to handle up to 512 tokens in a sequence; this is a result of the architecture and design principles of the transformer model. This limitation was, at first glance, thought to be a setback since many of the articles are lengthy in content. However, it was thought reasonable that other characteristics may already tell the model enough about the article, even though not all the text is included. Therefore, all relevant textual information was placed in one sequence back-to-back, making sure that the

most important items were included in the beginning, and the rest of the sequence was the article text content itself. BERT truncates the length of the sequence to the limited 512 tokens, thus leaving some of the article content out in most cases, but it was believed that the first 512 tokens might contain enough information in order for BERT to understand the context and style of the article.

Some special article content types don't allow for text content to be stored in the same way as others, and therefore not all samples in the dataset have any text content in the main_text. All articles, however, have titles, and most have lead texts. Therefore, the following rule was introduced:
If the text content is empty, combine the title, lead text, media caption (if any), organisations, and concepts. If the article has text in the main text, the lead text is already included, and therefore the contents are organised as follows: title, organisations, concepts, text content, and lastly, the possible media caption. A separation token, [SEP], is added between the different text input columns at this stage. This token is later used by the BERT text encoder to distinguish between different parts of the combined sequence (Hugging Face).

In case of an empty main text content, the input sequence is as follows:

$$[title] + [SEP] + [lead\ text] + [SEP] + [media\ caption] + [SEP] + [organisations] + [SEP] + [lead\ text] + [SEP] + [concepts]$$

When the main text is available, the sequence is formed like the following:

$$[title] + [SEP] + [organisations] + [SEP] + [concepts] + [SEP] + [text\ content] + [SEP] + [media\ caption]$$

Once the text was combined, the source text columns were removed from the data frame.

The text contents were not modified in many other ways because BERT is designed to work with raw text without the need for extensive pre-processing like stemming or lemmatization. Throughout the course of pre-training, BERT acquires representations for words, sub words, and even characters across various contexts. As a result, transfer learning benefits from BERT's pre-learned ability to comprehend differences in word forms, tenses, and other linguistic variations without necessitating standardization (Lendave 2021). The text was, however, cleaned to not include some types of content, like URLs. Even if it was not to make it more understandable for BERT, they were

not thought to be a significant feature of the content, therefore merely taking space from more relevant content. Some of the HTML tags and annotations, like bolded text, for example, were found to be beneficial for the prediction outcomes.

At this point, the dataset was split into training and validation datasets.
The simplest way to create a validation set would be to randomly select some instances from a large dataset, typically 20% of the original dataset. In statistics, this method is known as Simple Random Sampling. Random sampling is generally suitable if the original dataset is large enough; otherwise, sampling error introduces bias. Stratified Sampling is a sampling method that reduces sampling error by performing random sampling for each class separately. The method ensures that each group within the dataset receives appropriate representation in the sample, making the test set more representative of the overall population. This is especially needed for imbalanced datasets, where certain classes or instances may be more frequent than others. (Baeldung 2021.)

Scikit-Learn's train_test_split function was used to create the division. Because the original dataset was limited in size, the division of the split was critical to work both ways: ensure there is as much training material as possible, but also enough cases to validate against. The validation size of 18% was selected, resulting in the following division: train with 478 articles, validate with 105.

After the datasets had been split, the prepared data was ready to be moved into tensors. Tensors are a fundamental data structure in ML and DL frameworks. They are mathematical objects that generalise scalars, vectors, and matrices to higher dimensions. While data frames and tensors are both capable of representing multidimensional data, the difference is that data frames are specifically designed to be two-dimensional, consisting of rows and columns. While data frames support a variety of operations, they are more oriented towards data manipulation than mathematical operations on large datasets. Tensors enable calculations on large datasets, which are required for training neural networks, and can be run on hardware accelerators like GPUs.

Categorical and numerical columns each require their own tensor because, while data frames can hold different data types in one data frame, tensors are homogeneous. Separate tensors are created for each column type's training and testing datasets as well. As a result, there are four different tensors for the non-textual features.

The text contents are to be encoded into tokens that the BERT model knows.

The tokenization algorithms introduced in chapter 3.5.1 are used when LLMs are pre-trained. When a model like BERT is fine-tuned for a specific task, the encoding of the textual contents into token ID's is done based on the pre-trained corpus of the model, and the contents of the fine-tuning inputs do not affect the way the text is split into sub words (Au Yeung 2020).
When text inputs are fed to a pre-trained BERT tokenizer, the WordPiece algorithm will first try to match whole words from its pre-trained vocabulary. If it doesn't find a match, it breaks down the word into smaller sub words or characters that exist in the vocabulary. This greedy longest-match-first strategy to tokenize a single word is also known as maximum matching, or MaxMatch, and has been used for Chinese word segmentation since the 1980s (Song & Zhou, 2021).

Hugging Face's Transformers library's function batch_encode_plus is used for encoding the text. This method, as its name suggests, can process batches of text data, not just individual sequences.

```
encoded_data = tokenizer.batch_encode_plus(
    df.text.values,
    add_special_tokens=True,
    return_attention_mask=True,
    padding = "max_length",
    truncation = True,
    max_length=512,
    return_tensors='pt'
)
```

In the excerpt above, the encoder function is given parameter values on how to encode the data. Setting "add_special_tokens" to true tells the tokenizer to add its own identifiers for the encoded text. According to the BERT tokenizer documentation (Hugging Face), these special tokens include the likes of [CLS], which tells the model to classify the whole sequence of tokens that follows it instead of per-token classification, and [PAD], which is for padding the length of the sequence to its set maximum length when needed.
To maintain consistent length, the tokenized texts are either padded or truncated accordingly. In this case, the maximum sequence length of 512 tokens is selected, and the text contents that would exceed the 512 token limit are left out. If the text would be under the limit of 512 tokens, the special [PAD] token is used for the remaining tokens to keep the length consistent. For this reason, attention masks are created to help the model distinguish between actual tokens and possible padding tokens in the input.

An example article’s text content is displayed as tokens, token IDs, and attention mask values in figures 18 and 19. The example is a short article about the EU’s plans to unify mobile device chargers. The main text content consists of only 194 words, and when the text input sequence is combined with the title and keywords, the length of the input sequence is 233 words.

These 233 words, plus punctuation and other characters (excluding spaces), are morphed into 404 tokens that are found in TurkuNLP’s pre-trained FinBERT. Because the sequence length is set to 512, the encoder function adds 108 [PAD] tokens to fill in the remaining tokens.

```
Original Text: EU-komissio haluaa vihdoin yhdenmukaistaa kännykkälaturit – parlamentti esitti asiaa jo kymmenen vuotta sitten.[SEP]1418-maa
ilma[SEP]ulkomaat,Matkapuhelimet ja mobiililaitteet,Euroopan unioni,Elektroniikkajäte,Euroopan komissio,Langattoman latauksen teknologiat,M
obiililaitteiden laturit.[SEP]Taustalla ovat ympäristösyyt ja kuluttajaystävällisyys. Komission esitys vaatii jäsenmaiden ja EU-parlamentin
hyväksynnän. Elektroniikkayhtiöt saisivat kahden vuoden siirtymäajan, jos standardisointi toteutuu. Elektroniikkavalmistajista Apple vastus
taa hanketta. EU-komissio haluaa stantardisoida matkapuhelinten ja muiden kannettavien elektroniikkalaitteiden laturit. Komissio kertoi tän
ään esityksestään. Taustalla on huoli elektroniikkaromun kasvavasta määrästä: EU:ssa on arvioitu, että vanhoista latureista syntyy vuositta
in 51 000 tonnia jätettä. Yhtenäiset laturit olisivat kätevämpiä myös kuluttajien kannalta. On arvioitu, että kuluttajat säästäisivät stand
ardilaturien ansiosta vuosittain 250 miljoonaa euroa. Europarlamentaarikot ovat arvostelleet komissiota hidastelusta asiassa. Parlamentti h
alusi jo kymmenen vuotta sitten yhtenäistää laturit. Komissio esittää standardiksi USB-C-liitäntää. Esityksen mukaan laturit myytäisiin jat
kossa erillään laitteista. Samanlaisella laturilla voisi ladata eri valmistajien laitteita. Samaan aikaan langaton latausteknologia on kuit
enkin yleistymässä. Komission esitys vaatii jäsenmaiden ja EU-parlamentin hyväksynnän. Elektroniikkayhtiöt saisivat kahden vuoden siirtymää
jan, jos standardisointi toteutuu. Standardilaturi olisi takaisku Applelle, joka on vastustanut hanketta. Jo vuonna 2009 Nokia, Apple ja Sa
msung allekirjoittivat sitoumuksen yhteensopivan mikro-USB-lataustekniikan kehittämisestä. Apple on kuitenkin pitäytynyt tuotteissaan erila
isissa liittimissä ja komission lakiesityksen arvioidaankin vaikuttavan siihen eniten valmistajista. _Korjaus 23.9.2021 kello 16.30: Artikk
elissa on täsmennetty, että Applen osalta kyse on erilaisista liittimistä, ei latureista._ **Lisää aiheesta:** [Vanha vaatimus yleislaturis
ta junnaa yhä EU:ssa – langaton latausteknologia voi tehdä vuosien piuhakeskustelusta turhan ] haluaa yhtenäistää kännyköiden laturit.
Original Text Lnegth: 2205

Tokenized version of input_ids:
['[CLS]', 'EU', '-', 'komissio', 'haluaa', 'vihdoin', 'yhden', '##mukaista', '##a', 'kännykkä', '##lat', '##urit', '–', 'parlamentti', 'esi
tti', 'asiaa', 'jo', 'kymmenen', 'vuotta', 'sitten', '.', '[SEP]', '141', '##8', '-', 'maailma', '[SEP]', 'ulkomaat', ',', 'Matka', '##puhe
limet', 'ja', 'mobiilila', '##itteet', ',', 'Euroopan', 'unioni', ',', 'Elektron', '##iikka', '##jäte', ',', 'Euroopan', 'komissio', ',',
'Lang', '##attoman', 'lata', '##uksen', 'teknologiat', ',', 'Mobiil', '##ila', '##itteiden', 'lat', '##urit', '.', '[SEP]', 'Taustalla', 'o
vat', 'ympäristö', '##syyt', 'ja', 'kuluttaja', '##ystävä', '##llisyys', '.', 'Komission', 'esitys', 'vaatii', 'jäsenmaiden', 'ja', 'EU',
'-', 'parlamentin', 'hyväksynnän', '.', 'Elektron', '##iikka', '##yhtiöt', 'saisivat', 'kahden', 'vuoden', 'siirtymä', '##ajan', ',', 'jo
s', 'standardi', '##so', '##inti', 'toteutuu', '.', 'Elektron', '##iikka', '##valmista', '##jista', 'Apple', 'vastustaa', 'hanketta', '.',
'EU', '-', 'komissio', 'haluaa', 'sta', '##nt', '##ardi', '##so', '##ida', 'matkapuhelin', '##ten', 'ja', 'muiden', 'kanne', '##tta', '##vi
en', 'elektroniikka', '##laitteiden', 'lat', '##urit', '.', 'Komissio', 'kertoi', 'tänään', 'esityksestä', '##än', '.', 'Taustalla', 'on',
'huoli', 'elektroniikka', '##rom', '##un', 'kasvava', '##sta', 'määrästä', ':', 'EU', ':', 'ssa', 'on', 'arvioitu', ',', 'että', 'vanhoist
a', 'lat', '##ure', '##ista', 'syntyy', 'vuosittain', '51', '000', 'tonnia', 'jätettä', '.', 'Yhtenä', '##iset', 'lat', '##urit', 'olisiva
t', 'kätevä', '##mpiä', 'myös', 'kuluttajien', 'kannalta', '.', 'On', 'arvioitu', ',', 'että', 'kuluttajat', 'säästä', '##isivät', 'standar
di', '##lat', '##urien', 'ansiosta', 'vuosittain', '250', 'miljoonaa', 'euroa', '.', 'Euro', '##parlamen', '##taar', '##ikot', 'ovat', 'arv
ostel', '##leet', 'komissio', '##ta', 'hidas', '##telusta', 'asiassa', '.', 'Parlamentti', 'halusi', 'jo', 'kymmenen', 'vuotta', 'sitten',
'yhtenäistä', '##ä', 'lat', '##urit', '.', 'Komissio', 'esittää', 'standardi', '##ksi', 'USB', '-', 'C', '-', 'liitän', '##tää', '.', 'Esit
yksen', 'mukaan', 'lat', '##urit', 'myytä', '##isiin', 'jatkossa', 'erillään', 'laitteista', '.', 'Saman', '##laisella', 'lat', '##urilla',
'voisi', 'ladata', 'eri', 'valmistajien', 'laitteita', '.', 'Samaan', 'aikaan', 'langaton', 'lataus', '##teknologia', 'on', 'kuitenkin', 'y
leisty', '##mässä', '.', 'Komission', 'esitys', 'vaatii', 'jäsenmaiden', 'ja', 'EU', '-', 'parlamentin', 'hyväksynnän', '.', 'Elektron', '#
#iikka', '##yhtiöt', 'saisivat', 'kahden', 'vuoden', 'siirtymä', '##ajan', ',', 'jos', 'standardi', '##so', '##inti', 'toteutuu', '.', 'Sta
ndard', '##ilat', '##uri', 'olisi', 'takaisku', 'Apple', '##lle', ',', 'joka', 'on', 'vastustanut', 'hanketta', '.', 'Jo', 'vuonna', '200
9', 'Nokia', ',', 'Apple', 'ja', 'Samsung', 'allekirjoitti', '##vat', 'sitoum', '##uksen', 'yhteensopiv', '##an', 'mikro', '-', 'USB', '-',
'lataus', '##tekniikan', 'kehittämisestä', '.', 'Apple', 'on', 'kuitenkin', 'pitä', '##ytynyt', 'tuotteissa', '##an', 'erilaisissa', 'liitt
i', '##missä', 'ja', 'komission', 'lakiesityksen', 'arvioidaan', '##kin', 'vaikuttavan', 'siihen', 'eniten', 'valmista', '##jista', '.',
'_', 'Korjaus', '23', '.', '9', '.', '20', '##21', 'kello', '16', '.', '30', ':', 'Artikkeli', '##ssa', 'on', 'täsmen', '##netty', ',', 'et
tä', 'Applen', 'osalta', 'kyse', 'on', 'erila', '##isistä', 'liitti', '##mistä', ',', 'ei', 'lat', '##ure', '##ista', '.', '_', '*', '*',
'Lisää', 'aiheesta', ':', '*', '*', '[', 'Vanha', 'vaatimus', 'yleis', '##lat', '##urista', 'junn', '##aa', 'yhä', 'EU', ':', 'ssa', '–',
'langaton', 'lataus', '##teknologia', 'voi', 'tehdä', 'vuosien', 'pi', '##uha', '##keskustelu', '##sta', 'turhan', ']', 'haluaa', 'yhtenäis
tä', '##ä', 'kännykö', '##iden', 'lat', '##urit', '.', '[SEP]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[P
AD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PA
D]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]',
'[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]', '[PAD]']
```

FIGURE 18. Example of input text sequence converted to 512 tokens with BERT’s vocabulary

To know when a word is a suffix, WordPiece uses a “##” prefix to denote the division of a word into sub words. For example, the word "liittimistä" in figure 18 is split into "liitti” and “##mistä”, where the "##" symbol is used to differentiate tokens that typically begin words from those that occur within the middle or at the end. Like in this example word, it is beneficial to differentiate the individual question word “mistä” from the Finnish language suffix “-mistä” because they carry different meanings. Based on the scoring formula explained in chapter 3.6.1, “liittimistä”, even if it was

encountered multiple times in training materials, would not have been merged into a single token if the words “liitti” and the suffix “-mistä” were present in the training material more often.

Looking at figure 18, one can see that there are many long words that are included in the BERT vocabulary in their entirety, even if they are compound words or words with conjugation and declension. Such examples include “kehittämisestä”, “lakiesityksen”, and “jäsenmaiden”.
Examples of words that are unknown for the algorithm in use, a.k.a. Out of Vocabulary words (OOVs), and seemingly difficult to split include the following: “Europarlamentaarikot” (which is split into “Euro”, “##parlamen”, “##taar”, “##ikot”), “Elektroniikkavalmistajista” (“'Elektron”, “##iikka”, “##valmista”, “##jista”), and “kannettavien” (represented as ”kanne”, “##tta”, “##vien). The strength of subword tokenization is that without this partial matching capability, these entire words would be tokenized as [UNK], which stands for unknown, and none of their meanings would have been known to the model (Au Yeung 2020). Even the long, colloquial word “piuhakeskustelusta” in the article is matched to refer to the main word “keskustelu” when it’s corresponding tokens are “pi”, “uha”, “keskustelu” and “sta”.

```
Contents of input_ids:
tensor([  102,  1306,   166, 12397,  1826,  6379,  1727, 14513, 50006, 18749,
         8567, 10269,   405,  8140,  7825,  2224,   164,  2875,  1054,   472,
          111,   103, 48368, 50069,   166,  5824,   103, 49965,   119,  6828,
        46153,   142, 41646, 11295,   119,  1883, 19945,   119, 49342, 17919,
        32298,   119,  1883, 12397,   119, 24325, 10093,  9173,   364, 49840,
          119, 48294,   498, 21914,  7295, 10269,   111,   103, 22290,   339,
         2282, 36661,   142, 10413,  6203,  5203,   111, 26527,  6387,  3144,
        19600,   142,  1306,   166,  8416, 33278,   111, 49342, 17919, 11825,
        12257,  2050,   767, 18232,  5186,   119,   302, 43251,   704,  4563,
        14369,   111, 49342, 17919, 13615,  3036, 19800, 10563, 19477,   111,
         1306,   166, 12397,  1826,  3388, 43417, 15291,   704,  9376, 15249,
          246,   142,  1940, 46735,   471,  1191, 23662, 38246,  7295, 10269,
          111, 24606,  1517,  1726, 27615,   134,   111, 22290,   145, 11370,
        23662, 35617,   138, 21160,   734, 13623,   236,  1306,   236,  4017,
          145, 14856,   119,   206, 21093,  7295,  8053,   170,  5107,  5855,
         9871,  1329, 11621, 38953,   111, 19139,   418,  7295, 10269,  3347,
        21298,  8466,   346, 18568,  3030,   111,   724, 14856,   119,   206,
        23135,  7890,  3830, 43251,  8567, 24645,  6045,  5855,  9484,  1880,
         1604,   111,  9367, 43893, 10585,  3866,   339, 17060, 27374, 12397,
          105,  9987,  9191,  3225,   111, 39605,  6154,   164,  2875,  1054,
          472, 34778, 50016,  7295, 10269,   111, 24606,  4048, 43251,   204,
        26299,   166,   553,   166, 30675,   279,   111, 32442,   427,  7295,
        10269, 20019,   649,  4026, 25674, 49922,   111,  9321, 20511,  7295,
        27458,  1351, 16474,   943, 37969, 17457,   111,  9522,  1539, 45974,
        23206, 49837,   145,   684, 17749,  1345,   111, 26527,  6387,  3144,
        19600,   142,  1306,   166,  8416, 33278,   111, 49342, 17919, 11825,
        12257,  2050,   767, 18232,  5186,   119,   302, 43251,   704,  4563,
        14369,   111, 42393, 16747,  1971,   527, 41969, 19800,   172,   119,
          374,   145, 35732, 19477,   111,   721,   862,  5270,  6858,   119,
        19800,   142, 27474, 32504,   207, 28527,   364, 38607,   109, 10645,
          166, 26299,   166, 23206, 15596, 22990,   111, 19800,   145,   684,
          456,  4236, 41633,   109, 20550, 28758,  5436,   142,  8480, 46565,
         7194,   184, 25959,   891,  3592,  2700,  3036,   111,  3888, 38150,
         3293,   111,  1049,   111,   732, 22095,  3172,  1900,   111,  1490,
          236, 44050,   377,   145, 17845, 24568,   119,   206, 22308,  3476,
         1283,   145,  1591,  3705, 28758,  1263,   119,   193,  7295,  8053,
          170,   111,  3888,  2116,  2116,  7538,  5801,   236,  2116,  2116,
         3043, 10574, 19338,  2110,  8567, 30310, 19352,   149,  2431,  1306,
          236,  4017,   405, 45974, 23206, 49837,   326,  1013,  4561,   883,
         2347, 23173,   734,  8548,  3013,  1826, 34778, 50016, 45173,   407,
         7295, 10269,   111,   103,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0,     0,     0,     0,     0,     0,     0,     0,     0,
            0,     0])
```

```
Contents of token_type_ids:
tensor([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])

Contents of attention_mask:
tensor([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
        1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
        0, 0, 0, 0, 0, 0, 0, 0])
```

FIGURE 19. Representation of example sequence with tokens translated to ID’s, token types and the attention mask

The algorithm works with the numeric IDs that represent the tokens in its corpus. The token IDs of the example sequence are displayed in figure 19. All tokens have a unique identifier in the model's vocabulary; for example, the [SEP] token that separates the different texts in the sequence is displayed as 103, and the [PAD] token corresponds to ID number 0. These special tokens all have fixed IDs in all BERT model variations, regardless of their pre-training. The word "kännykkä", on the other hand, which might not be known to all BERTs as is, is known to TurkuNLP's FinBERT model's corpus as the identifier number 18749.

Because there are padding tokens in this example sequence, the model needs to know which tokens to discard from analysis. This is done with an attention mask. The attention mask value is set to 1 per token that represents the original sequence and to 0 for the added [PAD] tokens that the model can ignore. Token type IDs, which too are displayed as a binary mask in figure 19, are, in our case, all set to 0. As explained in the BERT tokenizer documentation (Hugging Face), token-type IDs are used in cases where there are two distinct types of sequences in the input, for example, pairs of questions and answers. The separation of the two input types is represented as binary, where 0 might be set for the tokens forming the question and 1 for the answer. Because this sort of binary representation is not required for the text sequences in this study, all the token type IDs are left set to 0.

Once the texts are formatted in this way, they are ready to be fed into the model for fine-tuning. Correct formatting is crucial for ensuring that the language model comprehends and processes the data accurately (Au Yeung 2020).

Finally, when all different input types are formatted to fit the model, they are brought back together into combined sets of data. The PyTorch framework offers tools for storing data with its torch.utils.data library. A TensorDataset object wraps tensors together in a way that makes it possible to retrieve individual samples by indexing tensors along the first dimension, according to the torch.utils.data documentation (PyTorch). PyTorch datasets are formed in a way that simplifies the process of iterating through the data in training and evaluation (Wherlock 2021).

The tensors in the final TensorDatasets, similar for both training and validation, are as follows:

- Input text token IDs: These tensors represent tokenized input text sequences
- Categorical features: These tensors represent encoded categorical features
- Numerical features: These tensors represent normalised numerical features

- Attention mask: These tensors are attention masks that specify which tokens in the input sequences should be attended to (1 for actual tokens and 0 for padding tokens)
- Labels: These tensors represent the prediction targets, i.e. user-need classes

4.3 Implementation

This chapter introduces the fine-tuning of the model with a feedforward neural network. First, the model for this case is introduced, followed by the training process. Lastly, the outputs of the model with the training dataset are validated against the validation dataset.

4.3.1 The Neural Network

The training is done on a classification model, which combines text, categorical, and numerical features. The text features are first processed with BERT, and all three different types of features are then concatenated into a single vector, which is fed into a MLP for final classification.
This model uses custom classes that are built from the examples in the dqi_predictor project (GitHub 2021). The model requires a Transformers BertConfig object from the Hugging Face Transformers library, and the config object needs to have three additional properties manually added to it: “text_feat_dim” being the length of the BERT vector, “cat_feat_dim” the number of categorical features, and “numerical_feat_dim” the number of numerical features.

```
config = BertConfig.from_pretrained(
    'TurkuNLP/bert-base-finnish-cased-v1',
    num_labels=len(label_dict)
  )
```

In the excerpt, calling BertConfig with the parameters "TurkuNLP/bert-base-finnish-cased-v1" and "number of possible different outputs" (retrieved by getting the length of the label dictionary structure) resulted in the creation of a configuration object.

Next, feature dimension lengths are added to the object's corresponding variable described above. The text dimension length is set to the "hidden_size" parameter, which comes from BertConfig. This hidden size is 768 embeddings for each token in BERT-base models. With this object, setting up the model can be done.

```
model = BertConcatFeatures.from_pretrained(
    "TurkuNLP/bert-base-finnish-cased-v1",
    config = config
)
```

The BertConfic object is used in the excerpt to set up an instance of BertConcatFeartures, which is a custom extension of the Hugging Face BertForSequenceClassification class based on the concatmodel.py published in the dqi_predictor Github project (2021).

To prepare the MLP for feature combination, the length of the final input vector needs to be calculated by summing together the lengths of the three different content types.
For numeric features, a batch normalizer is created using PyTorch's neural network library and its function BatchNorm1. Even though in pre-processing, the numerical values were once normalised with quantile transformer to make the overall distribution of data more Gaussian, the statistics of each mini batch during training can still vary. Batch normalising addresses these per-batch fluctuations, potentially making training smoother and more stable (Ioffe & Szegedy 2015). While quantile transformer is a fixed, one-time normalisation done in pre-processing, batch normalisation's parameters are learnable and will adjust during training.

The combining MLP module used in this study follows the implementation introduced in the mlp.py script in the dqi_predictor Github project (2021). The MLP class is a custom implementation of a MLP that references the Multimodal Toolkit's (Gu & Budhkar 2021) functions. To set up the combination module MLP, the number of layers and the number of neurons in each layer need to be specified. The dqi_predictor project comments refer to the Multimodal Transformers package, which Gu and Budhkar (2021) introduced, as having a formula for choosing these dimensions. The formula is that each layer of the MLP has $\frac{1}{4}$th the number of neurons as the previous one. So, the combined feature length is iterated over, divided by 4, until it drops below the number of outputs the MLP needs to have.
MLP layer sizes as calculated with this formula are as follows: Input: 790, being the full length of all combined features; Hidden: [197, 49, 12], being the full length of all combined being divided by 4 as many times as possible; and Output: 3, target user needs label count. With this information, an MLP object is constructed using the determined architecture.

The excerpt below initiates the MLP object with a combined vector length of 790 for 3 output labels

and 3 hidden layers, as calculated above, with a dropout rate of 10%, specifications for the three hidden channels' neurons [197, 49, 12], and batch normalisation set to true.

```
self.mlp = MLP(combined_feat_dim,
               self.num_labels,
               num_hidden_lyr=len(dims),
               dropout_prob=0.1,
               hidden_channels=dims,
               bn=True)
```

In this custom BertConcatFeatures implementation, the forward-passing function takes in the same inputs as the feedforward logic of BertForSequenceClassification. However, there are two extra parameters that hold the tensors for the categorical and numerical features.
All text content is run through BERT. Invoking "self.bert" returns outputs from the encoding layers and not from the final classifier.

Overfitting is caused when the network learns spurious patterns in training data. To recognise these patterns, the network relies on a specific combination of weights, known as a "conspiracy" of weights. However, removing one weight can damage the conspiracy. This is the concept behind dropout. To avoid conspiracies, a fraction of a layer's input units is randomly dropped at each training step, forcing the network to search for general patterns with more robust weight patterns. (Charles 2021.)

In practice, the choice of dropout rate, along with other hyperparameters, requires fine-tuning and validation against the dataset and task at hand to determine the most effective configuration. The dropout rate in this implementation is set to 0.1, which means dropping 10% of samples. This was also the rate used in the concatmodel.py published in the dqi_predictor Github project (2021). Varying the rate slightly was tested, but 0.1 was found to yield the best outcomes. The study "How to Fine-Tune BERT for Text Classification?" (Sun et al. 2020) also kept the dropout probability constant at 0.1 during fine-tuning, which might suggest that it is a commonly used setting for this sort of scenario.

Next, the other features are concatenated together with the BERT's outputs, but first, the previously introduced normalisation is performed on the numerical features. After that, all the feature tensors are concatenated into one with the PyTorch torch.cat function as follows:

```
combined_feats = torch.cat((cls, cat_feats, numerical_feats),dim=1)
```

where “cls” refers to the outputs of BERT for the text sequence. The “dim=1” argument means that the tensors are being concatenated side-by-side (horizontally). In other words, if each tensor is imagined as a table, this means adding more columns to that table. These combined samples are next run through the MLP for outputs by calling:

```
logits = self.mlp(combined_feats)
```

The output logits contain the predicted probabilities of samples belonging to any of the three possible classes. In a multi-class classification case, the true values are that one class has a probability of 1.0, and every other label has a probability of 0. When making predictions, however, the model returns values somewhere in between these two for all three output labels.
For the model to learn if there are improvements to be made, the differences between true values and predicted likelihoods need to be calculated. This is done through loss calculations.

The training loss is calculated with cross-entropy loss, which is commonly used in classification tasks. With cross-entropy loss, each predicted class probability is compared to the desired output of 0 or 1. The calculated loss penalises the probability based on how far it is from the expected value. The penalty is logarithmic, yielding a large score for significant differences close to 1 and a small score for minor differences close to 0. (Shah 2023.)

4.3.2 Model Training

The training and validation process is started by creating PyTorch DataLoaders for training and validation datasets. According to Piepenbreier (2022), DataLoaders are a crucial part of PyTorch that simplifies the complexity of handling data for deep learning models. Data loaders enable multiple operations on the data, for example, batching the defined dataset and shuffling and sampling its data points consistently across all tensors.

```
batch_size = 16

dataloader_train = DataLoader(dataset_train,
                              sampler=RandomSampler(dataset_train),
```

```
                                batch_size=batch_size,drop_last=True)

    dataloader_validation = DataLoader(dataset_val,
                                       sampler=SequentialSampler(dataset_val),
                                       batch_size=batch_size,drop_last=True)
```

First, one must set the batch size, which means the number of data samples the model is to process at once. In the field of training neural networks, there has been an ongoing debate regarding the preferred batch size and whether it should be large or small. A comprehensive study titled "Revisiting Small Batch Training for Deep Neural Networks" (Masters & Luschi 2018) delves into this matter. The study emphasises that while using larger batches enhances efficiency and parallelism, it can limit the range of learning rates suitable for stable training and potentially harm test performance. On the other hand, according to Masters & Luschi (2018), smaller batch sizes have demonstrated better generalisation performance and optimisation convergence because smaller batches allow for more up-to-date gradient calculations, resulting in stable and reliable training. However, opting for smaller batches comes with a trade-off, as while it requires less memory, training typically takes longer because small batch sizes mean more iterations are needed to complete an epoch, as each iteration processes fewer data points.
Ayyadevara (2019) concludes that smaller batch sizes result in better accuracy for the same number of epochs, but points out that while deciding the number of data points to be considered for a batch size, one should ensure that the batch size is not too small so that it might overfit on top of a small batch of data.

For this instance, the batch size is set to 16 samples at once. Batch sizes such as 16, 32, and 64 (numbers that are powers of 2) are frequently used because they enable efficient memory utilisation (Upadhyay 2020). A batch size of 16 is comparatively small, and it is a middle ground between true stochastic gradient descent (where the batch size is 1) and larger batch sizes. As stated previously, smaller batches, while often leading to more robust training, do extend the training duration. As the GPU is used for training, there might have been enough memory to increase the batch size to accelerate the training process. However, with this small dataset of under 500 rows, the time for training wasn't all that long, especially when it was run on the MPS backend. If one had access to even more computational resources and wanted to experiment, larger batch sizes (like 32 or 64) could speed up training.

Sampler functions from the PyTorch library are set as parameters when initiating the data loaders. RandomSampler is used for the training data. It shuffles the data randomly, which is often desired during training to ensure the model does not memorise the sequence of training data. As stated in the documentation (PyTorch), shuffling the samples is relevant to mitigating the risk of overfitting occurring during training. SequentialSampler, on the other hand, retrieves data in the order it's stored in the dataset. This is used for validation, where the sequence of data doesn't need to be randomised.

The last parameter for the data loader is "drop_last=true". It refers to dropping the last batch if it's smaller than the specified batch size. It is often set to “true” to ensure consistent batch sizes; that might be required, especially if a batch normalisation layer is used. When the batch size is 16, there will be 29 batches for the selected training dataset with 478 rows, but the 29th batch will contain 14 samples. Dropping out the last one means that only 464 rows out of 478 rows of data are used in each epoch of the training. With a limited amount of data, ensuring that the model has exposure to the entire dataset during training might be worth considering, so it could learn as much as possible from the available examples. However, the model seemed to perform better with uniform batch sizes, even though the difference in the last batch size wasn’t remarkable. Therefore, dropping the last batch was set to true.

The next step is to set up the optimizer. It is done using the AdamW optimizer from PyTorch:

```
optimizer = AdamW(model.parameters(),
                  lr=1e-4)
```

The optimal learning rate depends on the specific task and the amount and nature of the training data. When fine-tuning a pre-trained model, it's common to use a smaller learning rate to make sure that the already-learned features are not distorted too quickly (Buhl 2023).
In the dqi_predictor classifier_combined_sophisticated.py (GitHub 2021) script, the learning rate is set to 5e-3 with a comment that the solution was based on the Multimodal Toolkit’s (Gu & Budhkar 2021) documentation. Using the suggested learning rate, however, resulted in poor model performance with this study’s data and implementation. Chugh (2022), on the other hand, summarises that a varying learning rate between 0.0001 and 0.01 is considered optimal in most of the cases. Therefore, the learning rate of 1e-4 was tried out and kept, because it yielded the best results.

```
scheduler = get_linear_schedule_with_warmup(optimizer,
                                            num_warmup_steps=0,
                                            num_training_steps=len(dataloader_train)*epochs)
```

The Hugging Face Transformers library includes a learning rate scheduler, which helps improve convergence and model performance by modifying the learning rate during training.
According to the documentation (Hugging Face), "get_linear_schedule_with_warmup" is used to linearly decrease the set learning rate from 0.0001 to 0 as training progresses. “num_warmup_steps=0” as a parameter means there is no warm up period, which would first linearly increase the learning rate. The total number of training steps is calculated by multiplying the number of batches by the number of epochs. This tells the scheduler when to reduce the learning rate to 0.

Once the data has been prepared, loaded, and the optimizer set up, the training is ready to begin. The model is trained over a specified number of epochs in the main training loop.
Within the training loop, there is another loop that iterates through batches of data from the DataLoader.

```
for batch in dataloader_train:

    model.zero_grad()

    b_input_ids = batch[0].to(device)
    b_categ_feats = batch[1].to(device)
    b_numer_feats = batch[2].to(device)
    b_input_mask = batch[3].to(device)
    b_labels = batch[4].to(device)

    inputs = {'input_ids':      b_input_ids,
              'cat_feats':      b_categ_feats,
              'numerical_feats':b_numer_feats,
              'attention_mask': b_input_mask,
              'labels':         b_labels
             }
```

```
        outputs = model(**inputs)

        loss = outputs['loss']
        loss_train_total += loss.item()
        loss.backward()

        torch.nn.utils.clip_grad_norm_(model.parameters(), 1.0)

        optimizer.step()
        scheduler.step()
```

When a set of inputs is passed to the model, it processes them through its layers and returns an output. This is referred to as a forward pass in the context of deep learning models. Sophisticated models, like those found in the Transformer library, return a dictionary as their output instead of a single result. This dictionary includes the model's predictions (logits for each category) and the losses that occurred during training (Hugging Face).

After the loss is identified, the subsequent step involves assessing the impact of each parameter, including the model's weights and biases. This requires determining which part of the model is primarily responsible for the error and how to modify it in order to minimise that error. Backpropagation comes into play here, as it helps calculate the extent to which each parameter should be adjusted to reduce the loss. This adjustment is achieved by utilizing the loss.backward() function, which computes gradients for each parameter. (Glassner 2021, chapter 14.)

A technique called gradient clipping is used to prevent excessively large updates to the model's weights, ensuring stable and consistent learning. This is done by torch.nn.utils.clip_grad_norm_, which is a PyTorch function that enables clipping by calculating the norm (magnitude) of gradients and reduces their scale if the norm surpasses a threshold.
By controlling the size of gradients, extremely large updates that could destabilise the model's learning are avoided. (Hany & Walters 2019.)

The batch loop in the excerpt above concludes when an update step is performed by the optimizer, and the learning rate is adjusted by the scheduler.
Once all the batches have been trained for one epoch, the average training loss is calculated for the whole epoch by dividing the accumulated training total with the number of batches.

```
loss_train_avg = loss_train_total/len(dataloader_train)
```

4.3.3 Model Validation

The outputs of an epoch are next evaluated against the validation dataset.

```
val_loss, predictions, true_vals, all_probabilities = evaluate(dataloader_validation)
```

The validation dataset is run through a loop of its own, where for every batch of data in the validation set, the inputs are extracted from the model the same as in training.
The model returns the loss and logits (the scores that the model predicts for each class) in exchange for the inputs. These logits are accumulated to calculate the overall validation loss, and logits and true labels are stored in lists for use outside of this loop. This is repeated for each batch to validate the model's performance.

```
loss = outputs['loss']
logits = outputs['logits']
probabilities = outputs['probabilities']
loss_val_total += loss.item()

logits = logits.detach().cpu().numpy()
probabilities = probabilities.detach().cpu().numpy()
label_ids = inputs['labels'].numpy()
 predictions.append(logits)
 true_vals.append(label_ids)
all_probabilities.append(probabilities)
```

After looping through the batches, the average validation is calculated by dividing the accumulated validation loss by the number of batches. Additionally, the lists of logits (predictions) are combined with the true values into NumPy arrays.

```
    loss_val_avg = loss_val_total/len(dataloader_val)

    predictions = np.concatenate(predictions, axis=0)
    true_vals = np.concatenate(true_vals, axis=0)
    all_probabilities = np.concatenate(all_probabilities, axis=0)

     return loss_val_avg, predictions, true_vals, all_probabilities
```

These returned values are then used to calculate a weighted F1 score for the epoch.
The calculation of the F1 score is done with a small function that uses NumPy and Scikit-Learn's functions.

```
def f1_score_func(all_probabilities, true_vals):
    preds_flat = np.argmax(all_probabilities, axis=1).flatten()
    labels_flat = true_vals.flatten()
    return f1_score(labels_flat, preds_flat, average='weighted')
```

With NumPy's argmax function, the class with the highest probability is retrieved. For example, for out of three user need classes, the predicted output for one article could be [0.25, 0.40, 0.35], meaning that the article, in the models' opinion, is in 40% likelihood of type 2: understanding. With index = np.argmax([0.25, 0.40, 0.35]), the index of the most likely class is returned, in this case index 1.
All argmax indexes are stored in an array per sample, and this array is compared against the actual labels with the Scikit-Learns' f1_score function. The average='weighted' parameter instructs the function to compute metrics for each label and calculate their average weighted by the number of true instances for each label.

This process of training and evaluating is repeated for the number of epochs that were determined before training. At the end of each epoch, after having seen all the training examples, the model will have updated its weights and biases multiple times based on the error it computed for each training example.
The goal is to look for the epoch with the best accuracy and the least validation loss, while also keeping an eye out for overfitting the model (Baeldung 2023).

5 RESULTS AND ANALYSIS

In this chapter, the results of the model trainings are reviewed and analysed.

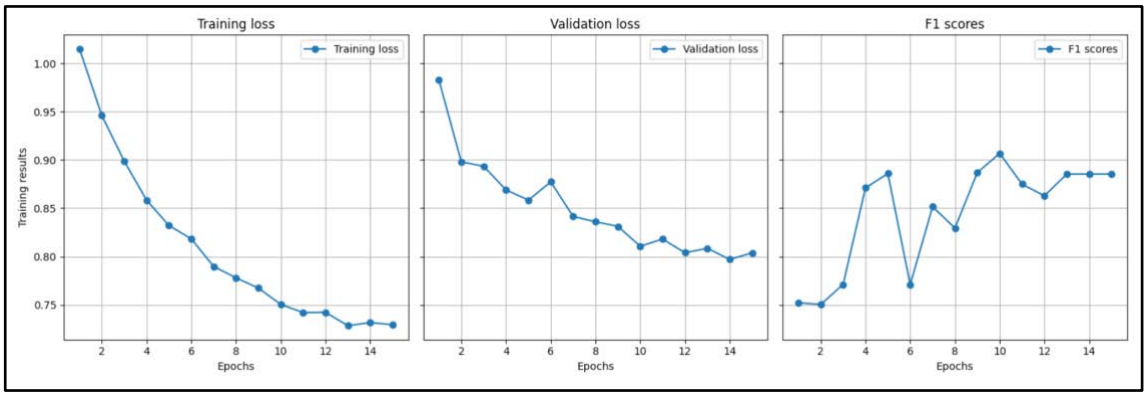

FIGURE 20. Training metrics over 15 epochs

The results from the model training and evaluation are looked at against the loss and accuracy during training epochs. Figure 20 is an example of training over 15 epochs, where the training loss decreases evenly during training. The F1 score for accuracy is the one measure that behaves in the most unexpected ways during early training. It is worth noting that all the scales vary from 1.5 down to 0.7.

The validation loss is not so linear, but overall decreases over the epochs as well. The fact that it rises during training may not necessarily be a sign of definite overfitting (Brownlee 2019). It is important to consider that the data can contain noise. Optimising a neural network is a complex process, and sometimes the model might temporarily fit to noise in the data, resulting in a brief increase in validation loss, but then continue to find a more generalizable solution in subsequent epochs (Brownlee 2019).

The loss landscape can also fluctuate based on the optimizer and learning rate. For instance, with a higher learning rate, there's a chance that the model might overshoot the solution temporarily, leading to an increase in validation loss. When the learning rate decreases over time and the optimizer adjusts, the model might converge back to a better solution in later epochs. (Brownlee 2019.)

Techniques used in this model, like dropout and batch normalisation, can introduce some variability into the training process. Consequently, these techniques may occasionally cause fluctuations in validation loss (Charles 2021).

Although a temporary rise in validation loss followed by a decrease in later epochs doesn't provide clear evidence of overfitting, it could indicate the early signs of it. If the validation loss continues to rise in subsequent epochs or oscillates frequently, it could be a sign that the model is beginning to overfit. (Brownlee 2019.)

The accuracy in the above example stays in place after the 13th epoch, which suggests that for training, there are enough epochs. The highest accuracy was introduced in the 10th epoch. The 14th epoch here has the lowest validation loss at the highest F1 accuracy. Taking a closer look at the metrics of the model at the state of the 10th, where the was the highest accuracy, and the 14th epoch with the highest accuracy together with the lowest validation loss, are evaluated against the following metrics:

A simple average of the above two metrics is a way to summarise the model's performance in a way considers both accuracy and F1 score.

TABLE 6. Performance metrics compared between two states of training

Metric	Epoch 10	Epoch 14
Validation loss	0.81072	0.79705
Accuracy	0.90625	0.88542
F1 Score weighted	0.90681	0.88542
Avg. Acc./F1 Score	0.90653	0.88542

All these three metrics point in the direction that the model did well in predicting both outcomes for roughly 89% of the values in the test dataset. This can be looked at against the absolute and percentage accuracy per class:

TABLE 7. Accuracy per class compared between two states of training

	Accuracy	
Class	Epoch 11	Epoch 14
information	47/50 (94%)	47/50 (94%)
understanding	22/25 (87%)	20/25 (80%)
other: emotional, social/community, identity/self-reflection, learning	18/21 (86%)	18/21 (86%)

When encountered with a situation where the validation loss continues to decrease but there is a model from a previous epoch that has higher accuracy, it might be beneficial to prioritise the model with the lower validation loss. This is because the model with lower validation loss tends to be more robust and capable of generalising to unfamiliar data (Brownlee 2019). However, the decision also depends on the application and requirements. From the example above, one might want to favour Epoch 10 for practical considerations, where all the classes are predicted correctly at an over-85% succession rate. This means that the model might be less versatile for unseen data but more accurate for this specific dataset. Keeping in mind the research question "Can articles be classified by user needs", the higher accuracy seems more compelling. But the real-life application of this research question will not work if it is not addressed with a generalised model.

Based on these values, it is possible to think that the model did okay, but it is worth pointing out that the number of test cases is low, especially for the smaller classes. This alone might be the cause of varying accuracy results with different evaluation datasets.
A high score on a minimal evaluation set might still mean it is not generalizable to a wider dataset, because the model was tested against only 20 or so samples in a few iterations.

Changing the train-test split to include a higher portion of the dataset in the testing set leads to poorer scores in the evaluation. This might be because the already limited number of samples in the training set is diminished even more, that higher performance metrics previously were a fluke, or that there's a chance that the model is overfitted with fewer test cases.

Factors that were noted to influence the accuracy of the predictions are, for example, but are not limited to: size of the training dataset, learning rate, dropout rate, batch size, and amount of text sequence cleaning.
However, running the training with the same operations on data with the same parameters still results in some differences in outputs. This might be because the samples are different due new random sampling by train-test split and dropping uneven batches, etc.

In addition to the division of the train-test split, what also affects the amount of data to train on and validate against are the dropouts in each training batch. The dropout rate is in place for a reason: to prevent overfitting (Charles 2021), which in the case of a small dataset is a concern to begin with. However, allowing a larger portion of the already limited training data to be dropped was found to significantly decrease the performance of the model.

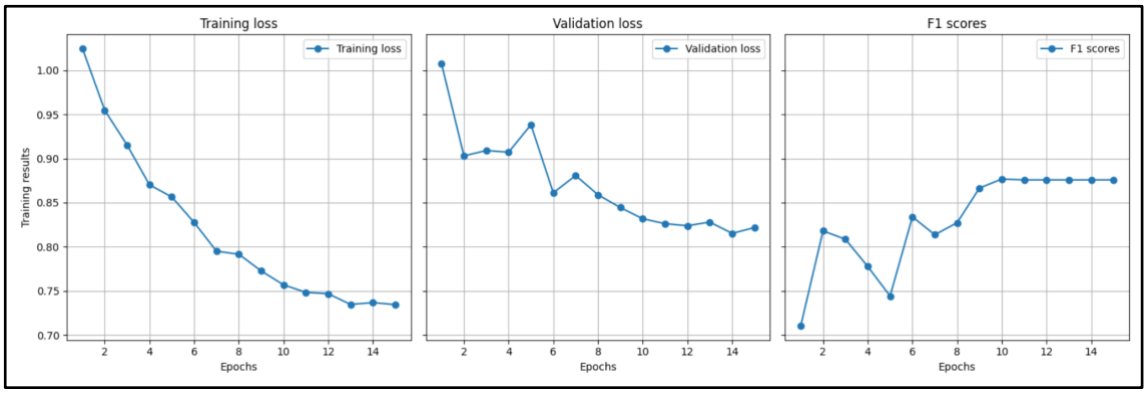

FIGURE 21. Metrics over another 15 iterations of training with the same specifications

As portrayed in figure 21, which visualizes another training run of 15 epochs on the same specifications as before, the validation loss remains at a higher level and the accuracy does not increase as much as earlier.

A closer look at epoch 12, which is the point before the validation loss takes a shift upward, and epoch 14, with the lowest validation loss, looks like the following:

TABLE 8. Performance metrics compared between two states of training

Metric	Epoch 12	Epoch 14
Validation loss	0.82381	0.81518
Accuracy	0.87500	0.87500
F1 Score weighted	0.87578	0.87578
Avg. Acc./F1 Score	0.87539	0.87539

TABLE 9. Accuracy per class compared between two states of training

	Accuracy	
Class	Epoch 12	Epoch 14
information	43/50 (86%)	43/50 (86%)
understanding	21/25 (84%)	21/25 (84%)
other: emotional, social/community, identity/self-reflection, learning	20/21 (95%)	2021 (95%)

The contents of the tables above show that the results don't change after epoch 12, and while the overall F1-score for the whole model was less than in the previous model, it seems that the predictions per class are not doing much worse than in the earlier example.
With these results, one could conclude that 12 epochs are enough for training.

Looking at the final likelihood percentage predictions, one could analyse how easy or difficult it is for the model to make distinctions between the different classes.

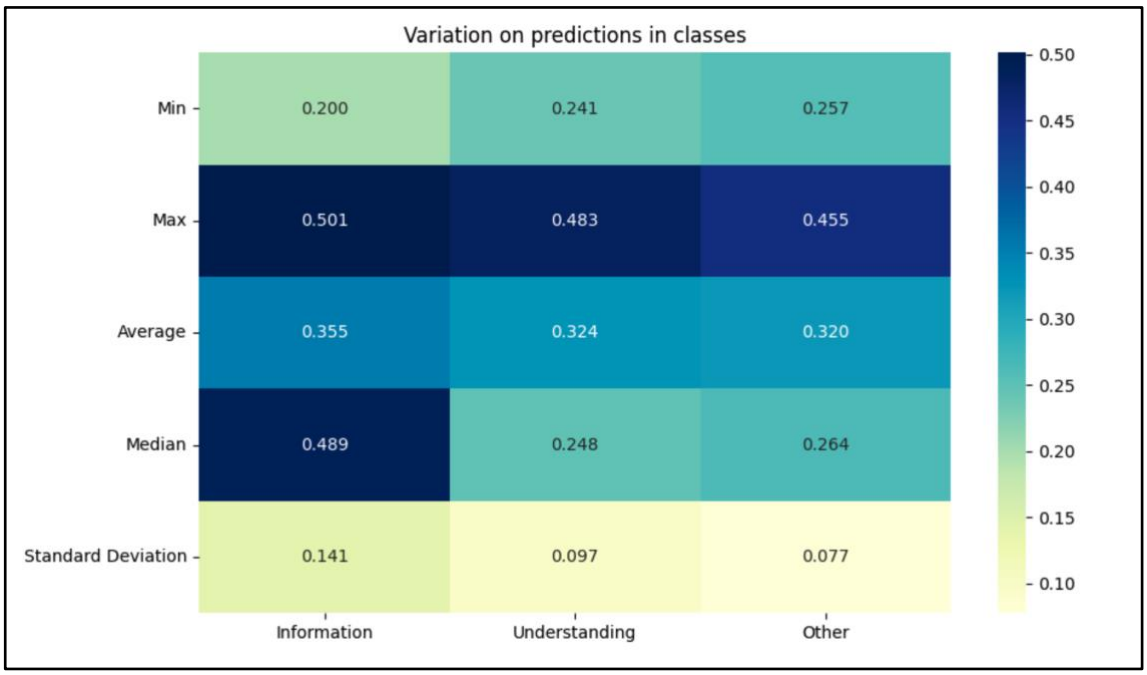

FIGURE 22. The probabilities of class and statistics during training

By plotting out the variation in probabilities in figure 22, one can see that the scale of values present is minimal. The average value for all samples to belong in any of the classes seems to roughly be around 33%, which could well enough be a coin toss between the three target classes. But the maximum and minimum predictions, and thus the median, vary a little further from the average. It could then be concluded that there were samples present, of which the model could be sort of confident about predicting the class of, at least to differ from the 33% chance of it being any. Still, the smallest likelihood is 20%, and the largest is 50% at best, meaning that the model is not overly confident about any predictions. The largest class, “Information”, is the most varied, but given that there are more samples in the class to predict for, the variation is likely to grow. The small variation in the other two classes could be due to the dataset size; each of the classes has roughly half the samples as the first class.

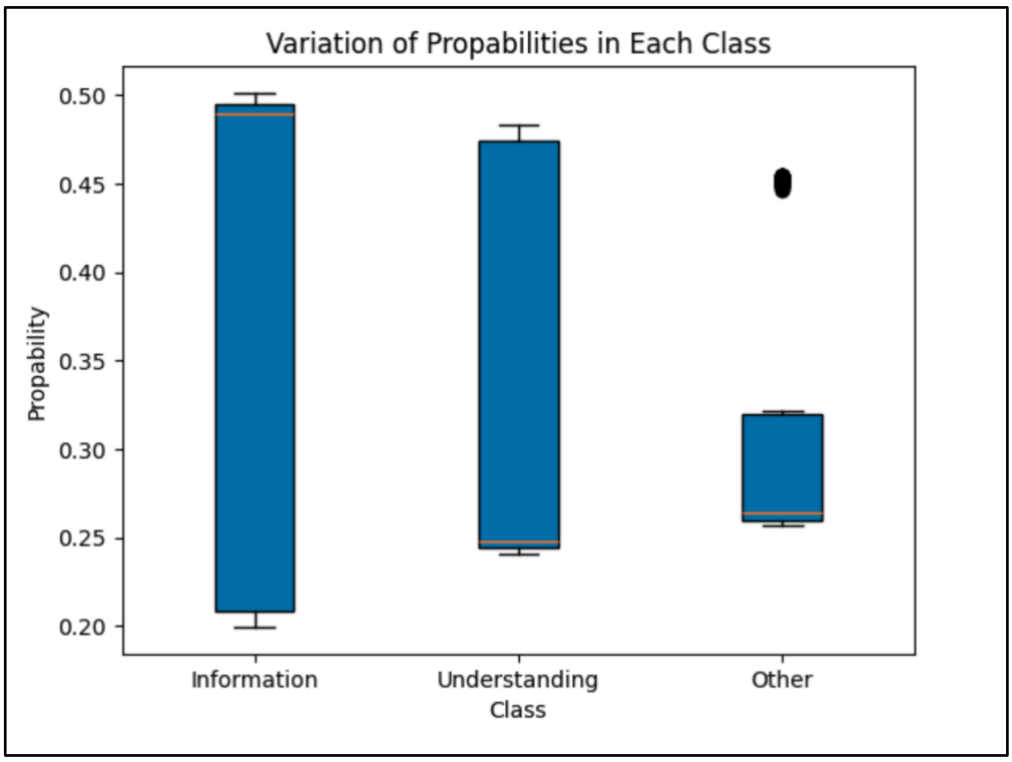

FIGURE 23. The variation of class probabilities for all samples during training as a boxplot

As the boxplot in figure 23 represents, the two smallest classes do not behave alike. The third class, “other”, is the one whose predictions behave most unexpectedly, where most of the predictions are within the range of 25–30%, but there are some fliers with a 45% predicted likelihood. This can be said to be in correlation with the labelling of the samples for the training dataset, where the two first classes were the easiest to distinguish and the last one the most obscure and up to speculation. Therefore, these differences and outliers are not necessarily to be interpreted as faults.

Overall, it seems that the biggest limitation here is the size of the dataset. To test what happens with an even slightly larger dataset, the training was run on a set with 40 more samples, but all with the same settings. Adding 40 samples when 500 more might be needed is not a lot, but it might give an idea of the impact of the dataset size. It’s a 7% increase from the original dataset size.

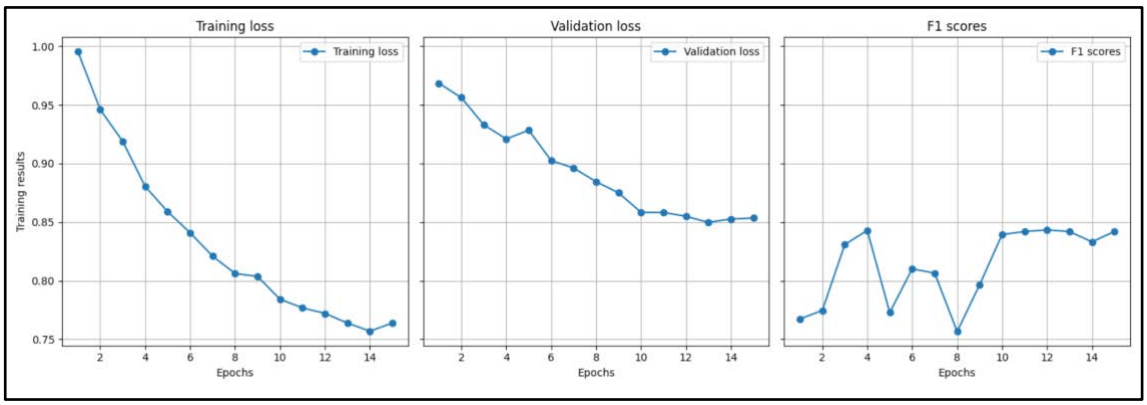

FIGURE 24. Same training procedure on a slightly larger dataset

Figure 24 shows that with a new dataset and same settings, the accuracy is worse, and the loss is larger. Running the numbers on the 13th epoch, which is the best in this case based on the metrics, shows the ability to predict the 2nd class of "understanding" only 61% of the time, in this dataset being 17 samples of 28.

TABLE 10. Metrics of larger dataset at 13 iterations

Metric	Value
Validation loss	0.849775
Accuracy	0.84821
F1 Score weighted	0.84193
Avg. Acc./F1 Score	0.84507
Accuracy for Class: information	53/56 (95%)
Accuracy for Class: understanding	17/28 (61%)
Accuracy for Class: other: emotional, social/community, identity/self-reflection, learning	25/28 (90%)

The largest class ended up getting better results, correctly predicting 95% of the time.

This would suggest that with these parameters in place, the trained model is likely to not be well generalizable to unseen data, at least not in all three classes. It is notable that the large likelihood of predicting the first class correctly might already in itself be usable information for an example application. For a more generalizable model for all three classes, training with a twice-the-size dataset to begin with might be required.

When assessing the outputs of the model based on accuracy, it is worth considering that deciding between classes was not straightforward for humans either in all cases. Examining some of those samples that were not classified correctly was thought to be worthwhile. Based on just 9 misclassified samples it is fair to say that there were not many clear-cut cases present, and even then, it was possible to deduce some reasons why the model might have predicted as it did.
Examples of clearer errors included brief articles about bike thefts and a then-recent volcano eruption in Tonga. Both were labelled information by humans and understanding by machines. The first included tips from the police on how to best safeguard bicycles and was written with some playful language. The latter contained a video from the eruption, and in addition to stating the facts, included some speculations from experts about what might be expected. Regardless, neither of these are hard for humans to place in other than the first category.
Some of the example cases did, however, seem to be those that were difficult to place in the first place. These include an introduction to a new military museum, that contains background on the Finnish military service's history and some historical audio material. This was deemed as information by the experts because the underlying motive was to inform the reader about a new museum. The historical contents of the article could still be considered to offer understanding and reflections to one's own possible history in the military service. Another example was a column about harassment, which the model classified as understanding, but humans as other. It is understandable to place a column as offering understanding when it contains points of view. It was placed in the third category because it is a personal point of view, rather than general information, and these articles are then often sources for self-reflection and identification. A third example is a news story about a complex pyramid scheme, originally classified as information and predicted as understanding. This article was not long, but still contained some comparisons to previous schemes, a brief caption about cryptocurrencies and links to investigative video content that was released on the subject a couple of years before. All these considering it might be fit for the predicted class as well.

6 CONCLUSIONS AND FUTURE APPLICATIONS

Since the dataset for this study was created in 2022, some of the metadata used in the dataset has changed due to changes in publishing systems and analytics collection. Therefore, the model, trained on this dataset as is, would not likely be suitable for classifying articles published at this time. That, however, is not the biggest concern about the models' applicability, since the generalisability of the model does not seem to be great for unseen data. This is most likely largely because the dataset to begin with was modest in size.

Were this study to be done again, more time would have been spent labelling more training data instead of trying to fit the model to the existing dataset as well as possible.

Regarding future applications, the research question that started this process is about to become redundant since the Yle News and Current Affairs unit has decided that journalists will begin labelling user needs to articles as they publish them. This labelling will be done during the publishing process within the publishing systems, and the data will be collected as metadata and stored in Yle's data warehouse, accessible for analysis. Therefore, a separate classification algorithm will not be needed for labelling, at least not the kind that was built for this research. It is worth keeping in mind, however, that once more labelled metadata about articles' user needs will be generated in the future, larger and updated volumes of possible training data for future use cases will be available. This might be taken to use in a case where it would be of interest to classify older articles, the ones created before the label-as-published process begins, to gain a better overview of the user need divisions development over time.

A future development possibility might be to assist the journalists in the manual labelling with the algorithm. This might mean suggesting the most likely predicted user need category as a default, which could then be changed if it was seen unfit. This approach might also highlight possible contradictions in labelling. For example, if one was planning to write a piece for deeper understanding, but the machine labels the outcome to belong in the up-to-date information category, one might still rethink if there were any choices to be made to distinguish the piece more clearly from the information category.

Examining the possible differences in the labelling outcomes between humans and machines could be interesting for the development of the user need labelling process. This could bring to light situations where teams would like to produce less articles in the information category, so much so

that they would be more likely to label the articles to other classes more leniently than a machine might.

Documenting clearer guidelines and developing a more systematic approach for labelling the articles by hand might be required in the future. Even though the team that performed the initial labelling thought they had figured out what meeting each of the user needs meant in article form in practise, the reality wasn't always clear. Some error, therefore, might need to be accepted then for machines as well, because there are always grey areas when dealing with such abstract matters as human needs.

All this being said, the challenge of putting together a custom implementation of a modular deep neural network was an interesting and teachable opportunity. The structure of the model might still be useful in some other future applications that would benefit from a similar logic.
For possible future implementations, planning and testing systematically what non-textual features to bring into the model might be a good way to look for improvements. Even though DNNs extract the relevant features from the data, the possible features that were not included in the original dataset to begin with by oversight in model planning cannot contribute to the model performance at all. A more structured plan on what data is available at the given time and a proper evaluation of the data might be a good place to start to look for better results.

It is worth noting that since the field is so rapidly evolving, employing a different architecture or a more recently released LLM with different learning capabilities might be the next step towards better performance. There could be more suitable models and training options available for cases with limited datasets, like this one.

7 SUMMARY

To answer the research question “Is it possible to use a machine learning algorithm to identify the different user needs in the context of Yle News?”, the black-and-white answer seems to be yes, it is possible. But what remains in the grey area is to what extent the classification is reliable for unseen data. Putting together a multi-layer deep neural network that combines both textual and non-textual features for classification is possible, but to train a generalizable model with it would have most likely required a larger dataset. Despite of the size, however, the machine can learn only from what is presented to it. When the original labelling of user needs was not always easy to the people performing it, neither can the model be expected to always distinguish between the different classes easily. Thus, the outcome of this study might not be a functioning model to meet real-life requirements. Instead, what can be derived from this research is an understanding of the intricacies of neural networks, the importance of planning ahead and training with enough data.

REFERENCES

Alaloul, Wesam Salah & Qureshi, Abdul Hannan 2020. Data Processing Using Artificial Neural Networks. IntechOpen. Search date 31.10.2023. https://doi.org/10.5772/intechopen.91935.

Alam, Mahbub 2020. Data normalization in machine learning. Search date 30.11.2023. https://towardsdatascience.com/data-normalization-in-machine-learning-395fdec69d02.

Apple Inc. 2023. Accelerated PyTorch training on Mac - Metal. Search date 30.11.2023. https://developer.apple.com/metal/pytorch/.

Au Yeung, Albert 2020. BERT - Tokenization and Encoding. Search date 28.11.2023. https://albertauyeung.github.io/2020/06/19/bert-tokenization.html/.

Ayyadevara, V. Kishore 2019. Getting ready. Packt Publishing Ltd. Search date 2.12.2023. https://learning.oreilly.com/library/view/neural-networks-with/9781789346640/42198ee2-8da7-4901-89d7-545a17329590.xhtml.

Baeldung 2023. Training and Validation Loss in Deep Learning. Search date 3.12.2023. https://www.baeldung.com/cs/training-validation-loss-deep-learning.

Baeldung 2021. Stratified Sampling in Machine Learning. Search date 30.11.2023. https://www.baeldung.com/cs/ml-stratified-sampling.

Brown, Tom B., Mann, Benjamin, Ryder, Nick, Subbiah, Melanie, Kaplan, Jared, Dhariwal, Prafulla, Neelakantan, Arvind, Shyam, Pranav, Sastry, Girish, Askell, Amanda, Agarwal, Sandhini, Herbert-Voss, Ariel, Krueger, Gretchen, Henighan, Tom, Child, Rewon, Ramesh, Aditya, Ziegler, Daniel M., Wu, Jeffrey, Winter, Clemens, Hesse, Christopher, Chen, Mark, Sigler, Eric, Litwin, Mateusz, Gray, Scott, Chess, Benjamin, Clark, Jack, Berner, Christopher, McCandlish, Sam, Radford, Alec, Sutskever, Ilya & Amodei, Dario 2020. Language Models are Few-Shot Learners. Search date 31.10.2023. http://arxiv.org/abs/2005.14165.

Brownlee, Jason 2019a. Understand the Impact of Learning Rate on Neural Network Performance. Search date 29.11.2023. https://machinelearningmastery.com/understand-the-dynamics-of-learning-rate-on-deep-learning-neural-networks/.

Brownlee, Jason 2019b. How to Configure the Learning Rate When Training Deep Learning Neural Networks. Search date 3.12.2023. https://machinelearningmastery.com/learning-rate-for-deep-learning-neural-networks/.

Brownlee, Jason 2019c. Train Neural Networks With Noise to Reduce Overfitting. Search date 3.12.2023. https://machinelearningmastery.com/train-neural-networks-with-noise-to-reduce-overfitting/.

Brownlee, Jason 2019d. How to Avoid Overfitting in Deep Learning Neural Networks. Search date 3.12.2023. https://machinelearningmastery.com/introduction-to-regularization-to-reduce-overfitting-and-improve-generalization-error/.

Brownlee, Jason 2016. Overfitting and Underfitting With Machine Learning Algorithms. Search date 29.11.2023. https://machinelearningmastery.com/overfitting-and-underfitting-with-machine-learning-algorithms/.

Bubeck, Sébastien, Chandrasekaran, Varun, Eldan, Ronen, Gehrke, Johannes, Horvitz, Eric, Kamar, Ece, Lee, Peter, Lee, Yin Tat, Li, Yuanzhi, Lundberg, Scott, Nori, Harsha, Palangi, Hamid, Ribeiro, Marco Tulio & Zhang, Yi 2023. Sparks of Artificial General Intelligence. Search date 31.10.2023. http://arxiv.org/abs/2303.12712.

Buhl, Nikolaj 2023. Training vs. Fine-tuning. Search date 3.12.2023. https://encord.com/blog/training-vs-fine-tuning/.

Chandhok, Shivam 2020. Zero-shot Learning. Search date 31.10.2023. https://learnopencv.com/zero-shot-learning-an-introduction/.

Chandramouli, Subramanian, Dutt, Saikat & Das, Amit 2018. Introduction to Machine Learning. Pearson Education India. Search date 30.10.2023. https://learning.oreilly.com/library/view/machine-learning/9789389588132/xhtml/chapter001.xhtml.

Charles, Stephen 2021. Dropout and Batch Normalization. Search date 3.12.2023. https://sourestdeeds.github.io/blog/dropout-and-batch-normalization/.

Chugh, Vidhi 2022. Tuning Adam Optimizer Parameters in PyTorch. Search date 3.12.2023. https://www.kdnuggets.com/tuning-adam-optimizer-parameters-in-pytorch.

Dasaradh, S.K. 2020. A Gentle Introduction To Math Behind Neural Networks. Search date 31.10.2023. https://towardsdatascience.com/introduction-to-math-behind-neural-networks-e8b60dbbdeba.

Devlin, Jacob, Chang, Ming-Wei, Lee, Kenton & Toutanova, Kristina 2019. BERT. Search date 31.10.2023. http://arxiv.org/abs/1810.04805.

Domingos, Pedro 2012. A few useful things to know about machine learning. Communications of the ACM 55 (10), 78–87. Search date 17.9.2023. https://doi.org/10.1145/2347736.2347755.

Gad, Ahmed 2022. A Comprehensive Guide to the Backpropagation Algorithm in Neural Networks. Search date 29.11.2023. https://neptune.ai/blog/backpropagation-algorithm-in-neural-networks-guide.

Gage, Philip 1994. A new algorithm for data compression. The C Users Journal archive (12), 23–38. https://api.semanticscholar.org/CorpusID:59804030.

Geetha M, Sri 2021. Natural Language Processing Using Python & NLTK. Search date 31.10.2023. https://medium.com/nerd-for-tech/natural-language-processing-using-python-nltk-5c1804d0962d.

Gençay, Ramazan & Qi, Min 2001. Pricing and hedging derivative securities with neural networks. Neural Networks, IEEE Transactions on 12 ,726–734. https://doi.org/10.1109/72.935086.

Géron, Aurélien 2018. Training Deep Neural Nets. O'Reilly Media, Inc. Search date 30.10.2023. https://learning.oreilly.com/library/view/neural-networks-and/9781492037354/ch02.html.

GitHub - Blubberli 2021. dqi_predictor. Search date 1.12.2023. https://github.com/Blubberli/dqi_predictor.

Glassner, Andrew 2021. Backpropagation. No Starch Press. Search date 3.12.2023. https://learning.oreilly.com/library/view/deep-learning/9781098129019/c14.xhtml.

Gu, Ken & Budhkar, Akshay 2021. A Package for Learning on Tabular and Text Data with Transformers. Association for Computational Linguistics. Mexico City, Mexico. Search date 1.12.2023. https://doi.org/10.18653/v1/2021.maiworkshop-1.10.

Gupta, Ritesh 2023. 10 Most Common Machine Learning Algorithms Explained -2023. Search date 31.10.2023. https://medium.com/@riteshgupta.ai/10-most-common-machine-learning-algorithms-explained-2023-d7cfe41c2616.

Gurusamy, Ilango 2018. Precision, recall, and the F1 score. Search date 29.11.2023. https://learning.oreilly.com/library/view/modern-scala-projects/9781788624114/308e20fb-9335-4209-adb2-9c48175e11de.xhtml.

Hany, John & Walters, Greg 2019. Gradient clipping, weight clipping, and more. Packt Publishing. Search date 3.12.2023. https://learning.oreilly.com/library/view/hands-on-generative-adversarial/9781789530513/86884abc-46a7-4c24-a180-e7380f8fb755.xhtml.

Hristov, Hristo 2022. Attention Mechanism in the Transformers Model. Search date 31.10.2023. https://www.baeldung.com/cs/attention-mechanism-transformers.

Hugging Face 2022. The Hugging Face Course. Search date 29.11.2023. https://huggingface.co/learn/nlp-course/chapter6/6.

Hugging Face a. Optimization. Search date 3.12.2023. https://huggingface.co/docs/transformers/main_classes/optimizer_schedules.

Hugging Face b. Model Outputs. Search date 3.12.2023. https://huggingface.co/docs/transformers/main_classes/output.

Hugging Face c. BERT. Search date 1.12.2023. https://huggingface.co/docs/transformers/model_doc/bert.

Hugging Face d. BERT. Search date 1.12.2023. https://huggingface.co/docs/transformers/model_doc/bert.

Ioffe, Sergey & Szegedy, Christian 2015. Batch Normalization. Search date 1.12.2023. https://doi.org/10.48550/arXiv.1502.03167.

Jain, Kanishk 2020. Evolution of NLP — Part 4 — Transformers — BERT, XLNet, RoBERTa. Search date 31.10.2023. https://medium.com/analytics-vidhya/evolution-of-nlp-part-4-transformers-bert-xlnet-roberta-bd13b2371125.

Jaitley, Urvashi 2019. Why Data Normalization is necessary for Machine Learning models. Search date 1.11.2023. https://medium.com/@urvashilluniya/why-data-normalization-is-necessary-for-machine-learning-models-681b65a05029.

Jarapala, Krishnakanth Naik 2023. Categorical Data Encoding Techniques. Search date 1.11.2023. https://medium.com/aiskunks/categorical-data-encoding-techniques-d6296697a40f.

jorge, liva 2023. Introduction to OpenAI's GPT-3 Model. Search date 31.10.2023. https://medium.com/@livajorge7/introduction-to-openais-gpt-3-model-a-game-changer-in-ai-powered-natural-language-processing-99a23acd29ba.

Keldenich, Tom 2021. Recall, Precision, F1 Score - Simple Metric Explanation Machine Learning. Search date 29.11.2023. https://inside-machinelearning.com/en/recall-precision-f1-score-simple-metric-explanation-machine-learning/.

Khanna, Chetna 2021a. WordPiece. Search date 24.11.2023. https://towardsdatascience.com/wordpiece-subword-based-tokenization-algorithm-1fbd14394ed7.

Khanna, Chetna 2021b. Byte-Pair Encoding. Search date 24.11.2023. https://towardsdatascience.com/byte-pair-encoding-subword-based-tokenization-algorithm-77828a70bee0.

Kundu, Rohit 2022. F1 Score in Machine Learning. Search date 3.12.2023. https://www.v7labs.com/blog/f1-score-guide, https://www.v7labs.com/blog/f1-score-guide.

LeCun, Yann, Bengio, Y. & Hinton, Geoffrey 2015. Deep Learning. Nature 521 ,1,436-44. https://doi.org/10.1038/nature14539.

Lendave, Vijaysinh 2021. A Guide to Text Preprocessing Using BERT. Search date 30.11.2023. https://analyticsindiamag.com/a-guide-to-text-preprocessing-using-bert/.

Li, Katherine (Yi) 2022. How to Choose a Learning Rate Scheduler for Neural Networks. Search date 30.11.2023. https://neptune.ai/blog/how-to-choose-a-learning-rate-scheduler.

Loshchilov, Ilya & Hutter, Frank 2019. Decoupled Weight Decay Regularization. Search date 29.11.2023. https://doi.org/10.48550/arXiv.1711.05101.

Masters, Dominic & Luschi, Carlo 2018. Revisiting Small Batch Training for Deep Neural Networks. Search date 2.12.2023. https://doi.org/10.48550/arXiv.1804.07612.

Nayak, Pandu 2019. Understanding searches better than ever before. Search date 31.10.2023. https://blog.google/products/search/search-language-understanding-bert/.

Newman, Nic, Fletcher, Richard, Eddy, Kirsten, Robertson, Craig T & Nielsen, Rasmus Kleis 2023. Digital News Report 2023. Reuters Institute. Search date 16.9.2023. https://reutersinstitute.politics.ox.ac.uk/digital-news-report/2023.

Ozdemir, Sinan 2023. Overview of Large Language Models. Addison-Wesley Professional. Search date 31.10.2023. https://learning.oreilly.com/library/view/quick-start-guide/9780138199425/ch01.xhtml.

Peixoto, Francke 2020. A Simple overview of Multilayer Perceptron(MLP). Search date 5.12.2023. https://www.analyticsvidhya.com/blog/2020/12/mlp-multilayer-perceptron-simple-overview/.

Piepenbreier, Nik 2022. PyTorch DataLoader. Search date 2.12.2023. https://datagy.io/pytorch-dataloader/.

Prakash, Pritesh 2021. An Explanatory Guide to BERT Tokenizer. Search date 28.11.2023. https://www.analyticsvidhya.com/blog/2021/09/an-explanatory-guide-to-bert-tokenizer/.

Prakash, Tripathi Aditya 2022. An Introduction to NLP (Natural Language Processing) with NLTK. Search date 31.10.2023. https://medium.com/codex/an-introduction-to-nlp-natural-language-processing-with-nltk-ede02367830c.

PyTorch 2023. MPS backend. Search date 30.11.2023. https://pytorch.org/docs/stable/notes/mps.html.

PyTorch a. PyTorch Tutorials 2.1.1+cu121 documentation. Search date 2.12.2023. https://pytorch.org/tutorials/beginner/basics/data_tutorial.html.

PyTorch b. PyTorch 2.1 Documentation. Search date 1.12.2023. https://pytorch.org/docs/stable/data.html.

Rajapakse, Thilina 2020. Understanding ELECTRA and Training an ELECTRA Language Model. Search date 31.10.2023. https://towardsdatascience.com/understanding-electra-and-training-an-electra-language-model-3d33e3a9660d.

Riva, Martin 2021. Interpretation of Loss and Accuracy for a Machine Learning Model | Baeldung on Computer Science. Search date 29.11.2023. https://www.baeldung.com/cs/ml-loss-accuracy.

Rothman, Denis & Gulli, Antonio 2022. Fine-Tuning BERT Models. Packt Publishing, Limited. Search date 31.10.2023. https://learning.oreilly.com/library/view/transformers-for-natural/9781803247335/Text/Chapter_03.xhtml.

Sarker, Iqbal H. 2021. Deep Learning. SN Computer Science 2 (6), 420. Search date 31.10.2023. https://doi.org/10.1007/s42979-021-00815-1.

Sarker, Iqbal H. 2021. Machine Learning. SN Computer Science 2 (3), 1–21. Search date 30.10.2023. https://doi.org/10.1007/s42979-021-00592-x.

Schuster, Mike & Nakajima, Kaisuke 2012. Japanese and Korean voice search. IEEE. Kyoto, Japan. Search date 24.11.2023. https://doi.org/10.1109/ICASSP.2012.6289079.

Shah, Deval 2023. Cross Entropy Loss. Search date 1.12.2023. https://www.v7labs.com/blog/cross-entropy-loss-guide, https://www.v7labs.com/blog/cross-entropy-loss-guide.

Shaik, Md. Ebrahim, Islam, Md. Milon & Hossain, Quazi Sazzad 2021. A review on neural network techniques for the prediction of road traffic accident severity. Asian Transport Studies 7 ,100040. Search date 31.10.2023. https://doi.org/10.1016/j.eastsj.2021.100040.

Smartocto 2021. Actionable user needs.

Song, Xinying & Zhou, Denny 2021. A Fast WordPiece Tokenization System. Search date 30.11.2023. https://blog.research.google/2021/12/a-fast-wordpiece-tokenization-system.html.

Stanford University CS231n Convolutional Neural Networks for Visual Recognition. Search date 3.12.2023. https://cs231n.github.io/assets/nn3/learningrates.jpeg.

Sun, Chi, Qiu, Xipeng, Xu, Yige & Huang, Xuanjing 2020. How to Fine-Tune BERT for Text Classification? Search date 4.12.2023. https://doi.org/10.48550/arXiv.1905.05583.

Upadhyay, Yash 2020. Role of Batch Size in optimization. Search date 2.12.2023. https://medium.com/@yashup1997/role-of-batch-size-in-optimization-c3f326798862.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz & Polosukhin, Illia 2017. Attention Is All You Need. Search date 31.10.2023. http://arxiv.org/abs/1706.03762.

Virtanen, Antti, Kanerva, Jenna, Ilo, Rami, Luoma, Jouni, Luotolahti, Juhani, Salakoski, Tapio, Ginter, Filip & Pyysalo, Sampo 2019. Multilingual is not enough. Search date 31.10.2023. http://arxiv.org/abs/1912.07076.

Wherlock, Jake 2021. How to use Datasets and DataLoader in PyTorch for custom text data. Search date 1.12.2023. https://towardsdatascience.com/how-to-use-datasets-and-dataloader-in-pytorch-for-custom-text-data-270eed7f7c00.

Xiao, Maggie 2020. Understanding Language using XLNet with autoregressive pre-training. Search date 31.10.2023. https://medium.com/@zxiao2015/understanding-language-using-xlnet-with-autoregressive-pre-training-9c86e5bea443.

Xu, Rachel, Sellers, Ewan & Salmi, E. F. 2023. Rock recognition and identification for selective mechanical mining. Bulletin of Engineering Geology and the Environment 82. https://doi.org/10.1007/s10064-023-03311-3.

Y Arcas, Blaise Agüera 2022. Do Large Language Models Understand Us? Daedalus 151 (2), 183–197. Search date 31.10.2023. https://doi.org/10.1162/daed_a_01909.

Yang, Lina & Liu, Wei 2021. Design of English Intelligent Simulated Paper Marking System. Complexity 2021. https://doi.org/10.1155/2021/5529114.

Zhuang, Fuzhen, Qi, Zhiyuan, Duan, Keyu, Xi, Dongbo, Zhu, Yongchun, Zhu, Hengshu, Xiong, Hui & He, Qing 2020. A Comprehensive Survey on Transfer Learning. Search date 31.10.2023. http://arxiv.org/abs/1911.02685.