Ida Toivanen

Semi-supervised deep learning for the classification of eldercare workers' sentiments

Master's Thesis in Information Technology

May 30, 2022

University of Jyväskylä

Faculty of Information Technology

Author: Ida Toivanen

Contact information: ida.m.toivanen@jyu.fi

Supervisors: Sami Äyrämö, and Susanne Jauhiainen

Title: Semi-supervised deep learning for the classification of eldercare workers' sentiments

Työn nimi: Puoliohjattu syväoppiminen vanhushoidon työntekijöiden tuntemusten luokitteluun

Project: Master's Thesis

Study line: Specialisation in Mathematical Modelling in Science and Decision Analytics

Page count: 61+2

Abstract: There exists extensive research for text classification, but only a handful of it is put into practice by deep neural networks that use semi-supervised learning – especially when semi-supervised deep neural networks are not trained in English, or other majorly studied languages. In this thesis we go through previous literature regarding semi-supervised deep learning methods for text classification, and then build a hands-on solution for three semi-supervised text classification methods. These methods are trained and tested on a small dataset, that is in Finnish. The results suggest that regularization methods should be taken into consideration when using semi-supervised methods for training – particularly when using smaller datasets that easily leads to overfitting. More research on regularization and Finnish deep learning models should be conducted to have a more comprehensive view on the applicability and reliability of text classification in natural language processing.

Keywords: classification, sentiment analysis, semi-supervised learning, deep learning, BERT

Suomenkielinen tiivistelmä: Tekstin luokitteluun on olemassa laaja tutkimuksen kirjo, mutta vain osa siitä on puoliohjattujen syvien neuroverkkojen pohjalta tehtyä – etenkin, kun opetusaineisto on ollut englannin kielellä, tai muulla huomattavan paljon tutkitulla kielellä. Tässä pro gradussa käymme läpi puoliohjattujen syväoppimismenetelmien kirjallisuutta tekstin luokittelussa, ja luomme käytännön toteutuksen kolmelle puoliohjatulle tekstin luokit-

telumenetelmälle. Nämä menetelmät opetetaan ja testataan pienenpuoleisella, suomenkielisellä aineistolla. Tulosten perusteella voitaisiin sanoa, että puoliohjattujen menetelmien yhteydessä on kannattavaa käyttää regularisointimenetelmiä ylisovittumisen ehkäisemiseksi, varsinkin kun opetusaineisto on pieni. Jotta voitaisiin saada kokonaisvaltaisempi kuva eri puoliohjattujen menetelmien kannattavuudesta ja luotettavuudesta luonnollisen kielen luokittelutehtävässä, olisi suomenkielisistä syväoppimismalleista ja regularisoinnista hyvä tehdä lisää tutkimusta.

Avainsanat: luokittelu, sävyanalyysi, puoliohjattu oppiminen, syväoppiminen, BERT

Glossary

ML	Machine Learning
NLP	Natural Language Processing
DL	Deep Learning
DNN	Deep Neural Network
SSL	Semi-Supervised Learning
GAN	Generative Adversarial Network
BERT	Bidirectional Encoder Representations From Transformers
UPT	Unsupervised Pretraining
MLM	Masked Language Model
VAT	Virtual Adversarial Training
PL	Pseudo Labeling

List of Figures

Figure 1. Multilayer perceptron, or feedforward neural network architecture with two hidden layers (figure adapted from Haykin 2010). 8
Figure 2. GAN architecture, where G is generator, F is fake data samples, D is discriminator and R is real data samples (figure adapted from Vlachostergiou et al. 2018 and Croce, Castellucci, and Basili 2020). 9
Figure 3. Transformer architecture (adapted in a simplified form from Vaswani et al. 2017).11
Figure 4. Scaled Dot Product Attention architecture (adapted from Vaswani et al. 2017). . 12
Figure 5. Multi-Head Attention architecture (adapted from Vaswani et al. 2017). 12
Figure 6. Taxonomy of SSL text classification research that is used as the guideline of literature review and implementation part of this thesis (adapted in a simplified form from Van Engelen and Hoos 2020). 15
Figure 7. The usual model construction scheme for BERT modeling that involves pretraining and finetuning. Here L is labeled data, and U is unlabeled data. 16
Figure 8. Idea behind pseudo labeling described in two phases (following the idea presented in Lee 2013). Here L is labeled data, U is unlabeled data, and PL is pseudo labeled data. 19
Figure 9. GAN-BERT architecture, where G is generator, F is fake data samples, D is discriminator, L is labeled data samples, and U is unlabeled data samples (figure adapted from Croce, Castellucci, and Basili 2020). 23
Figure 10. Creating datasets involved two steps: 1) dividing data into labeled, unlabeled and testing datasets, and 2) dividing labeled and unlabeled datasets further into training and validation datasets. Here UPT model is unsupervised pretraining model (presented in section 5.2.2), PL model is pseudo labeling model (5.2.3) and GAN model is generative adversarial network model (5.2.4). 28
Figure 11. Baseline model architecture in a simplified form. 32
Figure 12. Neural network without and with dropouts, where red units represent the units that are dropped out (figure adapted from Srivastava et al. 2014). 33
Figure 13. Two versions for pseudo labeling: training a model with pseudo labeling (version 1) and with iterative pseudo labeling (version 2), where L is labeled dataset, U_{All} is unlabeled dataset including all samples, U_{Half_1} is unlabeled dataset including first half of the samples, U_{Half_2} is unlabeled dataset including second half of the samples, PL model 1 is pseudo labeling model 1, PL model 2 is pseudo labeling model 2 and PL model 3 is pseudo labeling model 3. 35

List of Tables

Table 1. The amount of data samples belonging to each class. 27
Table 2. All the best model results for validation dataset. Here UPT refers to unsupervised pretraining model, PL to pseudo labeling model and GAN to generative adversarial network model. 38

Table 3. All the best model results for testing dataset. Here UPT refers to unsupervised pretraining model, PL to pseudo labeling model and GAN to generative adversarial network model. 38
Table 4. PL model results for version 1 and 2 when evaluating with validation and testing datasets. 39

Contents

1 INTRODUCTION 1

2 THEORETICAL BACKGROUND 3
2.1 Machine learning 3
2.2 Deep learning for natural language processing 9

3 SEMI-SUPERVISED DEEP LEARNING FOR TEXT CLASSIFICATION 15
3.1 Unsupervised pretraining 16
3.2 Pseudo labeling 19
3.3 Generative models 22
3.4 Other semi-supervised methods 25

4 DATA 26

5 METHODS 29
5.1 Data preprocessing 29
5.2 Models 31
5.2.1 Baseline 32
5.2.2 Unsupervised pretraining 34
5.2.3 Pseudo labeling 34
5.2.4 Generative adversarial network 36
5.3 Evaluation 37

6 RESULTS 38

7 DISCUSSION 40

8 CONCLUSIONS 44

BIBLIOGRAPHY 45

APPENDICES 55
A FinBERT model and tokenizer configuration files 55

1 Introduction

At the moment, in machine learning and natural language processing there is a prevalence in the research of languages with larger count of speakers (e.g. English). Due to there being fewer resources on languages that have smaller audience, the amount of research done for these languages is also something few and far between. One of these languages is Finnish.

The aim of this thesis is to conduct a text classification task, or more specifically a sentiment analysis task (for three classes: positive, negative and neutral), on a Finnish dataset concerning the opinions and sentiments that are evoken in careworkers of elderly people when they use technology in their work. This is encapsulated in the research question that goes as follows: "What deep neural networks give the best performance for semi-supervised classification of eldercare workers' sentiments?".

This thesis focuses on semi-supervised learning methods that are implemented for textual data. Semi-supervised learning refers to the model being developed to utilize data that is partially annotated – thus, falling in between supervised (all annotated) and unsupervised (no annotations) learning. Through semi-supervised learning we are able to use the few labels we have, and include more data to better the performance, even when there is no labels available for it, or when preparing them would take too much time or resources.

When talking about sentiment analysis particularly for Finnish, there are a few papers that have developed methods for the task (e.g. a study by Vankka et al. 2019). Most recent Finnish sentiment analysis research, that is relevant to our work, seems to be the making of a social media corpus that is annotated with sentiment polarity (Lindén, Jauhiainen, and Hardwick 2020), and supervised FinBERT-finnsentiment model[1] built on it by finetuning FinBERT (Virtanen et al. 2019).

This thesis can generally be divided into two parts: the first part consisting of literature related to our topic (chapters 2–3), and the second part being predominantly about the constructive part, that contains information about the implementation work (chapters 4–7). In chapter 2 we introduce concepts that are necessary to understand when developing machine

1. https://huggingface.co/fergusq/finbert-finnsentiment

learning models and discussing what lays in the analysis of natural language processing with deep learning methods. These concepts are the foundation for the scientific work presented in chapter 3 that contains a literature review of semi-supervised deep learning for text classification. The constructive, more hands-on, part of this thesis is described in chapters 4–7. In these chapters we describe the data used in our thesis (chapter 4), give an overview of used methods to classify this data (chapter 5), report the concise results that are born from using these methods (chapter 6), and finally, we go through these results in a more detailed manner in discussion (chapter 7). In chapter 8 we offer conclusions, and give an ending statement about the successfulness of our implementations.

2 Theoretical background

In this chapter we go through theoretical background that is necessary for understanding where the work in chapter 3 is based on. First we explain the common concepts behind machine learning (2.1). After that, we introduce concepts behind deep learning for natural language processing (2.2), that include deep architectures developed for text data processing.

2.1 Machine learning

Machine learning (ML) is the field of computer science where the model is excepted to learn based on the data that is input in the model. ML algorithms gather observations from the information without explicitly assigning rules that would lead the model to produce the results (Minaee et al. 2021).

ML methods can generally be placed under one of three subcategories: supervised, unsupervised or semi-supervised learning (Goodfellow, Bengio, and Courville 2016), based on whether the implementation task involves annotated data (supervised learning), such as having labels of species for plants on a classification task, whether the task is not explored via explicit annotations affiliated with data samples (unsupervised learning), or whether the task at hand is approached by using both labeled data and unlabeled data (semi-supervised learning). To name a few representative examples of these ML methods, support vector machines and gradient boosters are cases of supervised, and K-means clustering and autoencoders of unsupervised methods. Semi-supervised learning (SSL) is usually implemented with both supervised and unsupervised approaches, and thus it falls somewhere between the two. In SSL, unlabeled data samples U from $P(\mathbf{x})$ and labeled data samples L from $P(\mathbf{x}, \mathbf{y})$ are used for the estimation of $P(\mathbf{y}|\mathbf{x})$, where $\mathbf{x}$ is input vector and $\mathbf{y}$ is label corresponding to $\mathbf{x}$ (Goodfellow, Bengio, and Courville 2016).

In addition to choosing the type of learning for a task, another crucial point is to decide where the focus of the study will be and what is the essential topic of the study. For example, in natural language processing (NLP), a subfield of ML that deals with the analysis natural languages (like English), these ML tasks can be one of the following: classification,

sentiment analysis, machine translation, question answering, language modeling, text augmentation, text generation, part-of-speech tagging or named entity recognition. For example, in a classification task data sample is supposed to be put into one of k categories which can be described by a function $f : \mathbb{R}^n \to \{1, ..., k\}$ that is to be produced by a learning algorithm (Goodfellow, Bengio, and Courville 2016). When a data sample $\mathbf{x}$ is input into the learning algorithm, it should give out a category y_{out} as an output, where $y_{\text{out}} = f(\mathbf{x})$ (Goodfellow, Bengio, and Courville 2016).

In order to design a model, learning algorithm is applied to data. Data is divided into training data samples and testing data samples, out of which training data is used to choose the model best in performance out of many candidate models (Haykin 2010). From the training data a subset for validation of the model is drawn for model validation. Because the model may overfit to the validation data, the testing data is used to test the generalization ability of the model (Haykin 2010). In order for the model to be able to generalize well to unseen data, the model should learn to fit the training data appropriately. Overfitting occurs when the model learns the training samples too well – in other words, the model fails to map the underlying function due to learning irrelevant data features (Haykin 2010). Underfitting is the opposite type of error that can be committed by the model. In this case the model is usually too simple for the data and not being able to map the underlying function correctly.

Certain steps are to be performed in order to use a model properly for the task it is intended to carry out. This usually means that in the model construction process data preprocessing is required at first so that the data at hand is in a form that is possible to utilize by the model. Then after choosing a model its performance is optimized with hyperparameter optimization.

Preprocessing

The most basic preprocessing includes the removing of duplicate or empty values, if they do not bring any important information to the model. For text data, implemented preprocessing methods may also include word embeddings, tokenization, and/or handling stop words, capitalization, punctuation and special characters (like emojis) (Kowsari et al. 2019).

Word embedding can be denoted as $\mathbf{e} \in \mathbb{R}^d$, where d is the dimension of the vector to which each word from vocabulary is mapped to (Kowsari et al. 2019). Traditional word embedding

methods include Word2Vec (Mikolov et al. 2013), GloVe (Pennington, Socher, and Manning 2014) and fastText (Bojanowski et al. 2017). All of these approaches have more to do with n-grams, bag-of-words and other unsupervised methods for dealing with text data.

Tokenization is the act of separating the sentences into separate parts, "tokens", that may be words, punctuation or other distinctive stand-alone parts used in sentences (especially vector parts) (Kowsari et al. 2019). In addition to simple tokenization, pretrained tokenizers can have additional features which can be used to e.g. add special tokens (e.g. "[CLS]" token for indicating the beginning of a data sample, or "[SEP]" token for indicating the ending), or to pad tokenized data samples to make them of the same size. During tokenization, it is also possible to set a maximum value of the amount of tokens that is allowed for a data sample. This limit for data sample size is called maximum sequence length.

Depending on the model type, the type of preprocessing steps that are needed may vary. For example, for neural networks based on BERT (Devlin et al. 2018), the deletion of stop words (i.e. words deemed irrelevant, such as "be" or "also") may be more detrimental than helpful for the model for learning a task as the sentence is being taken into account as whole. The same untouching of words goes to capitalization, punctuation and special characters for BERT modeling (of cased models that are sensitive to uppercase and lowercase letters; uncased models do not share this quality). Capitalization refers to having words written with capitalized letters (uppercase "M" instead of lowercase "m"). Usually for punctuation characters like dots and question marks are included. Special characters can be emojis, for example. In BERT these special characters are tokenized with UNK so the model can ignore them if they are unknown to the model [1], or if the meanings of the characters have been separately specialized to the model, they will have those specialized tokens assigned to them.

While these preprocessing steps are applicable to English among other languages, besides English there are languages that have other properties to be possibly taken into account. For example, accented letters can be transformed into their unaccented "counterparts", as was done when training FinBERT – a Finnish BERT based model – and letters "ä" and "ö" were

1. https://huggingface.co/docs/transformers/internal/tokenization_utils#transformers.PreTrainedTokenizerBase.encode_plus

switched into "a" and "o", respectively (Virtanen et al. 2019).

One data preparation stage may also be data augmentation, or the addition of synthetic data, which is useful especially if data is scarce. For language processing tasks, back translation is another method to add data. By using language translation applications, translating a sentence in language A to a new language B, and using that translation to get the same sentence in language A again, a new paraphrasing is obtained that can be used along with the original data (Xie et al. 2020). Naturally, if you are dealing with sensitive data and its use is restricted contractually, back translation might not be the most ideal data augmentation method.

Hyperparameter optimization

Substantial part of model training process is hyperparameter optimization. For example, gradient based deep neural networks are optimized by making changes in their hyperparameters. Gradient based calculation is based on an algorithm that uses derivatives of the function, and it is called gradient descent (Goodfellow, Bengio, and Courville 2016). There is many variations to optimization algorithms, or optimizers, that encompass the way gradients are calculated (Bengio 2012), and they can also affect majorly the way a model converges. For example Adam (Kingma and Ba 2014), AdamW (Loshchilov and Hutter 2017), and LAMB (You et al. 2019) are optimizers.

Hyperparameters related to gradient based modeling include hyperparameters for neural network (like initial learning rate, learning rate scheduler, mini-batch size, and number of training iterations) and the model and training criterion hyperparameters (like number of hidden units, weight decay, and random seeds) (Bengio 2012).

Initial learning rate is the starting point for the model training, usually set to below 1 (e.g. 0.1 or 0.005) (Bengio 2012). Learning rate is a hyperparameter used when calculating gradient in gradient based optimization tasks. Too large of a learning rate may negatively contribute to leading the model to a global minimum and make the average loss increase (Bengio 2012), but too small of a learning rate might prove to be a unnecessarily slow wait on the model converging, on the other hand. The use of learning rate schedulers, like adaptive learning rates, help set the learning rate throughout the whole training process. That is, in each iteration the

learning rate is adjusted in some preset way – such as linearly decreasing. Other schedulers include constant, cosine, cosine annealing, and cosine with warmups, for example.

To ease the training process and handling of gradients, the training dataset is usually divided into several batches or mini-batches (Bengio 2012). Batch size (of 8, 16, 32, 64 for example) describes the amount of data samples included in one batch, a larger batch size resulting to faster performance (at the expense of GPU memory).

Usually models are attempted to run through multiple iterations, or epochs, until the end but sometimes having many number of training iterations may be counterproductive to the model performance. The model training can be brought to a halt based on a set condition, such as after ten iterations in training when the model loss metric stops decreasing or accuracy metric is not improved on. This is called early stopping (Bengio 2012).

Deep neural networks (DNNs) usually have hidden layers which consist of hidden units. Extensive research has been conducted on how to decide the number of hidden units (Goodfellow, Bengio, and Courville 2016), but there is no definitive paradigm or consensus on the amount for one hidden layer, yet alone for the number of units for combinations of different hidden layers. One deciding factor in the matter may be the properties of the data (Bengio 2012). The visualization of hidden units on hidden layers can be seen in figure 1. This figure depicts the most basic type of DNN, multilayer perceptron, or feedforward network in other words, and it typically consists of an input, an output and hidden layer(s). When it is used for a classification task, a mapping $\mathbf{y}$ can be defined as follows:

$$\mathbf{y} = f(\mathbf{x}; \theta), \tag{2.1}$$

where $\mathbf{x}$ is input vector, and θ is parameters to some function f (Goodfellow, Bengio, and Courville 2016).

Random seeds are used to bring randomness in to the model training process. This randomness can be seen in initialization point of the model training, or when sampling the training data (Bengio 2012). Random seeds can be used to better reproducibility of model making by using the same seed for the initialization[2].

2. https://pytorch.org/docs/stable/notes/randomness.html

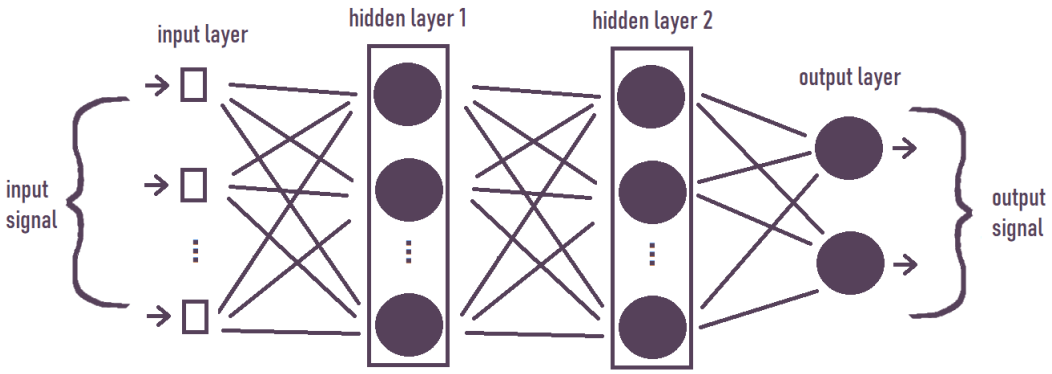

Figure 1. Multilayer perceptron, or feedforward neural network architecture with two hidden layers (figure adapted from Haykin 2010).

When updating the gradients each epoch, weight decay is another term that is added for regularization, preventing overfitting from happening (Bengio 2012).

In the model architecture several different activation functions can be utilized. Activation function is used at the final layer for calculating values in the preceding hidden layer (Goodfellow, Bengio, and Courville 2016). Depending on the type of problem and neural network, the following activation functions are probable choices: ReLU (rectified linear unit), leaky ReLU, and softmax. For example ReLU can be defined as (Goodfellow, Bengio, and Courville 2016):

$$g_r(\mathbf{x})_i = \max(0, x_i), \tag{2.2}$$

where $\mathbf{x}$ is the input vector. This means that there are potentially values of zero for gradients present during optimization. Leaky ReLU is a variant of ReLU, where zero gradients are replaced with non-zero gradients, and it can be defined as (Maas, Hannun, Ng, et al. 2013, and B. Xu et al. 2015):

$$g_l(\mathbf{x})_i = \max(\frac{x_i}{a}, x_i), \tag{2.3}$$

where $\mathbf{x}$ is the input vector, and a is some constant value, e.g. 100. Softmax function, that is usually used in classification tasks at the final layer, can be defined as (Goodfellow, Bengio, and Courville 2016):

$$\text{softmax}(\mathbf{x})_i = \frac{\exp(x_i)}{\sum_{j=1}^{n} \exp(x_j)}, \tag{2.4}$$

where $\mathbf{x}$ is the input vector, and n is the size of the sum of exponentials of $\mathbf{x}$.

2.2 Deep learning for natural language processing

Deep learning (DL), or synonymously used deep neural networks (DNNs), is the subfield of ML where representation learning does not take form only in the visible layers, but in hidden layers as well (Goodfellow, Bengio, and Courville 2016). These hidden layers are the pillars of more abstract feature learning, extracting features that the model deems important for the task at hand. In order to apply DNNs to the field of NLP, certain type of network architectures need to be used. In this section we go through several DNN architectures that are relevant to the NLP field and our research. These architectures include generative adversarial network, transformers, and BERT.

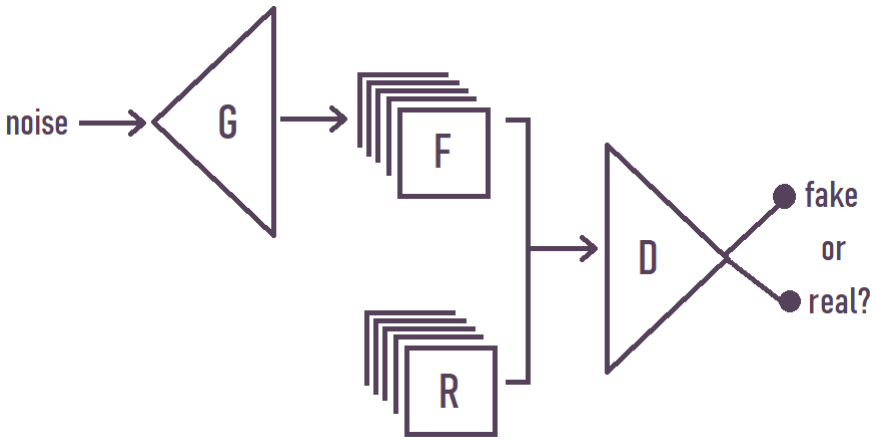

Figure 2. GAN architecture, where G is generator, F is fake data samples, D is discriminator and R is real data samples (figure adapted from Vlachostergiou et al. 2018 and Croce, Castellucci, and Basili 2020).

Generative adversarial network

Generative adversarial network (GAN) consists of a generator G and a discriminator D, that work together by having opposing functions. While the generator produces fake data samples F from noise, to "deceive" the discriminator, the discriminator is trying to classify the fake data samples and real data samples R to their corresponding classes of "fake" and "real" (see figure 2). Introduced by Goodfellow et al. 2014, the researchers suggest that the joint training of discriminator D and generator G can be thought as a minimax game for two, and define it

as follows:

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}\sim p_{\text{data}}(\mathbf{x})}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}\sim p_{\mathbf{z}}(\mathbf{z})}[\log(1-D(G(\mathbf{z})))], \tag{2.5}$$

where $V(D,G)$ is value function, $\mathbf{x}$ real data samples, $G(\mathbf{z})$ fake data samples, and $p_{\mathbf{z}}(\mathbf{z})$ is prior on input noise variable to learn the distribution p_g of generator G over data $\mathbf{x}$. If both G and D are differentiable functions represented by multilayer perceptrons with parameters θ_g and θ_d, respectively, the learning of the distribution p_g may be represented as a mapping to data space as $G(\mathbf{z};\theta_g)$, and the learning of the distribution p_d may be represented as a mapping to data space as $D(\mathbf{x};\theta_d)$. In the function the probability of right assignment of labels ("fake" and "real") is maximized for D, and the probability of having difference between the fake data samples and real data samples is minimized (Goodfellow et al. 2014).

Transformer

Transformer models abandon the use of traditional DNNs (e.g. recurrent neural network) altogether, and instead utilize attention mechanisms for NLP tasks (Vaswani et al. 2017). Transformer is built upon an encoder and a decoder which both consist of a stack of layers (see figure 3). Let us define an input sequence of symbol representations $\mathbf{x} = (x_1,...,x_n)$, that is mapped by the encoder to a sequence of continuous representations $\mathbf{z} = (z_1,...,z_n)$. Given this output $\mathbf{z}$ of encoder, the decoder generates an output sequence $(x_1,...,x_m)$ that consists of symbols one element at a time. One of the base structures used in transformers is scaled dot product attention. Scaled dot product attention can be defined as follows (Vaswani et al. 2017):

$$\text{Attention}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{softmax}(\frac{\mathbf{Q}\mathbf{K}^T}{\sqrt{d_k}})\mathbf{V}, \tag{2.6}$$

where $\mathbf{Q}$ is matrix of queries, $\mathbf{K}$ matrix of keys, $\mathbf{V}$ matrix of values, and d_k the dimension of keys and queries, and d_v is the dimension of values. After performing matrix multiplication for $\mathbf{Q}$ and $\mathbf{K}$, a series of operations are to be performed for them (including scaling with factor $\frac{1}{\sqrt{d_k}}$, possible masking and softmax) after which another matrix multiplication is performed together with $\mathbf{V}$, producing an output (see figure 4). Compatibility function of the query (with corresponding keys) is used for calculating weights assigned to each of the values of the output, which is a weighted sum of the values (Vaswani et al. 2017).

To make efficient use of attention, multiple attention heads are concatenated to create multi-

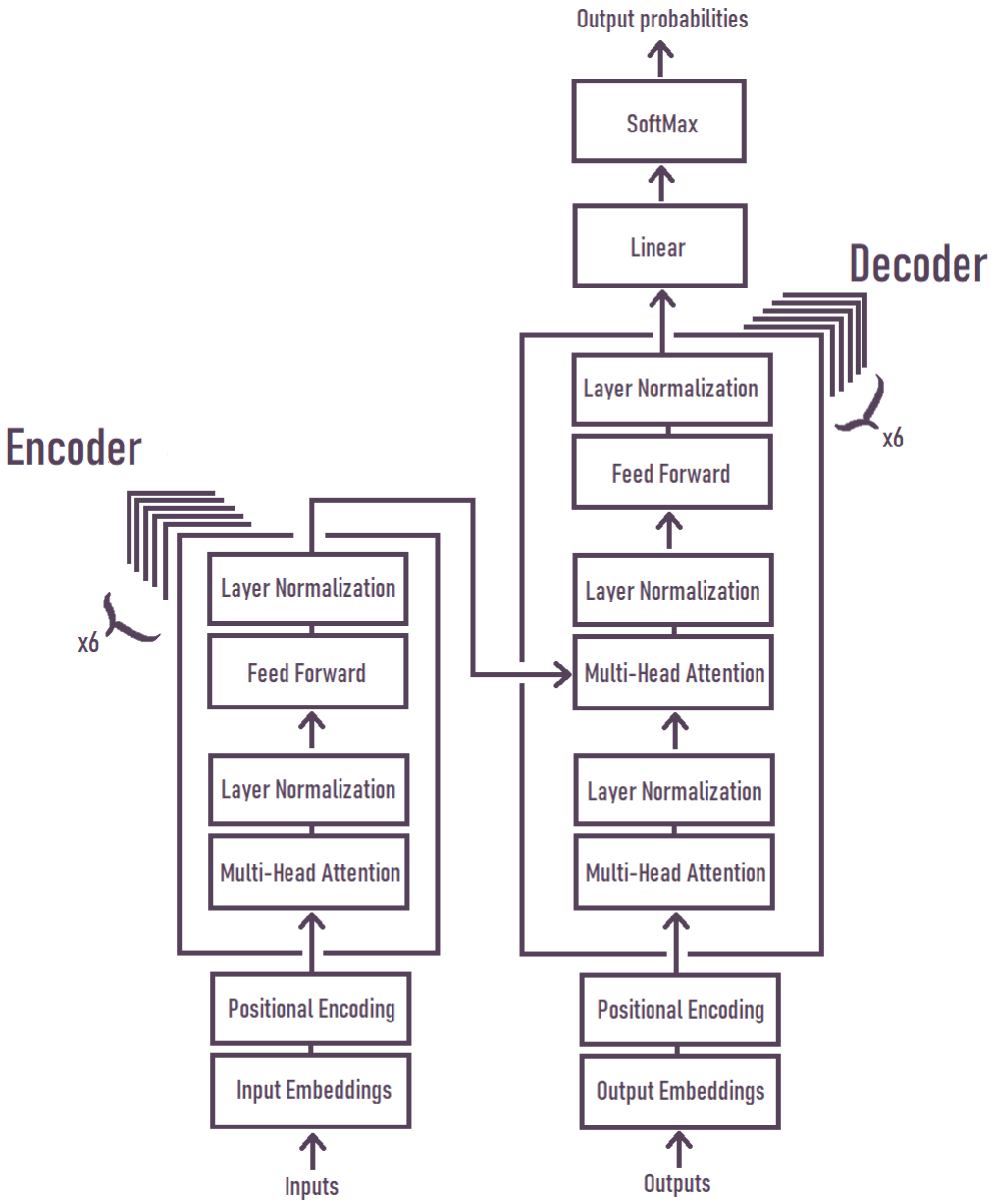

Figure 3. Transformer architecture (adapted in a simplified form from Vaswani et al. 2017).

head self-attention (see figure 5). Multi-head attention can be defined as follows (Vaswani et al. 2017):

$$\text{MultiHead}(\mathbf{Q},\mathbf{K},\mathbf{V}) = \text{Concat}(\text{head}_1,...,\text{head}_h)\mathbf{W}^O, \tag{2.7}$$

where a head can be defined as (Vaswani et al. 2017):

$$\text{head}_i = \text{Attention}(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V), \tag{2.8}$$

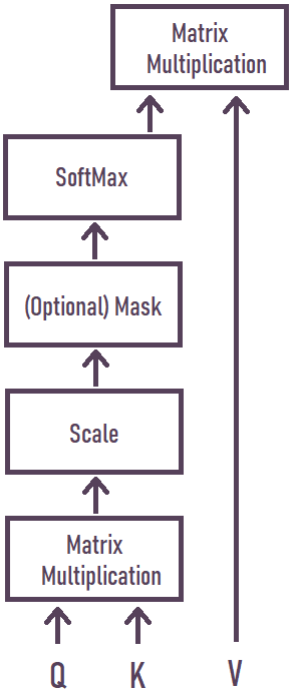

Figure 4. Scaled Dot Product Attention architecture (adapted from Vaswani et al. 2017).

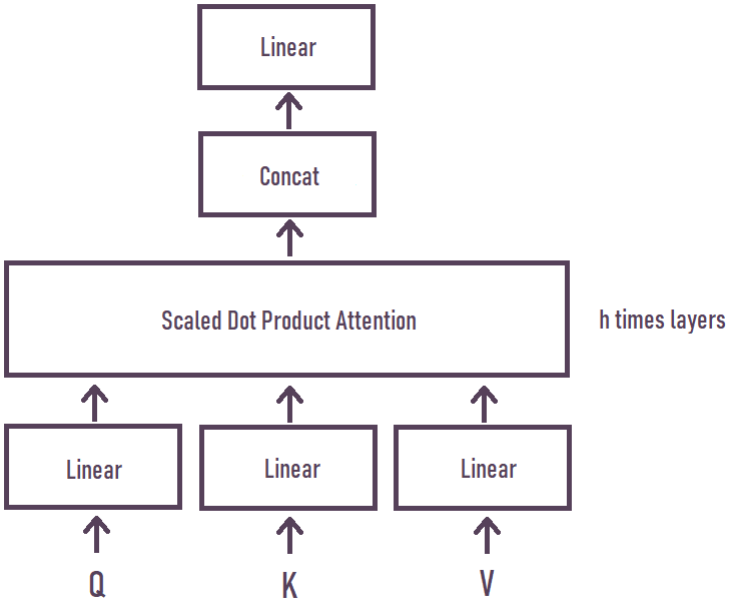

Figure 5. Multi-Head Attention architecture (adapted from Vaswani et al. 2017).

where parameter matrices $\mathbf{W}^O \in \mathbb{R}^{hd_v \times d_{\text{model}}}$, $\mathbf{W}_i^Q \in \mathbb{R}^{d_{\text{model}} \times d_k}$, $\mathbf{W}_i^K \in \mathbb{R}^{d_{\text{model}} \times d_k}$ and $\mathbf{W}_i^V \in \mathbb{R}^{d_{\text{model}} \times d_v}$ are projections. Unlike the scaled dot product attention, with multi-head attention linear projections are performed h times for $\mathbf{Q}$, $\mathbf{K}$ and $\mathbf{V}$ to dimensions d_k, d_k and d_v, in the corresponding order. After the projections, we end up with output values of dimension d_v, that are concatenated and projected again to produce final values of the output of multi-head attention (Vaswani et al. 2017).

In addition to attentional structures, the transformer makes use of position-wise feed-forward network, layer normalization, positional encoding, embeddings and softmax. Position-wise feed forward networks refer to a fully connected feed forward network, that is applied identically and separately to every position, and which is incorporated in every layer of the decoder and the encoder. In this context, feed forward network can be defined as (Vaswani et al. 2017):

$$\text{FFN}(\mathbf{x}) = \max(0, \mathbf{x}\mathbf{W}_1 + b_1)\mathbf{W}_2 + b_2, \tag{2.9}$$

where $\mathbf{x}$ is input vector, $\mathbf{W}_1$ and $\mathbf{W}_2$ weight matrices, and b_1 and b_2 bias values. Layer normalization is applied to every feedforward network and multi-head attention in the transformer network (see figure 3). Layer normalization refers to the type of normalization where variance and mean values are calculated as summed inputs of all neural units in a layer (Ba, Kiros, and Hinton 2016). The mean can be defined as follows:

$$\mu^l = \frac{1}{H}\sum_{i=1}^{H} a_i^l, \tag{2.10}$$

and the variance as follows:

$$\sigma^l = \sqrt{\frac{1}{H}\sum_{i=1}^{H}(a_i^l - \mu^l)^2}, \tag{2.11}$$

where a_i^l represents the summed inputs to the neurons in a layer l, and H is the number of hidden units in a layer (Ba, Kiros, and Hinton 2016). Positional encoding refers to a value that is concatenated to input embeddings, providing location information about the tokens in the sequence to the model. Embeddings are utilized to present input and output tokens in vectoral form in the same dimension as positional encodings, in dimension d_{model} (Vaswani et al. 2017). Softmax (see equation 2.4) is also used for the conversion of the outputs of the decoder into next-token probabilities that are predicted (Vaswani et al. 2017).

There are multiple transformers variations, ELECTRA (Clark et al. 2020) and XLNet (Z. Yang et al. 2019) being a few of the more recent ones. One extensively studied variation is called BERT.

BERT

Based off of the architecture of Transformers, BERT (Bidirectional Encoder Representations From Transformers) is a type of framework that was designed to extend to several different NLP tasks, referred to as transfer learning, without the aid of task-specific architecture changes. Instead, this generalizing power of BERT was achieved with masked language model (MLM) – masking of random input tokens that are to be predicted – while the context is being taken into account bidirectionally (Devlin et al. 2018). Let us denote a token as $\mathbf{s_t}$, that we replace with [MASK]. MLM can then be described with $\mathbf{S}_{\setminus t} := (\mathbf{s_1}, ..., \mathbf{s}_{t-1}, \mathbf{s}_{t+1}, ..., \mathbf{s}_{|\mathbf{S}|})$, where $\mathbf{S}_{\setminus t}$ refers to all the future and past tokens that are used for the prediction of the token $\mathbf{s_t}$ (Salazar et al. 2019). Besides MLM, next sentence prediction is used to give additional information to the model about sentence relationships, to extend BERT to tasks like natural language inference. In next sentence prediction, the model is given two sentences that either are related (so the second sentence logically follows the first sentence), which is labelled as IsNext, or are not related (the second sentence is not continuation of the first sentence), which is labelled as NotNext (Devlin et al. 2018). Training process terms related to BERT include pretraining and finetuning. BERT can be thought as a pretrained model body, that is ready for use only after feeding it your own data. Usually this feeding of data is done via pretraining and/or finetuning. For pretraining MLM is used to allow unsupervised training of the model, and it is usually done for general improvement of the model. Finetuning can be done after or without previous pretraining, and it refers to supervised training of the model to specify the model to a downstream task. There exists many variations to BERT – RoBERTa (Liu et al. 2019), ALBERT (Lan et al. 2019), DistilBERT (Sanh et al. 2019) and DeBERTa (He et al. 2020) being some of the other BERT models.

3 Semi-supervised deep learning for text classification

To have an encompassing view of the semi-supervised field in ML, one should take a look at the review done by X. Yang et al. 2021, in which the authors suggest that the taxonomy of deep SSL should be consisting of generative methods, consistency regularization methods, graph based methods, pseudo labeling methods and hybrid methods. This division was done by having evidence mainly for image classification studies. To have narrow enough focus on semi-supervised NLP for text classification, we choose to follow the taxonomy presented in Van Engelen and Hoos 2020 instead. In this chapter, following the taxonomy in Van Engelen and Hoos 2020 that was created to describe SSL methods that were extended from supervised framework ("inductive methods"), we decide to divide SSL methods for DNNs into three separate groups: including unlabeled data via unsupervised pretraining (section 3.1), pseudo labeling (section 3.2), or generative models (section 3.3). This taxonomy, that we use, can be seen from figure 6.

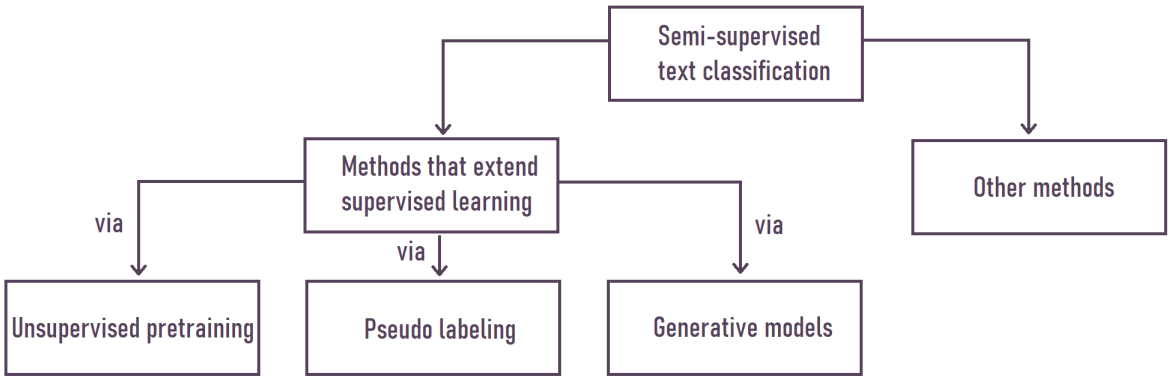

Figure 6. Taxonomy of SSL text classification research that is used as the guideline of literature review and implementation part of this thesis (adapted in a simplified form from Van Engelen and Hoos 2020).

In this literature review we focus on methods that are built on transformers, leaving out studies that cover more traditional DNNs (e.g. LSTMs) for text classification. These type of studies are very briefly discussed in section 3.4.

3.1 Unsupervised pretraining

When there is a lack of annotations for the data when working with DNNs, that usually require massive amounts of data, one way to approach the harnessing of unlabeled data is to utilize unsupervised pretraining. Unsupervised pretraining has proven to be useful in advancing the generalization ability of the model that is later trained with supervised learning (Dai and Le 2015). Using the pretrained model with unsupervised means enables the use of all available information (unlabeled data) without needing to apply hands-on annotation work, that usually take much time and resources.

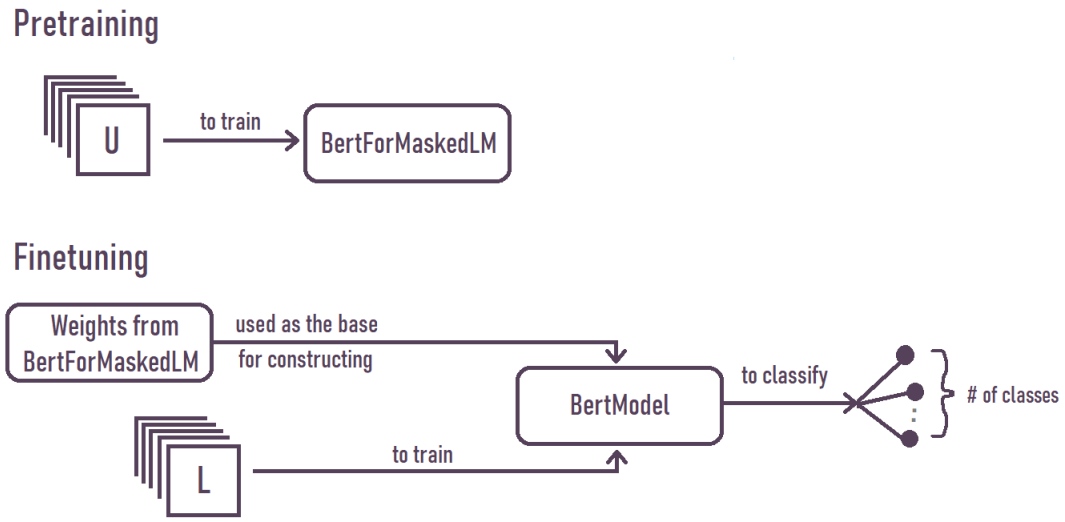

Figure 7. The usual model construction scheme for BERT modeling that involves pretraining and finetuning. Here L is labeled data, and U is unlabeled data.

When comparing the performances of BERT based models (Devlin et al. 2018), unusupervised pretraining can be understood via MLM pretraining objective. MLM essentially is a way to pretrain a model with unlabeled data by having tokens randomly masked and their corresponding vocabulary identifiers predicted. BERT based modeling can generally be thought to be consisting of pretraining (unsupervised learning) and finetuning (supervised learning). During pretraining, unlabeled data U is used to train BERT model for MLM, "BertForMaskedMLM". After that, the finetuning is done by using the weights from the pretrained BertForMaskedMLM model for constructing supervised BERT based model, "BertModel", and it is in turn trained with labeled data L. Then this BERT model is used to classify new

data samples. This modeling process, that involves first pretraining and then finetuning, is visualized in figure 7. The difference in the performance of a finetuned model compared to a pretrained and finetuned model, is described later on in the section 3.2, where we describe a study that also involves pseudo labeling (Sun et al. 2020).

In a study by Gururangan et al. 2019, a traditional neural network, namely variational autoencoder based model, fails in performance of text classification tasks when put next to a finetuned BERT, and is unsuccessful in almost all cases when compared to a finetuned ELMo (Peters et al. 2018). Both BERT and ELMo were pretrained with unlabeled data before finetuning them with labeled data – both unlabeled and labeled data being the same in-domain data. Labeled dataset sizes being 200, 500, 2500, 10 000, BERT accuracy (%) results for different datasets were the following: for IMDB 88.1, 89.4, 91.4, 93.1; for AG 87.1, 88.0, 90.1, 91.9; for Yahoo 45.3 (one ELMo model being exceptionally at 60.9), 69.2, 76.9, 81.0, and for Hatespeech 76.2, 78.3, 79.8, 80.2.

In the study of Li and Qiu 2020, BERT was again used as the backbone for a model, and conjoined with virtual adversarial training (VAT). This regularization method embodies the supplementing of small perturbations to normalized word embeddings throughout training, and was first introduced by Miyato, Dai, and Goodfellow 2016. Instead of just implementing VAT, Li and Qiu 2020 extend the framework and work towards Token-Aware VAT ("TAVAT") to prevent initialization and constraint problems. This was done by implementing global perturbation vocabulary to prevent accumulation of noise, that perturbations with random initialization could cause in text data, so that similar perturbation initialization is done for same tokens that exist in different sequences. Additionally, in order to use perturbation on token-level instead of sentence-level, TAVAT differentiates between tokens that are packed with more information and tokens that are irrelevant in terms of the task at hand. This is reflected in having larger perturbation for more crucial tokens, and having more constraining on unimportant tokens. The model developed in the study was tested against many tasks on GLUE benchmark, of which only SST-2 is a text classification task. Using GLUE datasets, for evaluation on development set BERT base with TAVAT scored 93.7% in accuracy (vs. 92.7% obtained with BERT base by Devlin et al. 2018), and for evaluation on test set BERT base with TAVAT scored 94.5% in accuracy (vs. 93.5% obtained with BERT base). When

compared to other current state-of-the-art methods, 3.3% improvement (accuracy of 97%) for the development set was obtained with the use of backbone ALBERT (xxlarge v2) along with FreeLB (Zhu et al. 2019), that is another framework for using BERT models enhanced with adversarial training. The developers of TAVAT had also incorporated FreeLB in the TAVAT architecture.

FreeLB (or Free Large-Batch) refers to the diversifying of training data by adding adversarial perturbations that are under different norm constraints (Zhu et al. 2019). To do this and to boost invariance in embedding space, multiple PGD (projected gradient descent; see study by Madry et al. 2017) iterations are carried out in FreeLB. In the study, BERT, ALBERT and RoBERTa were used as backbones and tested on GLUE benchmark. For SST-2 (having a training set size of 67K) one of the FreeLB implementations, "FreeLB-RoB", showed 96.8% accuracy – as did XLNet-Large (Z. Yang et al. 2019), too. One should notice that FreeLB-RoB is actually an ensemble model that contains seven large RoBERTa models that are averaged out and put together. This is to have a better regularized outcome for modeling (Goodfellow, Bengio, and Courville 2016), and it is also known as bootstrap aggregating, or bagging (Breiman 1996). In the study, a single model using FreeLB, "FreeLB-BERT", was also implemented, reaching the accuracy of 93.6% for the SST-2 task. The result (of 97%) obtained with ALBERT (xxlarge v2), that was mentioned in the previous paragraph, remains as the best accuracy obtained for SST-2 when utilizing FreeLB as a regularization method.

Unsupervised data augmentation (UDA) was suggested by Xie et al. 2020, which is a SSL method that links two UDA methods, back translation and TF-IDF word replacement, to text classification. Back translation is elaborated in section 2.1. TF-IDF (term frequency - inverse document frequency) word replacement is yet another method invented to emphasize the keywords present in sequence and modify words that do not bring out useful information. This way additional data can be generated – by focusing on replacing trivial words, so that the meaning the sequence is supposed to convey, does not change. Following results for several datasets were obtained, when using finetuned BERT (a large BERT that was fine-tuned on in-domain unlabeled data) with UDA: error rate of 4.20 for IMDB (with 20 supervised examples), 2.05 for Yelp-2 (20 examples), 32.08 for Yelp-5 (2.5K samples), 3.50 for Amazon-2 (20 samples) and 37.12 for Amazon-5 (2.5K samples).

3.2 Pseudo labeling

The idea behind pseudo labeling is to first teach a neural network, or "teacher" model, with labeled data L in order to attribute labels to a set of unlabeled data U. After this pseudo labeling, these pseudo labeled data samples PL can be in turn used along with the labeled data L for constructing a "student" model (Lee 2013). This process of two phases, for constructing a teacher model and a student model, is visualized in figure 8. Pseudo labels were inspired by entropy regularization, that was described in the paper of Grandvalet, Bengio, et al. 2005.

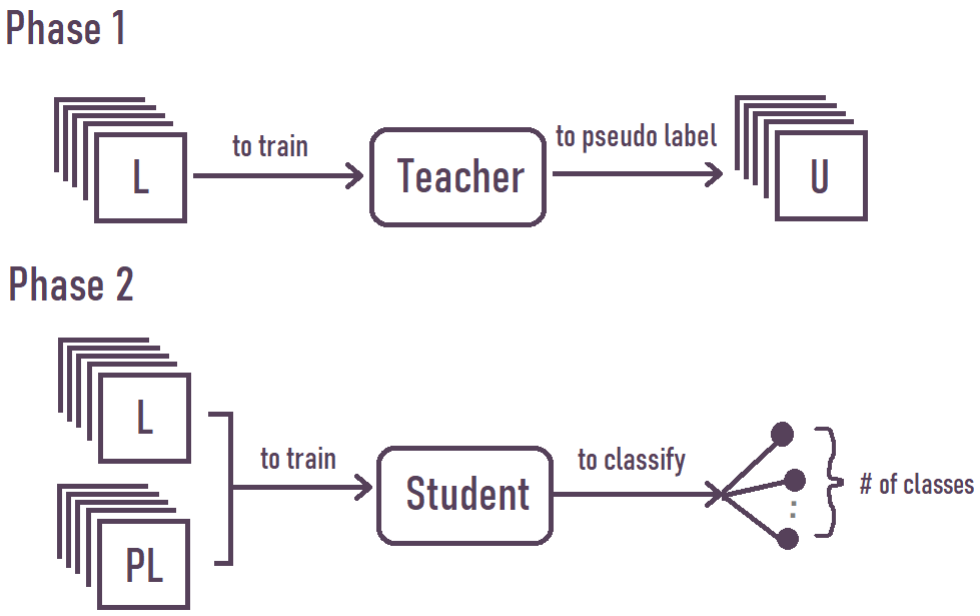

Figure 8. Idea behind pseudo labeling described in two phases (following the idea presented in Lee 2013). Here L is labeled data, U is unlabeled data, and PL is pseudo labeled data.

Pseudo labeling can be defined through the following formulaic expressions (from Arazo et al. 2020). Let us define a SSL model $h_\theta(\mathbf{x})$ and training dataset $\mathscr{D}$ of size n, that is divided into labeled dataset $\mathscr{D}_l = \{(x_i, y_i)\}_{i=1}^{n_l}$ and unlabeled dataset $\mathscr{D}_u = \{x_i\}_{i=1}^{n_u}$, where $n = n_l + n_u$ and $y_i \in \{0,1\}^k$ is one-hot encoding label for k classes that corresponds to x_i. We then assume that for n_u unlabeled data samples there is a pseudo label $\tilde{y}$ available. The training of the SSL model can then be re-defined as using $\tilde{\mathscr{D}} = \{(x_i, \tilde{y}_i)\}_{i=1}^{n}$, where $\tilde{y} = y$ for the n_l labeled data samples.

There exists several studies, that involve the use of low-entropy labeling or pseudo labeling.

MixText is a model that makes use of interpolation based regularizer, "Mixup", that is used for labeled and unlabeled data after pseudo labels, or "low-entropy labels" as is described in the paper, are formed for unlabeled data samples (Chen, Yang, and Yang 2020). In addition to labeled and unlabeled data augmented data is added via back translation (from English to German and back to English), that allows the paraphrasing for unlabeled data, after which consistency regularization is applied (which entails the use of perturbations, similarly to VAT). The model is constructed by using BERT uncased model. With 10, 200, 2500 labeled data samples, accuracies obtained for IMDB (2 classes) are 78.7%, 89.4%, 91.3%, for DBpedia (14 classes) 98.5%, 98.9%, 99.2%, and for Yahoo (4 classes) 200 and 2500 labeled data samples 71.3% and 74.1%, in the respective order.

Most of the same authors from the previously introduced study made the same type of use of low-entropy labeling in another study, but without applying interpolation for the text data (Chen, Wu, and Yang 2020). Similar to the previous study, augmented data was obtained by back translation and consistency regularization was also applied. The model, SMDA (from Semi-supervised Models via Data Augmentation), consists of a transformer based XLNet backbone (Z. Yang et al. 2019). In the study of Chen, Wu, and Yang 2020, used data included comments extracted from Reddit posts, and data was divided into subsets as follows: labeled data train set of size 8000, development set of 2000, test set of 2860 and unlabeled train set of 420k samples. Albeit dividing the labeled data into six groups (emotional disclosure, information disclosure, support, general support, information support and emotional support), it is not regarded as a multilabel classification task but rather it is divided into six separate binary classification tasks to assess whether a certain data sample belongs to a certain group or not. When compared to BERT baseline and XLNet baseline models, most accurate results were generally acquired with SMDA. For classification tasks for emotional disclosure, information disclosure, support, general support, infomartion support and emotional support, with SMDA the following accuracy/F_1 scores were obtained: 75.2/68.5, 74.3/71.0, 83.5/77.7, 91.7/63.7 (for XLNet base 92.7/65.0), 89.9/70.5 and 93.6/76.2, respectively.

A study focusing on pseudo labeling and MLM, or "LM pretraining" as it was referred in the study, for in-domain data had made primary use of BERT based modeling and had used small RoBERTa as the backbone of all implemented models (Sun et al. 2020). For the data, IMDB

was used with an additional IMDB data crawl of 3.4 million movie reviews. Pseudo labeling was done via training a teacher model in a supervised way with labeled movie reviews, then the teacher model was used to give labels to the unlabeled reviews of the data crawl which were in return used along with the labeled data to train a student model. The effect of differently sized labeled dataset was under scrutiny, and the study showed that the most promising results were extracted from two different settings: only including pretraining or including both pretraining and finetuning when constructing a model. The first setting proved to be successful with the most amount of labeled data (labeled/unlabeled division implicated in parentheses): accuracies (in %) being 93.79 (5K/1M) and 95.80 (25K/1M), while the second setting was especially useful when less labeled data was used: 55.47 (10/100K), 58.77 (20/100K), 85.86 (50/100K), 87.27 (100/100K) and 91.32 (1K/1M). Thus out of all the configurations, the best accuracy was obtained with 25K labeled and 1M unlabeled data. Iterative pseudo labeling was also applied, not showing as much of an effect when using more labeled data (1K, 5K or 25K) when compared to the use of less labeled data (10, 20, 50 or 100). In this case, iterative pseudo labeling refers to the way of the student model being used for re-labeling all of the unlabeled data. For the new student model formed in the next iteration, only a subset of these pseudo labeled data samples are picked – samples with higher confidence ("top-K instances"). In another study (Q. Xu et al. 2020), iterative pseudo labeling was conducted by drawing a subset of unlabeled data at each iteration, so the pseudo labeling was not directed at data samples that had previously been pseudo labeled.

The downside to pseudo labeling is the possible incorrect classifying of new data samples and its effect on model performance when training a new model with this pseudo labeled data. This can be referred as confirmation bias (Arazo et al. 2020). In a study by Kim and Kim 2020, confirmation bias was counterbalanced via adding negative feedback (antonym examples as additional data) to their framework. They implemented a domain classification task, but instead of giving only pseudo labels, self-distillation was used for bringing out confidence scores for each class and to see how probable it was that a data sample belonged to a certain class. These type of confidence scores can be used to determine how confident pseudo labeled data samples should be included and considered acceptable before the model performance starts to decline due to confirmation bias.

Another way to diminish confirmation bias is to train teacher and student models concurrently. The main idea behind meta pseudo labeling is similar to pseudo labeling, with the slight difference of teacher model having a feedback system that is backed up by the student model (Pham et al. 2021). After teaching a student model with a batch of pseudo labeled data (pseudo labels that the teacher model generated), the loss of student model is used for the teacher model to make it adjust its gradients for the next batch and training iteration.

In a study conducted by Hatefi et al. 2021, two models, Cformer and Distill-Cformer, were implemented that combined the use of BERT or DistilBERT and meta pseudo labels. For 10, 200 and 2500 examples, the following results were obtained (in accuracy, %): 88.7 (Cformer), 90.0 (Distill-Cformer) and 91.9 (Distill-Cformer) for AG News data, and 66.8, 72.0 and 74.5 (all Cformer) for Yahoo! data. Compared to UDA (by Xie et al. 2020, mentioned in section 3.1) and MixText (by Chen, Yang, and Yang 2020, mentioned above), Cformer surpassed them in all, and Distill-Cformer in most, cases – except in one, where Yahoo! data was used with only ten examples, MixText overpassed other models by a small margin (resulting to accuracy of 67.6%).

3.3 Generative models

Out of the existing generative models, that include a data producing structure, generative adversarial networks (GANs) have paved the way for generative work in ML. Substantial research on GANs have been made for computer vision tasks, and in the field a growing number of studies also focus on SSL (see the review by Sajun and Zualkernan 2022). One study conducted by Vlachostergiou et al. 2018 utilized the fact that unlabeled data can be processed with GANs and used them with bag-of-words in a unsupervised way for representation learning. In the NLP field, a number of GAN variations have been invented for text generation (Rosa and Papa 2021). However, there seems to be less research currently for GANs for semi-supervised text classification (or at least for sentiment analysis (Habimana et al. 2020)), especially when utilizing transformers modeling.

GAN-BERT (Croce, Castellucci, and Basili 2020) is a GAN based model, that was built with BERT model integrated into it. The real data samples, both labeled data samples L

and unlabeled data samples U, are fed into BERT that produces vector representations for them. These representations along with fake data samples F, that are generated from noise by the generator G, are then fed into the discriminator D that classifies them (see figure 9). The discriminator gives a class to a data sample out of a number of classes assigned to it, one additional class being "fake sample", that indicates if the data sample at hand is a fake data sample instead of a real one. GAN-BERT was trained with 20 News group (20 classes; 11314 samples in training dataset and 7531 in testing data), UIUC (6/50 classes; 5400 training samples) and SST-5 (5 classes; 11855 samples in total) for topic, question and sentiment classification tasks (in the corresponding order). With annotated (labeled) data of 1, 2, 5, 10, 20, 30, 40, 50% of the samples, the following results were obtained: for 20N data F_1-scores 45, 60, 70, 75, 78, 81, 83, 85, and for SST-5 accuracies 30, 37, 41, 46, 47, 49, 50, 51 were obtained. For UIUC data two settings were prepared: "coarse grained" with data divided into 6 classes and "fine grained" with data in 50 classes. These settings were the base of accuracy results of 63, 80, 90, 93, 94, 94.5, 95, 95.5 for coarse grained version and 27, 48, 66, 70, 76, 74, 77, 79 for fine grained. The results are approximate due to being checked from a picture (tables of results were not provided in the study).

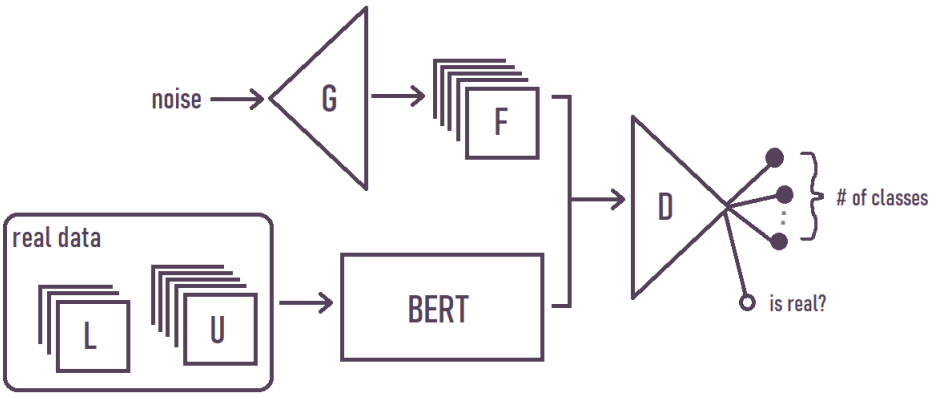

Figure 9. GAN-BERT architecture, where G is generator, F is fake data samples, D is discriminator, L is labeled data samples, and U is unlabeled data samples (figure adapted from Croce, Castellucci, and Basili 2020).

In addition to GANs, a transformer based model GPT has been invented to aid in several text generation tasks, e.g. in code generation. Few-shot learning was used for training GPT-3 (Generative Pretrained Transformer 3), a language model of 175 billion parameters, and

tested for a natural language inference (NLI) task among others (Brown et al. 2020). NLI can be constructed as a classification task of two or three classes, while ANLI (adversarial NLI) is an adversarially mined version of the same task that is performed in three rounds. For few-shot learning using the 175B model, 40% accuracy was obtained for the 3rd round (compared to 36% or lower accuracy scores obtained with smaller, 13B parameters or fewer, models).

There exists previous versions of GPT, most notably GPT-2 (Radford et al. 2019), that has also gained substantial research in the NLP field. Below there is two examples about how text generation with GPT-2 was harnessed for classification tasks.

In the study conducted by Puri and Catanzaro 2019, the main objective was to built a model for task adaptation, that entails the making of a model usable for several NLP tasks without using multiple heads (which is usually the convention). Puri and Catanzaro 2019 used text generation for creating an answer (out of a handful of choices) to task descriptor questions that were chained together with the text sample. These chains of task descriptors and text were input to the GPT-2 model, and the answer was obtained as the output. For this fine-tuning, OpenWebText dataset was used. When 355M model was used, the following results (in accuracy) for several evaluation datasets were obtained: for AGNews 68.3%, DBpedia 52.5% and Yahoo 52.2% when pretrained using 1/4 of data, and for SST-2 62.5%, Amazon-2 80.2% and Yelp-2 74.7% when pretrained using all of the data.

In another study (Edwards et al. 2021) the performance of models amped with text generation based data augmentation was compared to word and sentence replacement methods, where words are replaced, or sentences switched by performing back translation, for instance. The best performing model for all 20 Newsgroup (6 classes and 20 subclasses), Toxic comments (2 and 5), and Safeguarding (5 and 34) datasets was proven to be text generation based data augmentation model that was finetuned per label. This refers to the few-shot setting where the amount of data samples was 5 and 10 at the beginning, and then artificial data samples were added later on. For the text generation (GPT-2) model, the outcomes using four different seed selection strategies were also compared: random, maximum nouns-guided, subclass-guided, and expert-guided seed selection. In most of the cases random seed selection provided sufficient results, especially when using a modest number of seeds. FastText

classifier (merged together with FastText word embeddings) was used as the classifier along with the text generation model. In the end, the classification results of the best performing model (text generation per label) were shown with a t-test ($p_{value} < 0.05$) to be statistically significant over classification with no augmentation.

3.4 Other semi-supervised methods

There exists yet many other research methods, that are used for SSL. One of them is low-shot learning, that combines the methods that are referred to as zero-shot learning or few-shot learning methods. Low-shot learning could loosely be included or at least linked to SSL methodology. Due to their property of having either only a handful of annotations (few-shot learning) or none at all (zero-shot learning), they are promising alternatives to be considered for all types of tasks. For low-shot learning in NLP (and text classification) see the review by Xia et al. 2020, for example.

In addition to SSL methods used in transformer based modeling, there exists many studies that use traditional DNNs with SSL methods for text classification. We left them out from earlier sections due to them not being the most relevant to our research. For unsupervised pretraining methods used in conjunction with traditional DNNs, like LSTMs, see the studies by Dai and Le 2015; W. Xu et al. 2017; Miyato, Dai, and Goodfellow 2016; Sachan, Zaheer, and Salakhutdinov 2019; as well as Li and Ye 2018. In one study (Y. Zhang et al. 2017) unsupervised training was conducted simultaneously with supervised training. LSTMs have also been used for pseudo labeling (see the study by Li, Ko, and Choi 2019, for instance). Furthermore, generative models for text classification have utilized e.g. LSTMs (Li et al. 2018) and RNNs (Stanton and Irissappane 2019) in the model architecture.

4 Data

The data, that were used in this thesis, were collected in 2019 and 2021 from surveys that were part of two studies conducted by the Department of Social Sciences and Philosophy of University of Jyväskylä. The studies (see the papers Karhinen et al. 2019 and Karhinen et al. 2021) were carried out by Centre of Excellence in Research on Ageing and Care[1], or CoE AgeCare in short, which is a flagship research programme funded by the Academy of Finland for 2018–2025. The goal of these studies was to get information about the employees working in eldercare, the working environment and conditions the employees face, technology used in care work (such as devices and services), and the effects of digitalization in the workfield.

The questionnaire used in the studies consists of 62 questions, out of which two are open-ended. These two open-ended interview questions are:

1. What kind of emotions related to the use of technology have been present in your work during the last week? ("Millaisia tunteita teknologian käyttämiseen on liittynyt työssä viimeisen viikon aikana?")
2. What do you think about the following claim: "Technology improves the quality of eldercare work and decreases the pressure of employees." ("Mitä ajattelet väitteestä: Teknologia parantaa vanhustyön laatua ja vähentää työntekijöiden kuormitusta.")

The other sixty interview questions are e.g. about age, gender, marital status, job title, work union, educational level, type of employer, place of working in Finland (e.g. capital area), work experience (in years), working hours, and structured questions (with a few options for answering) about how the workers experience stress or pressure, and also about the use of technology at work – how much it is used, what type of devices, how much time the using takes, and estimation about employees' own digital skills.

All in all, 6903 filled out the questionnaire for the data collected in 2019. For the two open-ended questions 3652 answers were obtained (for both of the questions, so no empty values are present). 65 of them are in Swedish, the rest in Finnish. There are 367 duplicates for the

1. https://www.jyu.fi/hytk/fi/laitokset/yfi/en/research/projects/agecare

answers of the first question, and 494 duplicates for the answers of the second question. All in all, 1679 filled out the questionnaire for the data collected in 2021. For the first question 1000 answers were obtained, and for the second 1099 answers. All of them were in Finnish. There are 57 duplicates for the answers of the first question, and 93 duplicates for the answers of the second question. When the 2019 and 2021 datasets are combined without removing the aforementioned duplicates, there are 485 duplicates for the answers of the first question, and 649 duplicates for the answers of the second question.

When processing the data, we transform the data samples into vectoral form, "tokens". In the combined dataset, the minimum length of one data sample is 3 tokens, median length is 13 tokens, mean length is 19.5 tokens, and maximum length is 429 tokens. When using maximum sequence length of 100, there is no splitting (but all tokens are included) for 98.98% of the data samples. When using 272 for maximum sequence length, only one data sample (of the length 429) is cut to fit the size limit of 272.

For this thesis we use the answers obtained for the first open-ended question as the training, validation and testing data to conduct sentiment analysis. Sentiment analysis refers to a classification problem that focuses on sentimental content of the data samples, and usually relates to classifying data samples into three classes: negative, positive and neutral. In our data every answer, or data sample, consists of a sentence or several in natural language. These data samples are divided into these three classes later on, during the preprocessing phase described in section 5.1. The amount of data samples belonging to each class can be seen in table 1).

Table 1. The amount of data samples belonging to each class.

sentiment	# of samples	% of samples
neutral	1222	30.8
negative	2224	56.0
positive	525	13.2
all	3971	100

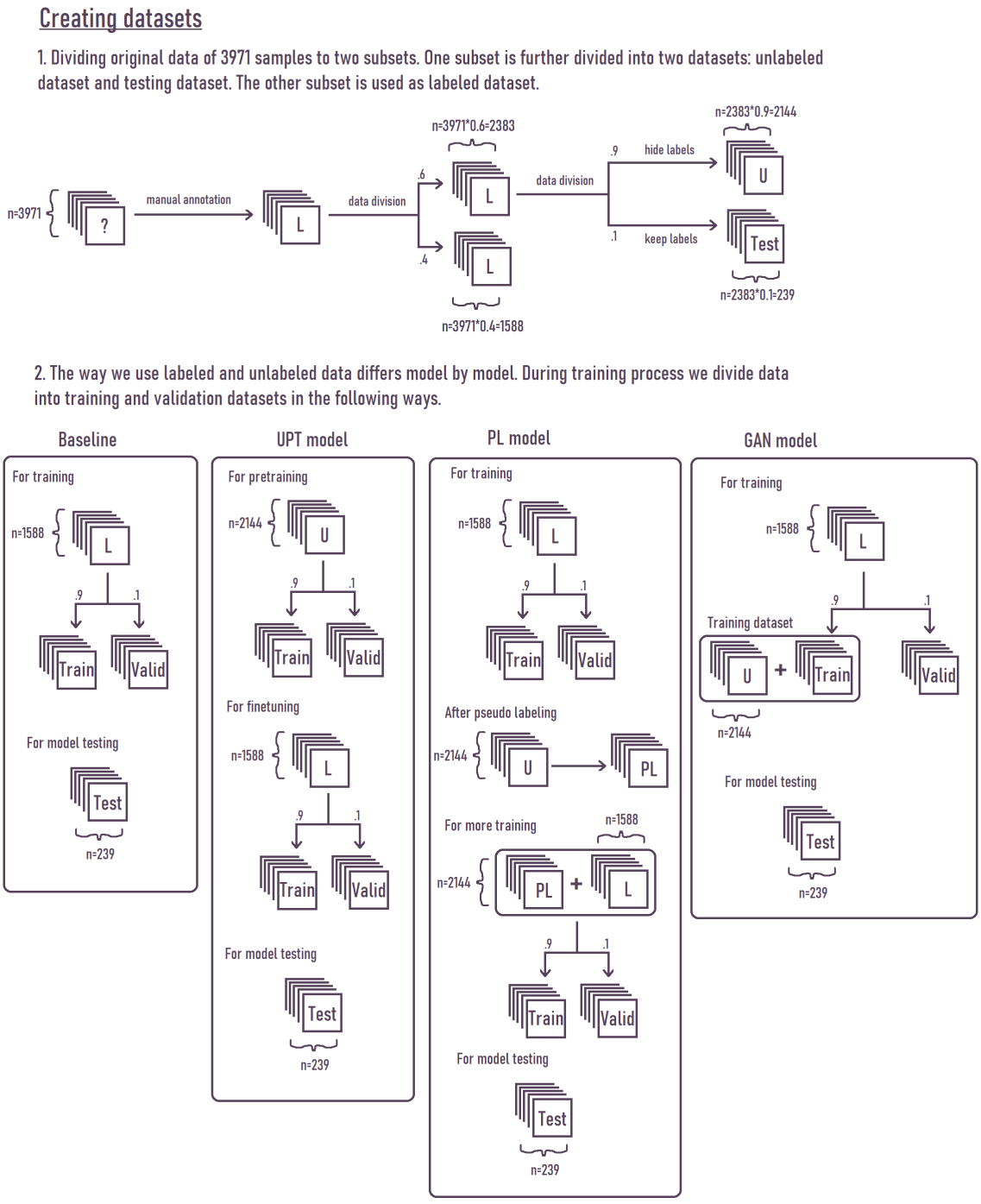

Figure 10. Creating datasets involved two steps: 1) dividing data into labeled, unlabeled and testing datasets, and 2) dividing labeled and unlabeled datasets further into training and validation datasets. Here UPT model is unsupervised pretraining model (presented in section 5.2.2), PL model is pseudo labeling model (5.2.3) and GAN model is generative adversarial network model (5.2.4).

5 Methods

Due to our data being on the small size we discard the use of non-transformer based DNNs, that are traditionally used in text classification and need lots of data, and compare SSL methods that are only implemented on FinBERT (Virtanen et al. 2019) backbone, which is based on transformer modeling.

First, we go through data preprocessing steps in section 5.1. Training DNNs with SSL methods suggests that not all data is annotated. The original data did not include any annotations, so we also introduce the annotative measures used to determine classes for data samples. Then we also go through steps that involve making of different datasets (can be seen in figure 10). After data related processing, we describe in section 5.2 the architectures of models that utilize SSL methods that we implement to perform text classification (or sentiment analysis) on our data. Finally, in section 5.3 we elaborate on evaluation that is used for the SSL models we build.

5.1 Data preprocessing

First for data privacy reasons, the most essential columns (column of identifiers and a column for answers to the first open-ended question) from the original data are chosen and the rest of the data is discarded during the model implementation process. At first, we concatenate the two datasets of 2019 and 2021. Then we check whether there are empty or duplicate values and remove them. We then remove non-text values (data samples with only emojis, punctuation and digits) to perform language detection with a library (langdetect[1], version 1.0.9) to remove Swedish answers from the dataset, as the sentiment analysis is intended to be monolingual. After detecting Swedish data samples, we remove them from the concatenated dataset along with data samples that have only punctuation in them. Finally, we end up with 3971 data samples.

After these preprocessing steps, manual annotation is done by hand by a native Finnish person, that has no professional background in Finnish language nor literature. Manual an-

1. https://pypi.org/project/langdetect/

notation means that the whole of data was read through data sample by data sample, during which every data sample was classified to one of the following classes: positive ("0"), negative ("1"), or neutral ("2"). Only after annotation the splitting of data into unlabeled and labeled datasets is carried out to have an even distribution of classes in each of the dataset. This even split is aimed to be achieved with stratified K-fold.

We use stratified K-fold to have a 60%/40% split to the data (unlabeled/labeled), having an even distribution of all three classes for both of the datasets. For stratified K-fold we use the parameter Shuffle=True to shuffle the data randomly before it is being split[2]. This split is achieved by having five folds, out of which we are randomly choosing two folds for the labeled dataset (2/5=0.4) and other three folds for the unlabeled dataset (3/5=0.6). We split the unlabeled data even further (with stratified K-fold to ten folds) to construct a testing dataset that can be used for evaluating all of the models after training. This is how we now have three different datasets drawn from the same in-domain data: unlabeled dataset (54% of data or 2144 samples), labeled dataset (40% of data or 1588 samples) and testing dataset (6% of data or 239 samples). This process of dividing data is visualized (in step one) in figure 10.

Before feeding the data into the model we use pretrained FinBERT cased tokenizer (Virtanen et al. 2019) for conducting a series of changes for every data sample (see appendix A for configuration information). These changes include tokenization, assigning maximum sequence length, adding [CLS] and [SEP] tokens to indicate the beginning and ending of a data sample (parameter add_special_tokens is set to True), padding to maximum length (so that every data sample is of the same size) and returning an attention mask that tells the model what are the tokens that should be taken into account (indicated by "1") and which are paddings (indicated by "0") [3].

Finally, before we load the data into a data loader[4], that is used for iterating through the data in batches during training, we assign a specific type of sampler[5] to the data loader according to which the data is sampled during training process. In the training process we use training

2. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.StratifiedKFold.html
3. https://huggingface.co/docs/transformers/v4.18.0/en/glossary#attention-mask
4. https://pytorch.org/docs/stable/data.html#torch.utils.data.DataLoader
5. https://pytorch.org/docs/stable/data.html#torch.utils.data.Sampler

and validation datasets, for both of which we create their own data loaders. The training data is randomly sampled, so the order in which samples are iterated cannot be predicted. This is to increase randomness in the model training process. The validation dataset is sequentially sampled, which means that the data samples are iterated through without changing the order. After training process the model is tested with the testing dataset, so when a data loader is created for it, we assign a sequential sampler for it.

5.2 Models

In this thesis, in order to utilize unlabeled and labeled data three different SSL methods were implemented. Along with the baseline (presented in section 5.2.1), we make three models: a model that utilizes unsupervised pretraining (in section 5.2.2), a model that utilizes pseudo labeling (in section 5.2.3), and a model that utilizes generative adversarial network (in section 5.2.4). This loosely follows the taxonomy set in Van Engelen and Hoos 2020.

For all of the models, cased FinBERT base (Virtanen et al. 2019) is used as the backbone (see appendix A for more information about the configuration of FinBERT). There are a few other BERT models that have been trained to work with Finnish data, that include multilingual BERT (Devlin et al. 2018), XLM-RoBERTa (Conneau et al. 2019), and FinEst BERT (Ulčar and Robnik-Šikonja 2020). While for FinBERT the size of training corpus was 13.5B tokens, the training corpus size for Finnish was 6730M tokens for XLM-RoBERTa (Conneau et al. 2019), and 925M tokens for FinEst BERT (Ulčar and Robnik-Šikonja 2020). The training corpus size of multilingual BERT is not reported for different languages, but the training corpora are mentioned to be based off of Wikipedia articles[6]. There are 530 668 Wikipedia articles in Finnish at the moment [7]. For reference, the authors of FinBERT report the training corpora to be of 98M documents in total in their study. FinBERT is chosen as it outperforms multilingual BERT (Virtanen et al. 2019) and shows the strongest training corpus size out of all the BERT models pretrained for Finnish.

During the implementation of the models, one random seed is also picked to be used con-

6. https://github.com/google-research/bert/blob/master/multilingual.md
7. https://meta.wikimedia.org/wiki/List_of_Wikipedias

sistently to make comparisons between different model variations. The classification task is directed at three classes, so the final layer in each model has output of three (0 for indicating positive, 1 for negative and 2 for neutral class) – except for GAN model, for which we have different amounts of outputs (see section 5.2.4). For all model training, we use the following libraries and versions: Python 3.7, Pytorch 0.8.1, Transformers 4.16.2, and CUDA 11.6. All of the work is done with one GPU (NVIDIA Tesla P100 with 16GB RAM).

5.2.1 Baseline

To compare the performance of models belonging to the SSL framework to a simple supervised model, we choose to train (or finetune) a model using only the labeled dataset and use it as the baseline. To the labeled dataset a split of 90%/10% is done to create training and validation datasets for finetuning. This process of dividing data is visualized (in step two) in figure 10.

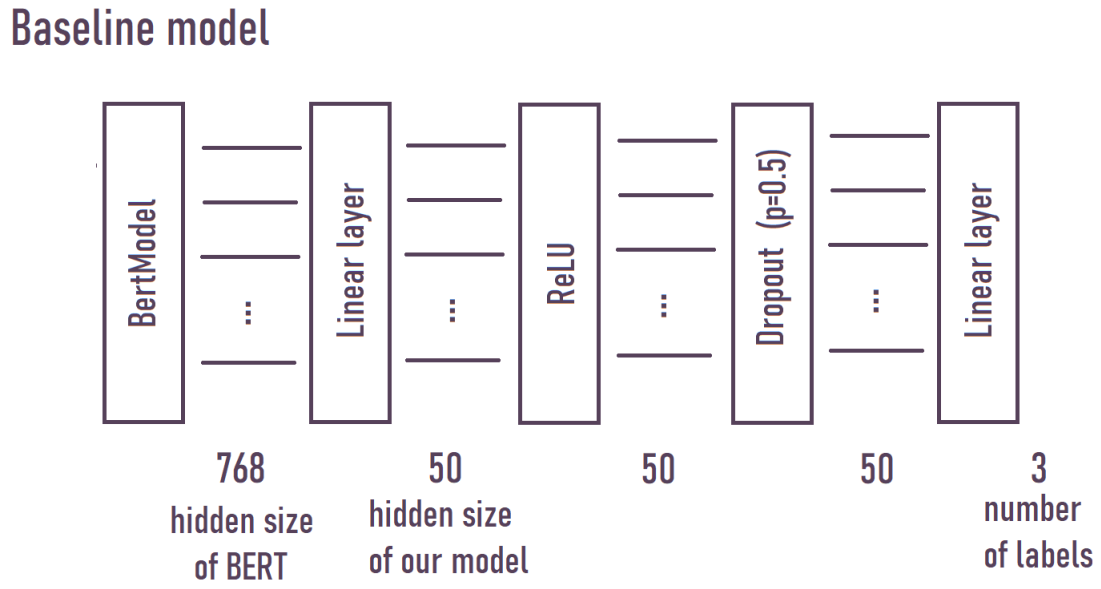

Figure 11. Baseline model architecture in a simplified form.

For the model hyperparameters we have learning rate of 1e-5, weight decay 0, epsilon 1e-8, cosine scheduler with warmup (number of warmup steps is the default of 0), and AdamW as optimizer (Loshchilov and Hutter 2017). Architecture of the model is the following (see the figure 11): linear layer (hidden size of BERT is 768, hidden size of our model 50), dropout

(with probability of 0.5), ReLU, and linear layer (hidden size of our model 50, output of 3). With the use of dropout layers, where units and their connections are randomly dropped in the middle of training (see figure 12), regularization can be adapted to the model architecture (Srivastava et al. 2014). We use BertModel as backbone and retrieve cased FinBERT from pretrained models ("TurkuNLP/bert-base-finnish-cased-v1"). Gradient clipping (to 1.0) is included to prevent exploding gradient problem, which refers to gradients getting too large in gradient based modeling (Goodfellow, Bengio, and Courville 2016). Maximum sequence length is 100 and batch size is 128. We use early stopping to stop the model from running if no apparent improvement happens for 10 epochs.

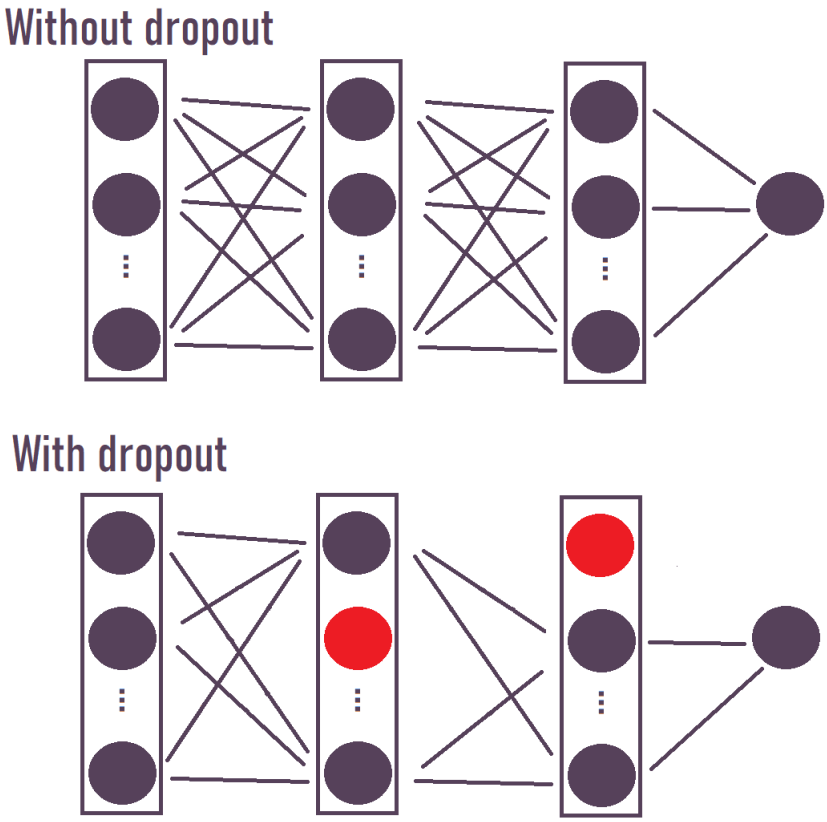

Figure 12. Neural network without and with dropouts, where red units represent the units that are dropped out (figure adapted from Srivastava et al. 2014).

5.2.2 Unsupervised pretraining

A model that makes use of unlabeled data via unsupervised pretraining in this study is a FinBERT that is pretrained with the unlabeled data and then finetuned with labeled data. We refer to this model as UPT model.

For pretraining, we use model for masked language modeling (BertForMaskedLM) and retrieve cased FinBERT from pretrained models ("TurkuNLP/bert-base-finnish-cased-v1") for it. Before running the model, we make a split of 90%/10% to the unlabeled data to create training and validation datasets. This process of dividing data is visualized (in step two) in figure 10. Then we run the model for 35 epochs with batch size 16 for both training and validation. We use optimizer AdamW (with learning rate 1e-6, epsilon 1e-8 and weight decay 0.01) and use cosine scheduler with warmups (number of warmups is at default, 0). Maximum sequence length is set to 275, and padding to maximum length is set to True. This does not cut down any data samples of the unlabeled data used for pretraining. MLM probability is set to 0.15.

To the labeled dataset a split of 90%/10% is done to create training and validation datasets for finetuning. This process of dividing data is visualized (in step two) in figure 10. For finetuning, the same hyperparameters and architecture are used as for the baseline.

5.2.3 Pseudo labeling

For training the pseudo labeling models, or PL models, the same model hyperparameters and architecture are used as for the baseline. When using the trained PL models for pseudo labeling, in all cases maximum sequence length is set to 200.

For pseudo labeling, we try two different versions (see figure 13) for training to see whether iterative pseudo labeling will make an effect on the results. In the first version, labeled dataset L is used to train PL model 1. PL model 1 is then used to pseudo label unlabeled dataset U_{All}, that includes all of the unlabeled data samples. Finally, PL model 2 is trained with labeled dataset L and unlabeled dataset U_{All}, that now has been pseudo labeled by PL model 1. We use PL model 2 when testing the first version of PL modeling.

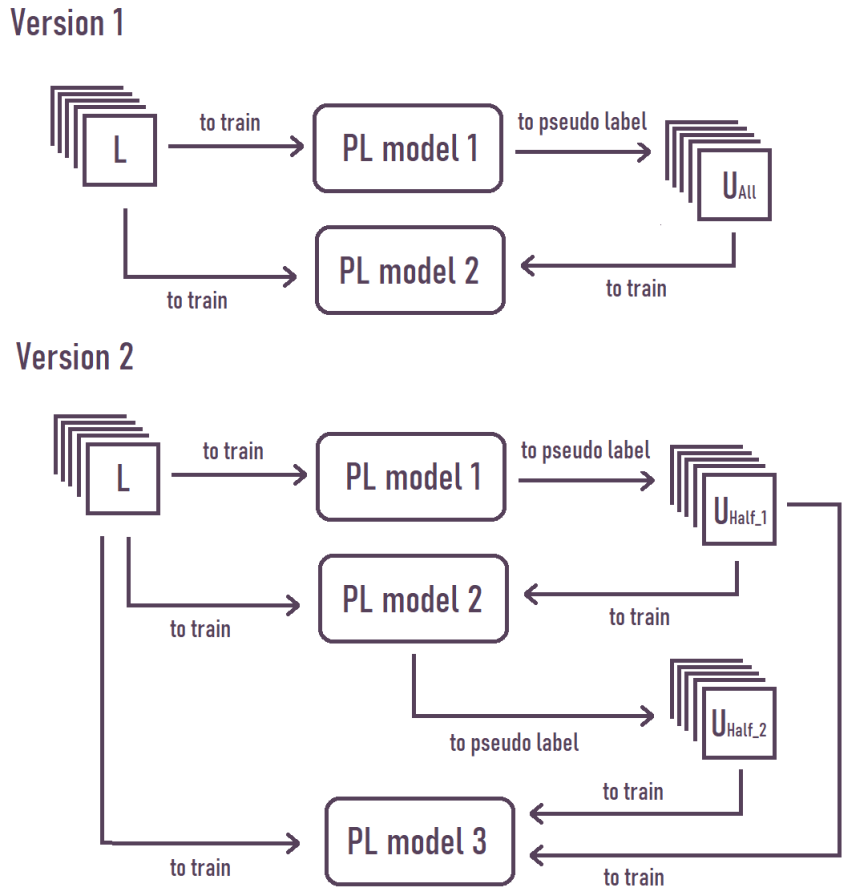

Figure 13. Two versions for pseudo labeling: training a model with pseudo labeling (version 1) and with iterative pseudo labeling (version 2), where L is labeled dataset, U_{All} is unlabeled dataset including all samples, U_{Half_1} is unlabeled dataset including first half of the samples, U_{Half_2} is unlabeled dataset including second half of the samples, PL model 1 is pseudo labeling model 1, PL model 2 is pseudo labeling model 2 and PL model 3 is pseudo labeling model 3.

Iterative pseudo labeling is attempted in the second version. In the second version, labeled dataset L is used to train PL model 1. PL model 1 is then used to pseudo label unlabeled dataset U_{Half_1}, that includes half of the unlabeled data samples. Next, PL model 2 is trained with labeled dataset L and unlabeled dataset U_{Half_1} (that now has pseudo labels). PL model 2 is then used to pseudo label unlabeled dataset U_{Half_2}, that includes the other half of the unlabeled data samples. Finally, model 3 is trained with labeled dataset L, and unlabeled

datasets U_{Half_1} and U_{Half_2}, that now have been pseudo labeled by PL model 1 and PL model 2, respectively. We use PL model 3 when testing the second version of PL modeling.

During the training process of each PL model, we thus use either only labeled dataset L, or concatenated dataset consisting of labeled data L and pseudo labeled data (based on unlabeled dataset U_{All}, U_{Half_1}, or both U_{Half_1} and U_{Half_2}). This concatenated dataset is split 90%/10% to training and validation datasets for every PL model before training. This process of dividing data is visualized (in step two) in figure 10.

5.2.4 Generative adversarial network

Generative adversarial network model, or GAN model, is implemented by using generator-discriminator structure with FinBERT as classifier. We follow the architecture used in the study by Croce, Castellucci, and Basili 2020, that is visualized in the figure 9. For constructing FinBERT, we use BertModel and retrieve cased FinBERT from pretrained models ("TurkuNLP/bert-base-finnish-cased-v1") for it. We train the model for 50 epochs.

In the GAN model the generator produces fake samples, that are input into the discriminator along with BERT outputs. BERT outputs are vector representations of real data samples, that consist of labeled and unlabeled data. The labeled dataset is split into 90%/10% to form training and validation datasets. Unlabeled data is concatenated with the training data, and a mask is set for the data samples in this concatenated dataset (True for indicating a labeled data sample, False for indicating an unlabeled data sample). This process of dividing data is visualized (in step two) in figure 10.

Model hyperparameters for both the generator and the discriminator include optimizer AdamW, constant schedulers with warmup of 0.1, learning rate 1e-6 and epsilon 1e-8. We use maximum sequence length of 100 and batch size 128. The noise size of generator is set to 100.

The discriminator has the following architecture: dropout layer (p=0.5), linear layer (input size 768, hidden size of 768), LeakyReLU (0.2), dropout (p=0.5), linear layer (input size 768, hidden size of 768), LeakyReLU (a=0.2), dropout (p=0.5), and final layer softmax, where number of labels is 5 – out of which four are for positive, negative, neutral and "UNK" for unlabeled data, and one for the probability of the sample being fake or real. The architecture

of the generator is the same as with discriminator – with the exception of the final layer output size being 768, instead of 5.

5.3 Evaluation

To estimate the performance abilities of the model, certain evaluation metrics can be used. For our classification task we use accuracy to determine the model performance and cross-entropy loss[8] for the loss function. Accuracy is the percentage of having all correctly predicted data samples out of all predictions (Minaee et al. 2021).

$$\text{Accuracy } = \frac{m}{n}, \tag{5.1}$$

where m is the amount of all correctly predicted data samples, and n is the amount of all data samples. Cross-entropy loss is used for classification tasks, and it can be defined as (Martinez and Stiefelhagen 2018):

$$H(p,q) = -\sum_{i=1}^{n} p_i \log(q_i), \tag{5.2}$$

when discrete distributions $p \in [0,1]^n$ and $q \in (0,1]^n$ are the size of n. Cross-entropy loss is calculated between these two distributions: p, that represents the target of classification, and q, which is the output of the softmax layer (see equation 2.4) – that is, the likelihood that is predicted per class (Martinez and Stiefelhagen 2018).

8. https://pytorch.org/docs/stable/generated/torch.nn.CrossEntropyLoss.html#crossentropyloss

6 Results

The performance of each model was evaluated with cross entropy loss and accuracy. These metrics were calculated to have the model performance evaluated twice: first, during model training with validation dataset, and second, after training with testing dataset. We used validation dataset to choose the best performing model variation for each SSL model, and testing dataset to test if the model generalizes well to unseen data. We use the model name abbreviations as follows: UPT model for unsupervised pretraining model, PL model for pseudo labeling model, and GAN model for generative adversarial network model.

The results for validation dataset can be seen from table 2. We can see from the table that the PL model seems to give the best performance out of all the models. Up next would be the baseline, and UPT model and GAN model after that. The results for testing dataset can be seen from table 3, that suggest that instead of the PL model the baseline seems to give the best performance out of all the models. The performance of the UPT model, along with the PL model and GAN model, is not far off from the performance of the baseline.

Table 2. All the best model results for validation dataset. Here UPT refers to unsupervised pretraining model, PL to pseudo labeling model and GAN to generative adversarial network model.

	baseline	UPT	PL	GAN
loss	0.29206	0.35041	0.18769	0.35918
acc	0.9170	0.9013	0.9438	0.8805

Table 3. All the best model results for testing dataset. Here UPT refers to unsupervised pretraining model, PL to pseudo labeling model and GAN to generative adversarial network model.

	baseline	UPT	PL	GAN
loss	0.36404	0.39304	0.40214	0.42164
acc	0.88889	0.86111	0.87037	0.86192

Additionally, there results for the PL model version 1 (PL without iterative process) and

version 2 (PL with iterative process) are indicated in table 4. The results suggest that the version 1 works better than the version 2 PL model, for both the validation and the testing dataset.

Table 4. PL model results for version 1 and 2 when evaluating with validation and testing datasets.

	Version 1 (valid)	Version 1 (test)	Version 2 (valid)	Version 2 (test)
loss	0.18769	0.40214	0.21003	0.44618
acc	0.9438	0.87037	0.9331	0.86111

7 Discussion

Usually for DL model training as much data as possible should be used. A few thousand data samples is generally a small dataset, so to prevent overfitting different regularization methods can provide to be useful. When looking at our results (in tables 2 and 3), the difference between the results for validation and testing datasets suggests that the models are most likely not generalizing well to new data. It is likely that at least some overfitting has happened during the SSL model training. In the final results one can see that the baseline shows better results than all the other models, when evaluated with testing dataset. The possible reasons why this might have happened are discussed below, SSL model by SSL model.

In the case of UPT model, the reason why this model shows weaker results than the baseline could be that it is more advantageous to use a large text dataset with diversity for pretraining and to use more in-domain data for finetuning to train the model for a specific task (Radford et al. 2018). When focusing on using only in-domain data for pretraining and finetuning, BERT based models may fail to adopt understanding of language in all of its complexities, which can possibly lead to weakened generalization and overfitting of the model (Jiang et al. 2019). Preservations towards using data that is not directly related to the domain of classification task (out-of-domain data) thus might be unwarranted. The results shown in the study by Sun et al. 2020 suggest that this could be bypassed by using very large amounts of in-domain data for pretraining. The initial attempts at finetuning for some UPT model variations, where pretraining was done additionally for 20 and 50 epochs, also indicates that with limited amount of data we come across a wall much sooner (in 50 epochs) when we add epochs to pretrain for longer. Moreover, for 20 epochs the model had not exceeded the limit of adopting new features from the data, and had higher loss and lower accuracy than does the best performing UPT model, that was pretrained with 35 epochs.

In the case of the first version of PL model, where iterative process was not applied, the model seemed to have confident results for the validation dataset (accuracy of 94.38% and loss of 0.18769), but looking at the testing dataset results (accuracy of 87.04% and loss of 0.40214) there seems to be a drop in model performance. This suggests that overfitting

might have occured, which could be due to confirmation bias. Pseudo labeling was not confined in the sense that no confidence values were calculated and applied for the model (like was done in the study of Sun et al. 2020). Another method for restricting confirmation bias could have been the use of meta pseudo labels (Pham et al. 2021), that could help with regularization of the model. In the case of second version of PL model, where iterative process was applied, the iterative training of multiple models with subsets of data seems to not bring an improvement on the PL modeling results. When looking at the testing dataset results, there is a slight increase in loss (of 0.044) and a small decrease in accuracy (0.9%) for the second version PL model when compared to the first version PL model results (see table 4).

In the case of GAN model, where GAN-BERT was trained, the model architecture is decidedly a different one when compared to the baseline and other implemented SSL models. In the paper where GAN was first introduced (by Goodfellow et al. 2014), the authors mention the possibility and the need for future research for using GANs for situations where very little data is annotated (for SSL settings). However, in the study conducted by Dai et al. 2017 the authors note that for a good discriminator a bad generator (meaning that the generator distribution should not be aligned with the distribution of true data) should be trained, and found out that complement generator is the type of generator that benefits the discriminator more. Complement generator is a generator that produced samples that are complement in feature space. We are likely to see this effect happening during our training – when the training goes on, the generator loss keeps increasing while the discriminator loss is decreasing. For improving the results for GAN, this type of adjustment of generator could be beneficial when implementing GAN based models. Another measure to better regularization could have been the use of consistency regularization (H. Zhang et al. 2019). The usefulness of the type of applied regularization method might depend on the model architecture. For example, gradient based regularization might not be that good of a fit for the discriminator of GAN, but consistency regularization might be better in terms of lighter computation, because it does not increase much overhead (H. Zhang et al. 2019).

Some other decisions made during the model implementation could also be examined. Regarding data, hand-made annotations could act as a pitfall when not done professionally.

One possible loophole for evading manual labor in this regard could have been using newly released FinBERT-finnsentiment, that is a FinBERT finetuned with 27 000 sentence dataset (Lindén, Jauhiainen, and Hardwick 2020) for sentiment analysis, for classifying unlabeled data samples. Stratified K-fold was used to divide the data into datasets with the same distribution of data samples pertaining to different classes (see section 5.1), but other data splitting techniques could have been tried out as well. The strength of stratified K-fold is that with additional parameter (Shuffle=True), before splitting the data it is shuffled, and thus, randomness is increased in the model making process. The original data includes samples from both 2019 and 2021. Possible temporal effects were also not taken into account – for example, more recent data from 2021 could have COVID-19 terminology used that is not visible in 2019 data, which could be considered as noise.

When thinking about model training decisions to improve on, hyperparameter optimization-wise there is much left to do. For example, trying out smaller learning rates and more training epochs would lead to slower convergence, but it could possibly be better for optimization of the model. When running all the models, one iteration took at most a minute and a half (with bigger batch sizes), totaling up to 35 minutes at most before the model performance stopped to improve. More GPUs could also have been used to have bigger batch sizes without compromising the use of larger maximum sequence lengths. Initial attempts at using smaller maximum sequence lengths (e.g. 64), especially to enable the use of bigger batch sizes in one GPU, and then switching to larger maximum sequence lengths (e.g. 100, or 272) showed that accessibility to more data (more tokens) seem to lead to better results. Another thing to note is that using a specific seed might effect the results even up to a few digits (of accuracy, for example), so in some cases the results may not seem to be possible to be reproduced exactly, based off of the seed behaviour alone. Different seeds were not tested during the implementation phase, but this type of effect on modeling can be seen in the community sites (e.g. GitHub). We also made initial research on the effect of modest use of dropout in the case of baseline, that seems to be potential for basic regularization of the model. However, adding dropouts might not be as efficient when training models for smaller datasets (Goodfellow, Bengio, and Courville 2016), so more testing of such regularization methods should be done to draw conclusive observations on the matter.

If further research were to be conducted, more data, hyperparameter optimization, different network structures, regularization measures and the assessment of seed behaviour could be focused on.

8 Conclusions

The research question presented in introduction (see chapter 1) had us looking for SSL methods that could be useful for a case where little annotated data is available for model training. The aim of this thesis was to research SSL methods and determine how they work for our dataset when building models for text classification. These methods are more thoroughly described in the literature review of this thesis (see chapter 3). We tested a few of these methods, and implemented BERT based models in the constructive part of this thesis (see implemented methods in chapter 5) for data that was concerning on the sentiments elicited in eldercare workers regarding the increasing use of technology in their work (in chapter 4). Depending on the SSL method in question, there are things that should be taken into account in order to train a model without it overfitting. These distinctive properties of each of the methods, that call for attention when looking out for probable problems in model training, are presented in the results (chapter 6) and discussion (chapter 7).

For more thorough optimization of the model more confined settings (i.e. regularization) should be set up for the model during its training process. Future research is called for to better discern the applicability of different SSL methods without overfitting interfering with the results.

Bibliography

Arazo, Eric, Diego Ortego, Paul Albert, Noel E O'Connor, and Kevin McGuinness. 2020. "Pseudo-labeling and confirmation bias in deep semi-supervised learning". In *2020 International Joint Conference on Neural Networks (IJCNN),* 1–8. IEEE. https://arxiv.org/pdf/1908.02983.pdf.

Ba, Jimmy Lei, Jamie Ryan Kiros, and Geoffrey E Hinton. 2016. "Layer normalization". *arXiv preprint arXiv:1607.06450,* https://arxiv.org/pdf/1607.06450v1.pdf.

Bengio, Yoshua. 2012. "Practical recommendations for gradient-based training of deep architectures". In *Neural networks: Tricks of the trade,* 437–478. Springer. https://arxiv.org/pdf/1206.5533.pdf.

Bojanowski, Piotr, Edouard Grave, Armand Joulin, and Tomas Mikolov. 2017. "Enriching word vectors with subword information". *Transactions of the association for computational linguistics* 5:135–146. https://arxiv.org/pdf/1607.04606v2.pdf.

Breiman, Leo. 1996. "Bagging predictors". *Machine learning* 24 (2): 123–140. https://link.springer.com/content/pdf/10.1007/BF00058655.pdf.

Brown, Tom, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. "Language models are few-shot learners". *Advances in neural information processing systems* 33:1877–1901. https://arxiv.org/abs/2005.14165.

Chen, Jiaao, Yuwei Wu, and Diyi Yang. 2020. "Semi-Supervised Models via Data Augmentation for Classifying Interactive Affective Responses". *arXiv preprint arXiv:2004.10972,* https://arxiv.org/pdf/2004.10972v1.pdf.

Chen, Jiaao, Zichao Yang, and Diyi Yang. 2020. "Mixtext: Linguistically-informed interpolation of hidden space for semi-supervised text classification". *arXiv preprint arXiv:2004.12239,* https://arxiv.org/pdf/2004.12239v1.pdf.

Clark, Kevin, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. "Electra: Pre-training text encoders as discriminators rather than generators". *arXiv preprint arXiv:2003.10555,* https://arxiv.org/pdf/2003.10555.pdf.

Conneau, Alexis, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. 2019. "Unsupervised cross-lingual representation learning at scale". *arXiv preprint arXiv:1911.02116,* https://arxiv.org/pdf/1911.02116.pdf.

Croce, Danilo, Giuseppe Castellucci, and Roberto Basili. 2020. "Gan-bert: Generative adversarial learning for robust text classification with a bunch of labeled examples". In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics,* 2114–2119. https://www.aclweb.org/anthology/2020.acl-main.191.pdf.

Dai, Andrew M, and Quoc V Le. 2015. "Semi-supervised sequence learning". *arXiv preprint arXiv:1511.01432,* https://arxiv.org/pdf/1511.01432.pdf.

Dai, Zihang, Zhilin Yang, Fan Yang, William W Cohen, and Russ R Salakhutdinov. 2017. "Good semi-supervised learning that requires a bad gan". *Advances in neural information processing systems* 30. https://arxiv.org/pdf/1705.09783.pdf.

Devlin, Jacob, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. "Bert: Pre-training of deep bidirectional transformers for language understanding". *arXiv preprint arXiv:1810.04805* https://arxiv.org/pdf/1810.04805.pdf.

Edwards, Aleksandra, Asahi Ushio, Jose Camacho-Collados, Hélène de Ribaupierre, and Alun Preece. 2021. "Guiding Generative Language Models for Data Augmentation in Few-Shot Text Classification". *arXiv preprint arXiv:2111.09064,* https://arxiv.org/pdf/2111.09064.pdf.

Goodfellow, Ian, Yoshua Bengio, and Aaron Courville. 2016. *Deep learning*. MIT press. https://www.deeplearningbook.org/.

Goodfellow, Ian, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2014. “Generative adversarial nets”. *Advances in neural information processing systems* 27. https://proceedings.neurips.cc/paper/2014/file/5ca3e9b122f61f8f06494c97b1afccf3-Paper.pdf.

Grandvalet, Yves, Yoshua Bengio, et al. 2005. “Semi-supervised learning by entropy minimization.” *CAP* 367:281–296. https://papers.nips.cc/paper/2004/file/96f2b50b5d3613adf9c27049b2a888c7-Paper.pdf.

Gururangan, Suchin, Tam Dang, Dallas Card, and Noah A. Smith. 2019. “Variational Pre-training for Semi-supervised Text Classification”. In *Proceedings of ACL.* https://arxiv.org/pdf/1906.02242v1.pdf.

Habimana, Olivier, Yuhua Li, Ruixuan Li, Xiwu Gu, and Ge Yu. 2020. “Sentiment analysis using deep learning approaches: an overview”. *Science China Information Sciences* 63 (1): 1–36. https://link.springer.com/article/10.1007/s11432-018-9941-6.

Hatefi, Arezoo, Xuan-Son Vu, Monowar Bhuyan, and Frank Drewes. 2021. “Cformer: Semi-Supervised Text Clustering Based on Pseudo Labeling”. In *Proceedings of the 30th ACM International Conference on Information & Knowledge Management,* 3078–3082. https://people.cs.umu.se/sonvx/files/Cformer-Preprint.pdf.

Haykin, Simon. 2010. *Neural networks and learning machines, 3/E.* Pearson Education India. https://dai.fmph.uniba.sk/courses/NN/haykin.neural-networks.3ed.2009.pdf.

He, Pengcheng, Xiaodong Liu, Jianfeng Gao, and Weizhu Chen. 2020. “Deberta: Decoding-enhanced bert with disentangled attention”. *arXiv preprint arXiv:2006.03654,* https://arxiv.org/pdf/2006.03654.pdf.

Jiang, Haoming, Pengcheng He, Weizhu Chen, Xiaodong Liu, Jianfeng Gao, and Tuo Zhao. 2019. “Smart: Robust and efficient fine-tuning for pre-trained natural language models through principled regularized optimization”. *arXiv preprint arXiv:1911.03437,* https://aclanthology.org/2020.acl-main.197.pdf.

Karhinen, Joonas, Tomi Oinas, Mia Tammelin, Antti Hämäläinen, Helena Hirvonen, Ville Mustola, Eero Rantala, and Sakari Taipale. 2021. “Vanhustyö ja teknologia. Jyväskylän yliopiston vanhustyön kyselytutkimus 2021 : Katsaus tutkimusaineistoon”. Jyväskylän yliopisto. https://jyx.jyu.fi/handle/123456789/78742.

Karhinen, Joonas, Sakari Taipale, Mia Tammelin, Antti Hämäläinen, and Tomi Hirvonen Helena & Oinas. 2019. “Vanhustyö ja teknologia. Jyväskylän yliopiston vanhustyön kyselytutkimus 2019 : Katsaus tutkimusaineistoon”. Jyväskylän yliopisto. https://jyx.jyu.fi/handle/123456789/65649.

Kim, Joo-Kyung, and Young-Bum Kim. 2020. “Pseudo Labeling and Negative Feedback Learning for Large-Scale Multi-Label Domain Classification”. In *ICASSP 2020-2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP),* 7964–7968. IEEE. https://arxiv.org/pdf/2003.03728.pdf.

Kingma, Diederik P, and Jimmy Ba. 2014. “Adam: A method for stochastic optimization”. *arXiv preprint arXiv:1412.6980,* https://arxiv.org/pdf/1412.6980.pdf.

Kowsari, Kamran, Kiana Jafari Meimandi, Mojtaba Heidarysafa, Sanjana Mendu, Laura Barnes, and Donald Brown. 2019. “Text classification algorithms: A survey”. *Information* 10 (4): 150. https://www.mdpi.com/2078-2489/10/4/150/pdf.

Lan, Zhenzhong, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. “Albert: A lite bert for self-supervised learning of language representations”. *arXiv preprint arXiv:1909.11942,* https://arxiv.org/pdf/1909.11942.pdf.

Lee, Dong-Hyun. 2013. “Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks”. In *Workshop on challenges in representation learning, ICML,* volume 3. 2. https://www.kaggle.com/blobs/download/forum-message-attachment-files/746/pseudo_label_final.pdf.

Li, Linyang, and Xipeng Qiu. 2020. “Tavat: Token-aware virtual adversarial training for language understanding”. *arXiv preprint arXiv:2004.14543,* https://arxiv.org/pdf/2004.14543.pdf.

Li, Yan, and Jieping Ye. 2018. “Learning adversarial networks for semi-supervised text classification via policy gradient”. In *Proceedings of the 24th acm sigkdd international conference on knowledge discovery & data mining,* 1715–1723. https://www.cs.ubc.ca/~amuham01/LING530/papers/li2018learning.pdf.

Li, Yang, Quan Pan, Suhang Wang, Tao Yang, and Erik Cambria. 2018. “A generative model for category text generation”. *Information Sciences* 450:301–315. https://suhangwang.ist.psu.edu/publications/CS_GAN.pdf.

Li, Zhun, ByungSoo Ko, and Ho-Jin Choi. 2019. “Naive semi-supervised deep learning using pseudo-label”. *Peer-to-Peer Networking and Applications* 12 (5): 1358–1368. https://link.springer.com/article/10.1007/s12083-018-0702-9.

Lindén, Krister, Tommi Jauhiainen, and Sam Hardwick. 2020. “FinnSentiment–A Finnish Social Media Corpus for Sentiment Polarity Annotation”. *arXiv preprint arXiv:2012.02613,* https://arxiv.org/pdf/2012.02613.pdf.

Liu, Yinhan, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. “Roberta: A robustly optimized bert pretraining approach”. *arXiv preprint arXiv:1907.11692,* https://arxiv.org/pdf/1907.11692.pdf.

Loshchilov, Ilya, and Frank Hutter. 2017. “Decoupled weight decay regularization”. *arXiv preprint arXiv:1711.05101,* https://arxiv.org/pdf/1711.05101.pdf.

Maas, Andrew L, Awni Y Hannun, Andrew Y Ng, et al. 2013. “Rectifier nonlinearities improve neural network acoustic models”. In *Proc. icml,* 30:3. 1. Citeseer. https://ai.stanford.edu/~amaas/papers/relu_hybrid_icml2013_final.pdf.

Madry, Aleksander, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, and Adrian Vladu. 2017. “Towards deep learning models resistant to adversarial attacks”. *arXiv preprint arXiv:1706.06083,* https://arxiv.org/pdf/1706.06083.pdf.

Martinez, Manuel, and Rainer Stiefelhagen. 2018. “Taming the cross entropy loss”. In *German Conference on Pattern Recognition,* 628–637. Springer. https://arxiv.org/pdf/1810.05075.pdf.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. “Efficient estimation of word representations in vector space”. *arXiv preprint arXiv:1301.3781,* https://arxiv.org/pdf/1301.3781.pdf.

Minaee, Shervin, Nal Kalchbrenner, Erik Cambria, Narjes Nikzad, Meysam Chenaghlu, and Jianfeng Gao. 2021. “Deep Learning–based Text Classification: A Comprehensive Review”. *ACM Computing Surveys (CSUR)* 54 (3): 1–40. https://arxiv.org/pdf/2004.03705.pdf.

Miyato, Takeru, Andrew M Dai, and Ian Goodfellow. 2016. “Adversarial training methods for semi-supervised text classification”. *arXiv preprint arXiv:1605.07725,* https://arxiv.org/abs/1605.07725.

Pennington, Jeffrey, Richard Socher, and Christopher D Manning. 2014. “Glove: Global vectors for word representation”. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP),* 1532–1543. https://aclanthology.org/D14-1162.pdf.

Peters, Matthew E., Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. “Deep contextualized word representations”. *CoRR* abs/1802.05365. arXiv: 1802.05365. http://arxiv.org/abs/1802.05365.

Pham, Hieu, Zihang Dai, Qizhe Xie, and Quoc V Le. 2021. “Meta pseudo labels”. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition,* 11557–11568. https://arxiv.org/pdf/2003.10580.pdf.

Puri, Raul, and Bryan Catanzaro. 2019. “Zero-shot text classification with generative language models”. *arXiv preprint arXiv:1912.10165,* https://arxiv.org/pdf/1912.10165.pdf.

Radford, Alec, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. “Improving language understanding by generative pre-training”, https://cdn.openai.com/research-covers/language-unsupervised/language_understanding_paper.pdf.

Radford, Alec, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. 2019. “Language models are unsupervised multitask learners”. *OpenAI blog* 1 (8): 9. https://www.persagen.com/files/misc/radford2019language.pdf.

Rosa, Gustavo H de, and Joao P Papa. 2021. “A survey on text generation using generative adversarial networks”. *Pattern Recognition* 119:108098. https://www.sciencedirect.com/science/article/abs/pii/S0031320321002855.

Sachan, Devendra Singh, Manzil Zaheer, and Ruslan Salakhutdinov. 2019. “Revisiting LSTM networks for semi-supervised text classification via mixed objective function”. In *Proceedings of the AAAI Conference on Artificial Intelligence,* 33:6940–6948. 01. https://arxiv.org/pdf/2009.04007.pdf.

Sajun, Ali Reza, and Imran Zualkernan. 2022. “Survey on Implementations of Generative Adversarial Networks for Semi-Supervised Learning”. *Applied Sciences* 12 (3): 1718. https://mdpi-res.com/d_attachment/applsci/applsci-12-01718/article_deploy/applsci-12-01718-v2.pdf?version=1644476961.

Salazar, Julian, Davis Liang, Toan Q Nguyen, and Katrin Kirchhoff. 2019. “Masked language model scoring”. *arXiv preprint arXiv:1910.14659,* https://arxiv.org/pdf/1910.14659.pdf.

Sanh, Victor, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. “DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter”. *arXiv preprint arXiv:1910.01108,* https://arxiv.org/pdf/1910.01108.pdf.

Srivastava, Nitish, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. “Dropout: a simple way to prevent neural networks from overfitting”. *The journal of machine learning research* 15 (1): 1929–1958. https://www.cs.toronto.edu/~hinton/absps/JMLRdropout.pdf.

Stanton, Gray, and Athirai A Irissappane. 2019. “GANs for semi-supervised opinion spam detection”. *arXiv preprint arXiv:1903.08289,* https://arxiv.org/pdf/1903.08289.pdf.

Sun, Zijun, Chun Fan, Xiaofei Sun, Yuxian Meng, Fei Wu, and Jiwei Li. 2020. “Neural Semi-supervised Learning for Text Classification Under Large-Scale Pretraining”. *arXiv preprint arXiv:2011.08626,* https://arxiv.org/pdf/2011.08626.pdf.

Ulčar, Matej, and Marko Robnik-Šikonja. 2020. “Finest bert and crosloengual bert”. In *International Conference on Text, Speech, and Dialogue,* 104–111. Springer. https://arxiv.org/pdf/2006.07890.pdf.

Van Engelen, Jesper E, and Holger H Hoos. 2020. “A survey on semi-supervised learning”. *Machine Learning* 109 (2): 373–440. https://link.springer.com/article/10.1007/s10994-019-05855-6.

Vankka, Jouko, Heikki Myllykoski, Tuomas Peltonen, and Ken Riippa. 2019. “Sentiment Analysis of Finnish Customer Reviews”. In *2019 Sixth International Conference on Social Networks Analysis, Management and Security (SNAMS),* 344–350. IEEE. https://www.researchgate.net/profile/Jouko-Vankka/publication/337982047_Sentiment_Analysis_of_Finnish_Customer_Reviews/links/5e037b0d299bf10bc3786849/Sentiment-Analysis-of-Finnish-Customer-Reviews.pdf.

Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. “Attention is all you need”. In *Advances in neural information processing systems,* 5998–6008. https://arxiv.org/pdf/1706.03762.pdf.

Virtanen, Antti, Jenna Kanerva, Rami Ilo, Jouni Luoma, Juhani Luotolahti, Tapio Salakoski, Filip Ginter, and Sampo Pyysalo. 2019. “Multilingual is not enough: BERT for Finnish”. *arXiv preprint arXiv:1912.07076,* https://arxiv.org/pdf/1912.07076.pdf.

Vlachostergiou, Aggeliki, George Caridakis, Phivos Mylonas, and Andreas Stafylopatis. 2018. “Learning representations of natural language texts with generative adversarial networks at document, sentence, and aspect level”. *Algorithms* 11 (10): 164. https://www.mdpi.com/1999-4893/11/10/164/htm.

Xia, Congying, Chenwei Zhang, Jiawei Zhang, Tingting Liang, Hao Peng, and Philip S. Yu. 2020. “Low-shot Learning in Natural Language Processing”. In *2020 IEEE Second International Conference on Cognitive Machine Intelligence (CogMI),* 185–189. https://doi.org/10.1109/CogMI50398.2020.00031. https://par.nsf.gov/servlets/purl/10228168.

Xie, Qizhe, Zihang Dai, Eduard Hovy, Thang Luong, and Quoc Le. 2020. “Unsupervised data augmentation for consistency training”. *Advances in Neural Information Processing Systems* 33:6256–6268. https://proceedings.neurips.cc/paper/2020/file/44feb0096faa8326192570788b38c1d1-Paper.pdf.

Xu, Bing, Naiyan Wang, Tianqi Chen, and Mu Li. 2015. “Empirical evaluation of rectified activations in convolutional network”. *arXiv preprint arXiv:1505.00853,* https://arxiv.org/pdf/1505.00853.pdf.

Xu, Qiantong, Tatiana Likhomanenko, Jacob Kahn, Awni Hannun, Gabriel Synnaeve, and Ronan Collobert. 2020. “Iterative pseudo-labeling for speech recognition”. *arXiv preprint arXiv:2005.09267,* https://arxiv.org/pdf/2005.09267.pdf.

Xu, Weidi, Haoze Sun, Chao Deng, and Ying Tan. 2017. “Variational autoencoder for semi-supervised text classification”. In *Proceedings of the AAAI Conference on Artificial Intelligence,* volume 31. 1. https://arxiv.org/pdf/1603.02514.pdf.

Yang, Xiangli, Zixing Song, Irwin King, and Zenglin Xu. 2021. “A survey on deep semi-supervised learning”. *arXiv preprint arXiv:2103.00550,* https://arxiv.org/pdf/2103.00550.pdf.

Yang, Zhilin, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. “Xlnet: Generalized autoregressive pretraining for language understanding”. *Advances in neural information processing systems* 32. https://arxiv.org/pdf/1906.08237v2.pdf.

You, Yang, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2019. “Large batch optimization for deep learning: Training bert in 76 minutes”. *arXiv preprint arXiv:1904.00962,* https://arxiv.org/pdf/1904.00962.pdf.

Zhang, Han, Zizhao Zhang, Augustus Odena, and Honglak Lee. 2019. “Consistency regularization for generative adversarial networks”. *arXiv preprint arXiv:1910.12027,* https://arxiv.org/pdf/1910.12027.pdf.

Zhang, Yizhe, Dinghan Shen, Guoyin Wang, Zhe Gan, Ricardo Henao, and Lawrence Carin. 2017. “Deconvolutional paragraph representation learning”. *arXiv preprint arXiv:1708.04729,* https://arxiv.org/pdf/1708.04729v3.pdf.

Zhu, Chen, Yu Cheng, Zhe Gan, Siqi Sun, Tom Goldstein, and Jingjing Liu. 2019. “Freelb: Enhanced adversarial training for natural language understanding”. *arXiv preprint arXiv:1909.11764,* https://arxiv.org/pdf/1909.11764.pdf.

Appendices

A FinBERT model and tokenizer configuration files

config.json

```
{
  "architectures": [
    "BertForMaskedLM"
  ],
  "attention_probs_dropout_prob": 0.1,
  "hidden_act": "gelu",
  "hidden_dropout_prob": 0.1,
  "hidden_size": 768,
  "initializer_range": 0.02,
  "intermediate_size": 3072,
  "layer_norm_eps": 1e-12,
  "max_position_embeddings": 512,
  "model_type": "bert",
  "num_attention_heads": 12,
  "num_hidden_layers": 12,
  "pad_token_id": 0,
  "type_vocab_size": 2,
  "vocab_size": 50105
}
```

tokenizer_config.json

```
{
  "do_lower_case": false
}
```

These configuration files are obtained from Huggingface documentation of FinBERT[1].

1. https://huggingface.co/TurkuNLP/bert-base-finnish-cased-v1