Swedish NLP Solutions for Email Classification

JOHN ROBERT CASTRONUOVO
Swedish NLP Solutions for Email Classification

JOHN ROBERT CASTRONUOVO
Assigning categories to text communications is a common task of Natural Language Processing (NLP). In 2018, a new deep learning language representation model, Bidirectional Encoder Representations from Transformers (BERT), was developed which can make inferences from text without task specific architecture. This research investigated whether or not a version of this new model could be used to accurately classify emails as well as, or better than a classical machine learning model such as a Support Vector Machine (SVM). In this thesis project, a BERT model was developed by solely pre-training on the Swedish language (svBERT) and investigated whether it could surpass a multilingual BERT (mBERT) model’s performance on a Swedish email classification task. Specifically, BERT was used in a classification task for customer emails. Fourteen email categories were defined by the client. All emails were in the Swedish language. Three different SVMs and four different BERT models were all created for this task. The best F1 score for the three classical machine learning models (standard or hybrid) and the four deep learning models was determined. The best machine learning model was a hybrid SVM using fastText with an F1 score of 84.33%. The best deep learning model, mPreBERT, achieved an F1 score of 85.16%. These results show that deep learning models can improve upon the accuracy of classical machine learning models and suggest that more extensive pre-training with a Swedish text corpus will markedly improve accuracy.
Sammanfattning

## Contents

1 Introduction 1
   1.1 Research Question ........................................... 2
   1.2 Limitations ......................................................... 2
   1.3 Societal and Ethical Considerations ............................ 3
   1.4 Sustainability .................................................... 3
   1.5 Overview .......................................................... 4

2 Background 5
   2.1 Traditional Text Classification Solutions ....................... 5
   2.2 Word Embeddings .................................................. 6
   2.3 Deep Learning in NLP ............................................ 8
   2.4 Transfer Learning ................................................ 11
   2.5 BERT ............................................................... 13

3 Methods 17
   3.1 Software .......................................................... 17
   3.2 Google Cloud ..................................................... 18
   3.3 Local Workstation ................................................ 18
   3.4 Data Preparation .................................................. 19
      3.4.1 Parkman Data Collection .................................... 19
      3.4.2 Parkman Email Formatting ................................... 20
      3.4.3 Parkman Email Text Corpus .................................. 21
      3.4.4 svBERT Text Corpus ........................................ 21
   3.5 svNLI Task ......................................................... 22
   3.6 Parkman Email Classifiers ....................................... 22
      3.6.1 SVM ........................................................... 23
      3.6.2 mBERT ........................................................ 23
      3.6.3 mPreBERT .................................................... 23
   3.7 svBERT .............................................................. 24
### CONTENTS

3.7.1 Pretraining ........................................ 24  
3.7.2 Vocabulary ........................................ 25

4 Results .............................................. 26  
4.1 Parkman Email Classification Task ...................... 27  
4.2 SVM ................................................ 27  
4.2.1 Text Filtering .................................... 28  
4.2.2 word2vec and fastText ......................... 30  
4.3 mBERT .............................................. 31  
4.3.1 mPreBERT ........................................ 32  
4.3.2 Added Vocabulary ................................. 33  
4.4 svBERT .............................................. 35  
4.4.1 Pre-training ...................................... 35  
4.4.2 Optimizers ....................................... 36  
4.4.3 svNLI Task ....................................... 36  
4.4.4 Pre-training on svNLI ........................... 37  
4.4.5 svNLI Custom Vocabulary ....................... 38  
4.5 Mixed Precision ..................................... 39  
4.5.1 fp16 .............................................. 39

5 Discussion ............................................ 40

6 Conclusions .......................................... 43

Bibliography .......................................... 44

A Model Visualizations .................................. 52  
A.1 Importance Vectors ................................ 52  
A.2 mBERT Word Embeddings ............................ 53
Chapter 1

Introduction

Seeing the need for Swedish Natural Language Processing (NLP) and in light of rapid developments in Artificial Intelligence (AI) in the area of NLP since 2018, this research undertook the development of an NLP machine learning solution for use by an enterprise with predominantly Swedish speaking customers. Such a solution would improve customer satisfaction and service as well as enterprise operations. The latest deep learning NLP language models are able to understand meaning and context in text communications better than prior solutions available in NLP research. The latest and most effective NLP language model is the Bidirectional Encoder Representations from Transformers (BERT) created by Devlin et al. [1]. The original BERT model was only trained on the English language, but subsequently the authors have made available a multilingual model trained for 100 languages (based on the sizes of the languages’ Wikipedia text corpus) known as Multilingual BERT (mBERT) [2]. However, there has yet to be created a BERT model specifically for the Swedish language.

A highly accurate Swedish language model would have great benefit to both public and private enterprises in Sweden. This study shows that efforts should be made to create a publicly available Swedish text corpus sufficiently large to enable off-the-shelf NLP solutions, such as BERT, to be pre-trained in Swedish to achieve an accuracy comparable to BERT when applied to English language NLP tasks. Customer and employee communications in the Swedish language represent a vast and untapped trove of information that is useful for improving operations and service. While the results found in this research of email categorization are insufficient for enterprise operation analysis or automated customer service functions, it demonstrates the feasibility of a comprehensive Swedish language model created with BERT.
1.1 Research Question

The goal of the research project was to compare two different NLP text classification solutions: BERT, a deep learning language model representation, and a Support Vector Machine (SVM), a classical machine learning model. Their task was to classify Swedish language customer emails from a client (Parkman). To examine this question, three versions of the SVM model and four versions of the BERT model were constructed to identify which version of the two models, SVM or BERT, most accurately classified the client emails.

Additionally, a novel version of BERT (svBERT) was created for this research by pre-training the model with only Swedish language corpora to determine whether or not the accuracy of the Swedish email classification task could be improved further. In order to better assess svBERT’s Swedish language modeling ability, a Swedish version of the Cross-lingual Natural Language Inference (XNLI) [3] Sentence Pair Classification task (svNLI) was created. The evaluation of svBERT’s performance on two different kinds of NLP tasks served as a comprehensive way to evaluate svBERT’s understanding of the Swedish language.

1.2 Limitations

The single most limiting factor for the research conducted for this thesis relates to the sizes of the Swedish text corpora used in pre-training the deep learning language model (svBERT). There were financial limitations as well due to the fact that the cost of exploring the potential of BERT to classify emails was significant because of the substantial computing resources required to train it. Additionally, an adequate knowledge of cloud computing tools was required to explore BERT’s capabilities, which may represent a significant limitation to small enterprises without cloud computing experts.

The largest Swedish text corpus assembled is the Gigaword corpus by Eide, Tahmasebi, and Borin [4]. It contains much of the content made available by Språkbanken [5] at the University of Gothenburg, including the Swedish Wikipedia. According to Devlin et al. [1]: "It is critical to use a document-level corpus rather than a shuffled sentence-level corpus... in order to extract long contiguous sequences." This makes the Wikipedia source the most coveted kind of data for the pre-training process. The BERT authors used the English Wikipedia which is a 30GB compressed file, more than 20 times larger than the Swedish Wikipedia at 1.4GB [6]. The format of the corpora from Språkbanken...
was a single sentence per line. However, there was no information to indicate whether or not a sentence belonged to a document. This information loss likely harmed BERT’s Next Sentence Prediction objective of pre-training.

1.3 Societal and Ethical Considerations

Some of the most significant ethical issues raised by research in Artificial Intelligence will be driven by NLP capabilities. The ambiguity in text interpretation, the creation of biased and fake news articles, and even possibly the successful development of a system that can pass the Turing Test [7] are current societal and ethical issues that should be addressed when creating NLP applications.

Machine learning models trained on data sources such as text corpora can adapt some of the prejudices inherent in the language of the text such as gender and ethnic biases as well as hate speech. For example, new systems that learn by analyzing thousands of online discussions may associate a term like president much more frequently with one gender than another. The models will also learn from the mistakes and biases that may be contained within it. Learning a cultural bias may not be as great a problem in systems such as the one created in this investigation because it is limited to the particular task of email classification. The results do not obviously lend themselves to conversational biases involving gender, race, or ethnicity. A functional system would need to be vetted for such prejudices. A recent example being David H. Hansson, the creator of Ruby on Rails, and the disparity of qualifying credit for an Apple credit card between him and his wife [8].

1.4 Sustainability

The rapid rise of Information Technology, such as Cloud Computing, is at odds with sustainability. The exponential growth in computing power is accompanied by the exponential growth of energy consumption. The affordability of cloud services masks the vast energy resources required to run such server facilities. It is important to recognize the ways in which tasks that consume large amounts of computational power justify the resources requirements.

The quest for near perfect scores on NLP benchmark tasks such as GLUE [9] and SQuAD [10] is at odds with sustainability. There is an exponential trend in a model’s parameter size starting with AllenNLP’s ELMo [11] at 94 million, Google’s BERT-Large at 340 million, OpenAI’s GTP-2 [12] at 1.5 billion, and NVIDIA’s Megatron [13] at 8.3 billion. The resources required
to train successively complex models can be astounding. It is estimated that power costs of training Megatron is that of an average American’s annual power consumption [14].

Due to the high power consumption of using GPUs to train neural networks like the NVIDIA Tesla V100 Tensor Core GPU Card [15], which requires 250 Watts and offers 16 RAM, Google developed an Application-specific integrated-circuit (ASIC) optimized for essential deep learning computations like matrix multiplication. It is known as the Tensor Processing Unit (TPU). The Google TPU v2 used in this research consumes 40 Watts and offers 64 RAM, meaning this single TPU is capable of the same training as 4 V100 Cards at 25 times less power consumption.

1.5 Overview

What follows is a detailed explanation of BERT and the advances in NLP of the novel approaches to accomplishing NLP tasks through the years. It covers the progression from using machine learning models with word vectors to solve basic NLP tasks to implementing deep learning architectures to create pre-trained language models to solve more advanced NLP tasks. In Chapter 3, there is a detailed description of the methods of the different modifications made to BERT to perform the Parkman Email classification from the aggregation of email data into a text corpus to the actual downstream fine-tuning. It also covers the attempt to create a Swedish pre-trained language model from BERT (svBERT). The creation of svBERT was a more extensive process as it required its own specialized data aggregating techniques and pre-training methods using Google Cloud TPUs. It also required a generalized Swedish NLP Sentence Pair Classification task (svNLI) for additional evaluation of svBERT. The results of this research are presented in Chapter 4 of both the Parkman and svNLI classification tasks and different optimization techniques used in the BERT models. In Chapter 5, there is a discussion of the findings of the research and on different ways that it can be extended. Finally, Chapter 6 presents a conclusion of the discussion and the research as a whole.
Chapter 2

Background

In the autumn of 2018, Devlin et al. [1] released a novel NLP deep learning model: Bi-directional Encoder Representations from Transformers (BERT). BERT represents a significant advance in the NLP field because it achieves state of the art results on different NLP tasks such as Named-entity Recognition (NER) and Question Answering without requiring task-specific architectures. This thesis examines its Text Classification abilities.

Text classification is the task of categorizing text according to its content. In the context of NLP, an advanced model like BERT can accomplish this task without requiring a human reader. Traditionally, text classification has been performed with adequate results using a classical machine learning model known as the Support Vector Machine (SVM). Recently developed deep learning language models can perform text classification at an even higher level of performance. The advantage of the newer deep learning models is that they learn a representative model of a language. This makes it possible to accomplish many different NLP tasks that would be ill-suited for an SVM.

2.1 Traditional Text Classification Solutions

Email spam is a ubiquitous problem. Fortunately, this problem can be solved as with a simple binary classification task. There are only two categories: spam or not spam. It is a perfect candidate for a NLP machine learning solution. However, the question remains as to how to best represent the emails as input data for the classifier model.

In Information Retrieval, the creation of a searchable database is accomplished with word vectorization techniques. The Bag of Words (BoW) method is a simple word vectorization process in which all words are counted in all
documents. The result is a large, sparse matrix that's dimension is all of the words by all of the documents. The downside to this is that words like prepositions that frequently occur in sentences will be overrepresented despite their irrelevance to a search query of the documents. The Term Frequency Inverse Document Frequency (tf-idf) method is an effective way to solve this problem. The term frequency part consists a word count across all documents, and the inverse document frequency part suppresses words like prepositions that have high counts. Here the words are given a weight of relevance instead of a count. In this way, the rarer words have a greater significance in a search query.

The representation of words as vectors from the BoW or tf-idf methods as input data for a machine learning model like a SVM makes it possible to use the model as a classifier to categorize documents into different classes according to their content. The SVM is a powerful model. Its goal is to find a decision boundary which best separates different classes of vectors of any dimension called a hyper-plane. This is ideal for the Parkman Email classification task with its 14 different categories of emails. The SVM is able to learn with a cost function. If a vector is misclassified, then the cost function updates the SVM's weights to better classify the vectors. Since the SVM is creating a decision boundary between the different vectors, it is agnostic about the source of the data as long as it's in vector format. This means that both BoW and tf-idf word vectors can be used, or even word vectors created by deep learning algorithms such as word2vec.

### 2.2 Word Embeddings

The introduction of word2vec created by Mikolov et al. [16] represented a novel approach to create word embeddings. This unsupervised learning algorithm is constructed as a two layer neural network that takes as input a large unlabeled text corpus and a pre-selected dimension $n$ and outputs $n$-dimensional vectors for all the words in the corpus. These word2vec word embeddings are able to capture relationships between words with similar meanings. This means that words with similar meanings are close to each other in the vector space, and their word associations are of equal distance to each other. This is depicted in Figure 2.1 with the word Sweden. This can aid greatly in applications like Neural Machine Translation (NMT), which makes it possible to find translations for words based on their relative locations in the word2vec space. It is also possible to use word2vec as a document classifier by clustering all of the document's words together to assign a category. However, word2vec has its shortcomings in that it is essentially a dictionary that maps words to vectors. Relevant words
Several years later, Bojanowski et al. [17] proposed an improved word2vec model, called fastText. Instead of generating a vector for each word, each word is treated as a collection of its $n$-grams. This means that for a 3-gram (meaning groups of 3 letters) of the word thesis would be: {the, hes, esi, sis}. Thus, each word vector is the sum of its $n$-grams vectors. This allows fastText to generate better vectors for rare words because they will share $n$-grams with other words. fastText offers an effective solution to the problem of out of vocabulary words by constructing unknown words in this way.

A shortcoming of both word2vec and fastText is their inability to detect the polysemy in a language. This means that they are unable to fully capture different meanings of the same word which is something that can be inferred based on a word’s context in a sentence. Capturing word’s contextual information is only possible with more advanced neural networks like Recurrent Neural Networks (RNNs) with Long-Short Term Memory (LSTM) that are designed to "remember" previous words in the sentence. These models paved the way for the more advanced Word Embeddings in use today.
2.3 Deep Learning in NLP

In 2012, Krizhevsky, Sutskever, and Hinton [18] published their ImageNet Classification with Deep Convolutional Neural Networks (CNNs) paper. This marked a watershed moment in the study of Deep Neural Networks (DNN). The findings produced a performance increase of over 40% in the ImageNet Challenge[19] and proved the viability of DNNs in Computer Vision (CV) and Deep Learning which changed the field forever.

These CNNs may be ideal for CV tasks but not for NLP ones. The input data of a picture for a basic CV task is represented by pixels corresponding to colors and shades. The network processes each of these pictures independently; the order of the pictures is irrelevant to learn. In order to capture things like context of words in sentences, the data must be treated as a sequence of words instead of processing the words independently of each other. This is why a feed-forward neural network like that of word2vec is unable to infer the context of words.

RNNs provide a way to process sequential data. The training objective of a RNN is to predict the next token in the sequence like a letter. The prediction is based on the previous tokens of the sequence with the use of a feedback loop. The loop receives the outcome of the previous step of the sequences back as input to affect the outcome of the current step of the sequence. This can seen in Figure 2.2a. In this way, RNNs create a mechanism for memory, much like the human brain, which is precisely what is needed for contextual understanding.

However, RNNs have a notorious problem: Vanishing Gradients. Hochreiter [20] was the first to conceptualize this problem. When discussing Gradients in the context of deep learning, it is a reference to the Gradient Descent algorithm. This is the most widespread optimization technique in machine learning today. For example, suppose there exists a function that takes an image as input, and this function always correctly classifies the input image. A neural network is attempting to best estimate this function by learning to perfect its own parameters through the feedback of its cost function. The cost function helps the network to learn by evaluating if the prediction made matches the true value. The cost function calculates the error and updates the network’s parameters accordingly.
Figure 2.2: Figures of a Recurrent Neural Network in (a), and a LSTM Cell in (b). The RNN always considers its previous state at its current input as depicted by the circular arrow. Each LSTM has three gates (input, output, and forget) to maintain or discard memories of previous states.

A cost function can be visualized as a large bowl. The model starts at the top of the bowl and its goal is to reach the bottom of the bowl by minimizing the cost function. It accomplishes this by calculating the gradient of this function which is done by taking a partial derivative of the parameters and selecting the lowest value or the path of steepest slope of descent. The Vanishing Gradient problem for RNNs occurs when using the Back Propagation algorithm to calculate its gradient. Back propagation uses the chain rule to compute all the partial derivatives of the parameters. When an RNN is back propagating its past steps, eventually those gradients become so small after several compositions that they have no influence on the current step. This is the RNN’s short term memory problem.

LSTMs provide an effective way to combat the short term memory problem. Their architectures allow them to maintain a cell state with input, output, and forget gates to regulate information into the cell. The gates make decisions on what to remember, forget, or use. This means that the information gathered from previous steps in the past will aid in determining what is useful to be considered in the current step in the sequence as seen in Figure 2.2b. This is a great improvement but it is still limited by its unidirectional input sequence processing. It is possible to process the input sequence in both directions by using a Bidirectional LSTM (BiLSTM) that can perform forward and backward passes. However, each pass must be conducted separately, so it fails to capture the input sequence simultaneously bidirectionally.

Sutskever, Vinyals, and Le [21] proposed the use of LSTMs in an Encoder-
Decoder model in 2014. The Encoder LSTMs take the input sequence and compress it into a context vector to be passed to the Decoder LSTMs to transform it into output. This revolutionized tasks like sequence to sequence translation. Encoder-Decoder LSTM models are still capable of achieving state of the art results today. However, their architecture dictates that their internally representative context vector must be represented by a fixed size. This limitation makes it difficult to handle long sequences larger than the context vector size.

The following year Bahdanau, Cho, and Bengio [22] introduced the Attention mechanism as a solution to this problem. The Attention mechanism can be conceptualized as one scanning a paragraph to determine which words are most relevant to its overall meaning. This frees the constraint of a fixed internal context vector; the internal vector now shares all the hidden states of the encoder and decoder. Thus, the output relies on the all input states that the model has paid most attention towards and no longer only the last state. It is also possible to easily visualize the words that the Attention head finds most relevant by inspecting the output and allowing for better insight as to how the model is learning.

Subsequently, Vaswani et al. [23] proposed the Transformer architecture using Multi-headed Self-Attention. It avoids the use of RNNs in favor of using

Figure 2.3: The Transformer architecture with Multi-headed Attention based on its depiction from Vaswani et al. [23].
Encoders and Decoders which are composed of a multi-head Self-Attention layers. Each Encoder has a Self-Attention layer and a feed-forward network layer. The Encoder creates query, key, and value vectors by each input word embedding with the corresponding matrices to produce a score of the importance of that word. The Multi-headed Attention means that each Attention head maintains its own key, value, and query matrices, as well as separate Attention scores. The scores are then concatenated and multiplied with a score matrix to produce the final score vector and finally fed into the feed-forward network.

The Decoder has the same structure except for a specific Encoder-Decoder attention layer to handle the key and value vectors sent from the Encoder layers at each step. It also maintains its own query vector created by its Self-Attention layer. One important difference is that the Decoder can only see previous words of the input. Any future words in the sequence are masked, so that the training objective is to predict the next word. Finally, the decoder output is fed into a Linear layer and Softmax layer to choose the value with the highest probability corresponding to a word. The Decoder does not handle the input sequentially in the same way as a RNN, so the positional data of words in a sequence must be encoded to the input data. This is accomplished by adding positional information to the input word embeddings. This allows the Transformer to consider the order of the sequence. Figure 2.3 is a representation of the Transformer architecture.

The Transformer architecture has inspired OpenAI’s GPT and Google’s BERT models. The two models are like the yin and the yang. GPT is only Decoder stacks, while BERT is only Encoder stacks. Both models form the basis of the some of the most state of the art language models to date.

2.4 Transfer Learning

An unexpected consequence of the high performing ImageNet DNN model with supervised learning was that different models’ performance would be greatly increased by using the ImageNet model’s weights as their initialized weights. It was as if the new model had already learned the basics of classifying images and could skip to learning the specifics for the task at hand. This is known as Transfer Learning. NLP’s ImageNet Moment [24] of Transfer Learning has now arrived; the latest NLP models are pre-trained to create a language model representation without having to first build a NLP task specific architecture. These pre-trained language models can then be used to build unique, state of the art NLP solutions. The two most well known techniques to prepare pre-trained language models for specific NLP tasks are feature-based and fine-
One of the most advanced unsupervised feature-based models, ELMo [11], introduced deep contextualized word embeddings. This represented an augmentation of word embeddings like ones created by word2vec. ELMo uses a Bi-directional LSTM (BiLSTM) to capture contextual information from both left to right and right to left sequence processing. This enables ELMo to capture a word’s polysemy based on its context in a sentence. It is possible to achieve state of the art results when these contextualized word embeddings are incorporated in pre-existing NLP task architectures like for Question and Answering and Sentence Pair Classification. Task architectures for handling NLP tasks are absent from ELMo, and its embeddings must be used in conjunction with pre-existing structures to perform tasks.

An example of the fine-tuning approach is implemented in the Generative Pre-trained Transformer (GPT) model from the work of Radford et al. [12]. The GPT architecture is based on the Transformer but consisting of only Decoder stacks. This is a smart choice for pre-training a language model because of the Decoder’s design principle of masking the future tokens in the sequence, and its learning objective predicting the next token. This means that GPT is capable of Text Generation. In fact, the successor to GPT, GPT-2, is so adept at generating text that the OpenAI creators debated ever releasing the code to the public for fear of GPT "being used to generate deceptive, biased, or abusive language at scale" [25]. The GPT model eliminates the need for task-specific architecture by instead fine-tuning all of pre-trained model’s parameters on the given task’s input. A visualization of both model’s architecture is shown in
Both of these approaches for pre-training models for use in downstream tasks are effective and achieve state of the art results. The authors of BERT sought to improve upon the fine-tuning approach by introducing a new Transformer model only constructed with Transformer Encoders.

## 2.5 BERT

BERT combines the Bi-directional nature of ELMo with the Transformer inspired design of GPT with one important difference: BERT is able to capture the context of a word from both directions in a sentence simultaneously. ELMo does technically capture context in both directions, but with separate forward and a backward passes through the sentence, and GPT only processes the input sentence in a left to right manner. The downside of using a fine-tuning approach pre-trained with a unidirectional language model like GPT can be seen when applied to a task like Question Answering because context is paramount to understanding a question. If only one direction of context is captured, then half of potential context data is lost.

There were two different BERT model architectures outlined in the paper: BERT-base and BERT-large. BERT-base has 12 Transformer Encoder blocks and 12 Self-Attention heads with a 786 feed-forward network size. BERT-large has 24 Transformer Encoder blocks and 16 Self-Attention heads with a 1024 feed-forward network size. BERT-base was created to be roughly the same size as GPT for a performance comparison, and BERT-large was created to unlock the full potential of the model architecture.
A property of Transformer Encoders is that the entire sentence can be seen at all intervals. This meant that the BERT authors had to come up with a completely different pre-training method that could work with this Encoder attribute. They looked to the Cloze task created by Taylor [26] and its concept of the Masked Language Model (MLM). By implementing a MLM task, the authors could take advantage of the Encoder’s property of a non-masked sentence input by randomly words masking words in the sentence. This made BERT’s learning objective to predict the masked words based solely on their context. The authors chose to mask 15% of all words of the input as as well as switching the words around in the sentence. They also created a Next Sentence Prediction (NSP) task as an additional learning objective to condition the model for text-pair representation tasks like Sentence Pair Classification.

In order to carry out these two pre-training objects, BERT is pre-trained on a large unlabeled text corpus. The authors used the English Wikipedia and BookCorpus for a total of 2.5 billion words. BERT’s vocabulary file is composed of the 30,000+ most commonly occurring words in its unlabeled text corpus. The vocabulary file is essential in the tokenization process. BERT’s Word Embeddings are unique and are visualized in Figure 2.6. The input words are converted into sub-words by the WordPiece Tokenizer that is based on the work of Wu et al. [27]. This method allows for graceful handling of out of vocabulary words or misspellings that are absent from BERT’s vocabulary file. For example, the phrase “We use Encoders.” tokenizes to: [CLS], we, use, en, ##code, ##rs, [SEP]. Despite the word “encoders” not present in BERT’s vocabulary file, the tokenization is able to reconstruct the word with the sub-units: en, ##code, ##rs. The tokenization also uses special tokens. [CLS] indicates the start of a new sequence. [SEP] is used for segment
embeddings for the NSP to mark the end of the first segment and the beginning of the next segment. [MASK] is used for the MLM to hide the input token from the Encoder. Finally, the positional embeddings are assigned to each token according to their order in the sequence. It is necessary that the input sequences combine both positional and segmental information so that BERT can learn contextual information at both word and sentence levels. This enables BERT to perform a wide variety of NLP tasks with little task-specific architecture needed.

Figure 2.7: Illustrations of Fine-tuning BERT on Different Tasks [1]. The svNLI task is modeled after the MNLI task [28] in Figure A and the Parkman task is modeled after the CoLA task [29] in Figure B. Based on Figure 4 of the BERT Paper [1].

The authors of BERT provided example code for NLP task architectures for which it was best suited. This research required a Sentence Classification task and a Sentence Pair Classification task. A BERT code example for a Sentence Pair Classification task was in evaluating the Corpus of Linguistic Acceptability (CoLA) by Warstadt, Singh, and Bowman [29]. The CoLA data set contains sentences from linguistics literature and labeled for acceptability. BERT’s objective is to predict the correct label. The code example that was most useful for the research’s Sentence Pair Classification task was in evaluating the Cross-lingual Natural Inference (XNLI) task by Conneau et al. [3]. The XNLI corpus contains crowd-sourced human translations of the Multi-Genre Natural Language Inference (MNLI) by Williams, Nangia, and Bowman [28]. The task pairs two sentences together and labels the relationship between them as either neutral, contradiction, or entailment. This is also happens to be the
only readily available way to evaluate the Multilingual BERT (mBERT) [2] model.

According to the mBERT documentation, the pre-training data was a concatenation of the 100 languages with the largest Wikipedia dumps. Since the size of these dumps vary greatly, the authors smoothed the weights of the data both during the pre-training data and WordPiece creations in order to boost the languages with smaller corpus sizes. The example given was that if English would normally be sampled a 1000 times more likely than Icelandic, the smoothing process brought that figure down to just a 100 times more likely. In theory, this means that mBERT can perform competently in over 100 different languages.

BERT uses a recent variation of the Adam optimizer known as Adam with weight decay (AdamW) Loshchilov and Hutter [30]. Despite the time-saving advantages of the Stochastic Gradient Descent algorithm, the training time for a large model like BERT is significant even when using multiple top of the line hardware chips like the Google TPUv3 or the NVIDIA Tesla GPUs.

Memory constraints are an issue in any deep learning model. BERT has a particularly high demand for memory due to its hundreds of millions of parameters. In its paper, the batch size and sequence length used in pre-training added up to 128GB. Despite using the latest Google TPUs, it still took around 3 days to complete BERT’s pre-training. You et al. [31] at Google Brain proposed a new kind of optimizer called the Layer-wise Adaptive Moments optimizer for Batch training (LAMB). LAMB provides a novel solution to pushing the limits of the batch size during pre-training. It is possible to greatly reduce pre-training time with a large batch size.
Chapter 3

Methods

The research was carried out in two separate phases. The email classification task was completed first followed by the creation of svBERT and its evaluation tasks. These two phases shared many of the same data preparation techniques. The first sections of the methods describe the software and hardware specifications and a description of the shared data management techniques. Subsequent sections describe the specific methods taken to carry out the research for each phase.

3.1 Software

There were different choices in software between local development and Google Cloud usage. The most significant difference was the choice between PyTorch [32] and TensorFlow [33]. The original BERT code was written in C++, but they authors chose to release the code in Python with the TensorFlow library because of its ease of use and Google Cloud TPU API. At the time, the only way to utilize a Google Cloud TPU for pre-training was with Tensorflow implementation of BERT, so TensorFlow was used in the Google Cloud instance. However, the local workstation used the PyTorch implementation of BERT [34]. The main reason for using PyTorch was because of its support of NVIDIA apex package [35]. This package enables a mixed floating point precision that can scale down from the default float32 values to float16 values that can significantly reduce GPU memory constraints.
3.2 Google Cloud

Using the Google Cloud TPUs was the only way to carry out the pre-training of the Swedish language BERT (svBERT) due to its large memory requirements. The local workstation was only suited to carry out the pre-training of mBERT on the Parkman email corpus and both the svNLI and Parkman tasks.

The pre-training was performed with both a single TPU v2-8 with 8 cores and 64 GB RAM and a TPU v2-256 pod with 256 cores and 2TB RAM [36]. TPUs require the use of Google Cloud Storage Buckets to function [37] and a single VM instance [38]. There is an option to use Google Collab [39], a Jupyter notebook type of environment, but it lacked the fine-grained control needed for pre-training svBERT.

The operating system (OS) used for the Google Cloud VM instance was Debian 9.2 [40]. To run BERT Python 3.6.8, TensorFlow 1.13.1 [33], NVIDIA CUDA 10.0 [41], and NVIDIA CuDNN 7.5.2 [42] were used.

3.3 Local Workstation

The workstation OS used was Kubuntu 18.04.2 [43] because of the user interface and the Ubuntu software repositories. The text preparation and fine tuning was done by an Intel i7 7700, 32 GB DDR4 RAM, Asus NVIDIA GTX 1080Ti-FE with 11 GB RAM [44]. Additionally, the GPU was overclocked from 1480MHz to 1950MHz, or about 30% faster. Water cooling was used on both the GPU and CPU, so that overheating was never an issue. To run BERT related and Parkman tasks Intel® Distribution for Python [45] 3.6.8, PyTorch 1.0 [32], NVIDIA CUDA 10.0 [41], and NVIDIA CuDNN 7.5.2 [42] were used.

The Liquorix kernel [46] was chosen over the standard Kubuntu distribution kernel because of its performance benefits during workload due to features like Zen Interactive Tuning and the offers features like MuQSS (formerly known as the BFS) created by veteran kernel developer Colin Kolivas [47].

Various kernel security parameters for Spectre-V1 [48], Spectre-V2 [49], and Meltdown [50] (infamous exploits affecting nearly all x86 processors that allow an attacker to read all memory) were disabled in favor of additional performance. These security patches were extremely controversial because of the varying performance impact they caused.
3.4 Data Preparation

Formatting text to input to BERT was of critical importance. Having such things as hyperlinks or html tags could end up hurting the performance of the MLM and SentencePrediction tasks because they are not words. This is a perfect application for using regular expressions to filter out unwanted strings.

However, due to the large size of the unlabeled text corpus for pre-training BERT of 10 GB, conventional methods of reading these files into memory became impossible for text filtering with regular expressions. This was a perfect use case for Big Data methods with Apache Spark [51] on Apache Hadoop [52]. The Spark RDDs were configured to be cached to disk because even 32 GB RAM was not enough to hold everything in volatile memory. The files were loading into Hadoop and by using pySpark, an API for Spark in Python, in a Jupyter [53] notebook, the task became manageable. All text formatting, word counting, and other text operations were carried out with this combination of tools.

3.4.1 Parkman Data Collection

Parkman i Sverige AB [54] is a Swedish-owned company with extensive experience in establishing and operating large and small parking facilities and garages. Parkman prides itself on personal customer relations through their expert customer service. They are also a technology oriented company that is eager to find new solutions to make their operations more efficient. Creating an email classification to aid their customer service agents is something that they were keen to investigate. Parkman provided their entire customer email correspondences to since 2014 to Digital as a usable data set.

The Parkman client uses a Microsoft Outlook service [55] for their email provider. In order to access the emails from the server, the Microsoft Graph API [56] was used to connect and to query the databases. Since the client had not previously done any classification work on their emails, a simple front-end web app was created using Vue.js [57] for a user to see the emails, analyze them, and then select a category for it. The categorized email data was then saved to PostgreSQL database [58]. In the end, there were 14 different possible categories of common questions consistently asked by customers. There were no emails assigned more than one label.
3.4.2 Parkman Email Formatting

A program was needed to clean all metadata, links, attachments, greetings and signatures from the Parkman emails. This utilized extensive regex rules for links, emoticons, emojis, and DOS line endings. A helpful Python package called talon [59] by Mailgun Technologies, Inc. [60] was used because it uses a machine learning model to extract plain text from HTML, message quotations, and email signatures. Extracting all emails from within a forwarded email chain was particularly challenging.

The end result was a total of 650 emails to be used for training and testing. The Parkman training and test emails were saved in JSON format with the keys id, subject, content, and category to separate files. These files were able to be used by both the SVM models and the BERT models for training and evaluating.

![Figure 3.1: The 14 categories of email classification. Each color represents a different category.](image)

Each of the 650 emails was classified by the staff of Digital Edge. As can be seen in Figure 3.1, the number of emails in a category can vary significantly. The size of the training set was 435 emails and the test set was 215 emails. There is an uneven distribution of data with just three categories accounting for almost 60% of all emails.
3.4.3 Parkman Email Text Corpus

In order to create the unlabeled text corpus for BERT pre-training the entire Parkman email database was saved to a MongoDB database [61]. After emails were filtered for metadata, they were formatted to the BERT input specifications of one sentence per line and the end of the email delimited by a new line. This is necessary for BERT’s Word Embeddings processing to do the pre-training tasks of both Word Masks and Next Sentence Prediction. sPaCy [62] was used for sentence segmentation to format it to that specification. The created Parkman unlabeled email text corpus was as a 22MB text file of 3,667,485 words.

3.4.4 svBERT Text Corpus

The creation of the svBERT involved aggregating the largest Swedish text corpus possible, formatting it into the document format required by the BERT architecture. First attempts at formatting the svBERT text corpus were done using Regular Expressions (regex) with shell scripts with sed [63] and awk [64], but finally the Big Data methods outlined above were used with success. The Swedish Wikipedia corpus Wikimedia Downloads [6] was an important source. It was extracted with [65]. After text processing, the total number words was 199,517,608. Also, the online database of the University of Gothenburg’s Språkbanken Text [5] provided a wide variety of text dumps such as 8 Sidor - Lättlästa nyheter [66], Aftonbladet [67] news articles, the Swedish EUbookshop [68] collection, and Swedish subtitles from OpenSubtitles [69] of the OPUS - an open source parallel corpus [70] created by Tiedemann [71]. Additionally, portions of the Swedish Culturomics Gigatex Corpus by Eide, Tahmasebi, and Borin [4] was used. This brought the total world count of the corpus to 615,773,841 words which tripled the word count of the Swedish Wikipedia dump alone.

At run-time, the pre-training BERT converts the text corpus into input data called tfrecords. The BERT authors recommended converting the text to tfrecords before pre-training in order to reduce the training time with a provided Python script. The resulting tfrecords are significantly larger than the text corpus, especially when setting the sequence length to the maximum value of 512. The 4 GB text corpus resulted in a tfrecord size of 170 GB. The text corpus was split into smaller text files, and eight creation scripts were run simultaneously to decrease the creation time. It was necessary to then upload the tfrecords to a Cloud Storage bucket in the US. This is because the TPUv2-v256 pod provided by TensorFlow Research Cloud [72] was located...
in the Ohio region. The parallel composite upload parameter was used in the Google Cloud SDK [73] gsutil upload command to increase performance of the transfer.

### 3.5 svNLI Task

The creation of the svNLI dataset first required Google Translate API [74] to translate the English section of the XNLI dataset [3]. The XNLI data set is contained in tab separated value file (tsv) format. Modin [75], a high-performance version of Pandas [76], was used to read the tsv file into a Dataframe. The sentences were sent one by one to be translated for a total of 392,396 training sentences and 39,841 test sentences that had to be translated. The sentences also had to be tokenized, so again sPaCy was used for that. A word count was also performed for modifying BERT’s vocabulary file. Finally, the Swedish sentences were appended to the Dataframe and exported to a new tsv file to be read into BERT for the svNLI task.

In order to evaluate the svNLI task, the BERT code to evaluate the XNLI task was modified to add Swedish as an additional language option. The hyper parameters used were a batch size of 32, a sequence length of 128, a learning rate, $\eta$, of $5e^{-5}$, and 2 epochs. The pre-training on the svNLI dataset was created from all the sentences from the svNLI. The tfrecords were made of sequence length 128, 15% masked tokens, and a maximum prediction per sequence of 20 tokens. The hyper parameters used for the pre-training was a batch size of 32, an $\eta$ of $3e^{-5}$, 5k warm up steps, and 45k training steps.

The 38,431 most occurring words were added to mBERT vocabulary of 119,547 words for the vocabulary expansion evaluation. The new vocabulary size was 157,978. Since the vocabulary size dictates the dimension of BERT’s Word Embedding layer, during run-time BERT will notice that there is a mismatch between the two. In order to get around this, the TensorFlow verification during model initialization was disabled, concatenated 38,431 additional randomly initialized weights to the Word Embedding layer, and successfully exported the custom vocabulary sized model.

### 3.6 Parkman Email Classifiers

The training and test sets were 434 and 215 respectively, or a 33% test set size. The same training and test sets were used in all of the SVM and BERT evaluations. The chosen evaluation metrics were accuracy, F1, and total correct. The
F1 score is a common metric in text classification because it evaluates not only recall and precision, but also the positives and false negatives to rate the overall quality of the incorrect choices. The confusion matrices, email distribution, cross-validation, and predicted label charts were done with Matplotlib [77]. Tensorboard was used to create the loss plots for the BERT models.

### 3.6.1 SVM

The tf-idf SVM was created as a LinearSVC from the scikit-learn package [78]. The training and test data was processed using the TfidfVectorizer with a minimum document frequency of 5, sublinear tf scaling, l2 normalization, and a n-gram range of 1-2. Then a pipeline was made with the TfidfVectorizer and a CalibratedClassifierCV wrapping the LinearSVC for using cross-validation for calibration.

The word2vec model was created by training it on the Parkman email text corpus. NLTK [79] was used to tokenize the corpus and then the model was trained using gensim’s Word2Vec training method. The same thing was done with the fastText model. These models were sufficient to load into memory. Finally, these models were fed into sklearn’s TfidfVectorizer to make use of the word2vec/fastText vectors and then to feed the LinearSVC for classification.

### 3.6.2 mBERT

The mBERT model was downloaded from the links Google provided on their Github repository. This model is only compatible with TensorFlow, so it was converted to a PyTorch implementation with the help of a conversion script from PyTorch Transformers [34]. The Parkman Email Classification task architecture was created by modifying the pre-existing BERT CoLA task [80] implementation. Fine-tuning and evaluation was performed on the Parkman emails. The hyper parameters used for fine-tuning the Parkman task were a batch size of 32, a sequence length of 128, and an $\eta$ of $3e^{-5}$ for 10 epochs. Negative log likelihood was used as the classification loss function as specified in the BERT paper. Early stopping and Dropout were used to prevent overfitting. All the BERT models used in this research used a softmax function to predict categories.

### 3.6.3 mPreBERT

After the Parkman Email Text Corpus was created, it had to be converted into a tfrecord. Due to the small size of the corpus it was unnecessary to split the
file, and the conversion time was short. In order to pre-train on the corpus, it had to be converted into tfrecords. This was accomplished with the use of PyTorch-Transformers [34] generation script. This script contained a sentence weighting sample method that was not used in the BERT author’s tfrecord creation process. This meant that during the random document selection to create a tfrecord the documents with longer sentences were more likely to be selected. It improved the quality of pre-training because BERT’s pre-training will not benefit from short documents. Also, the Whole Word Masking method was implemented that was absent in the original BERT release. With Whole Word Masking a sequence like [we, use, en, ##code, ##rs] could be masked as [we, use, en, [MASK]], but without the MASK would treat subwords like ##code or ##rs the same as regular words like we. This could result in a sequence like [we, use, en, [MASK], ##rs], which does not make much sense to learn. The BERT authors showed that with the Whole Word Masking the BERT-large performance on the SQuAD task increased about 1.5%.

The parameters chosen for the pre-training were a sequence length of 512, 15% masked tokens, and a maximum prediction per sequence of 20 tokens, which are recommended by the BERT authors. The hyper parameters used for the pre-training was a batch size of 32, an $\eta$ of $3e^{-5}$, 5k warm up steps, and 45k training steps. Cross-entropy loss was used for the MLM and NSP pre-training objectives as specified in the BERT Paper. The apex [35] package was used for mixed precision fp16. For the vocabulary expansion comparison, the 100 most occurring proper nouns, like Parkman or Kista, were added to BERT’s vocabulary file.

3.7 svBERT

Pre-training a BERT model from scratch is straightforward. The greatest care must be taken in the preparation of the text corpus and the vocabulary file. In order to increase the quality of the tfrecords from the corpus data, the sentence weighting method was ported to TensorFlow based on the PyTorch Transformers implementation, as well as Whole Word Masking.

3.7.1 Pretraining

After the tfrecords were transferred to the Cloud Bucket storage, the VM instance utilized the TPU to carry out the pre-training. Whole Word Masking was not applied to the tfrecord creation for the TPUv2 pre-training data. Also, tfrecords with a sequence length of 128 were used for 900,000 training steps
while 100,000 training steps used tfrecords with a 512 sequence length. This followed the BERT authors advice for dealing with hardware limitations, since the TPUv2-8 only had 64 GB RAM and couldn’t handle both a high batch size and sequence size. The TPUv2-256 did not have this constraint, so the entirety of the pre-training was done with 512 sequence length tfrecords. Additionally, Whole Word Masking was used in the TPUv2-256 tfrecord creation.

The hyper parameters chosen for the pre-training were an $\eta$ of $1e^{-4}$, 1000k training steps for the AdamW configurations, 125k training steps for the LAMB configuration, a 10% warm up steps to training steps ratio, 15% masked tokens per sequence, and a maximum prediction per sequence of 20 tokens. There were different batch sizes experimented with during the pre-training according to the different TPUv2 memory constraints. For the TPUv2-8, the batch size was 128. The TPUv2-256 Pod used both 256 and 4096 batch sizes according to the AdamW and LAMB optimizers. The loss function used was Cross-Entropy for the pre-training objectives as specified in the BERT Paper.

### 3.7.2 Vocabulary

BERT’s vocabulary file determines how it tokenizes input sentences. The size of the vocabulary file also dictates the size of its Word Embedding layer. The bert-vocab-builder [81] script was used to create the vocabulary file because the BERT authors did not release the code for creating it. The script uses the WordPiece tokenization algorithm from the tensor2tensor [82] library and formats it to BERT’s specification. Additional word counts of the svBERT pre-training text corpus were performed to further refine the vocabulary. The final vocabulary size was 39956.
Chapter 4

Results

The SVMs that used fastText and word2vec achieved identical accuracy, F1 scores, and correctly classified all the same emails (Figure 4.3). Because the cross validation score was higher for the SVM using fastText (Figure 4.1) this was considered to be the overall best SVM model. Similarly, different versions of BERT were optimized. The results of the four different BERT models showed that the model that was based on mBERT and pre-trained with the Parkman email text (mPreBERT) was the best performing of the four different BERT models and that mPreBERT delivered the better results compared to the best SVM model using fastText.

<table>
<thead>
<tr>
<th>Name</th>
<th>Model</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf-idf</td>
<td>SVM using tf-idf created word vectors</td>
</tr>
<tr>
<td>word2vec</td>
<td>SVM using word2vec embeddings</td>
</tr>
<tr>
<td>fastText</td>
<td>SVM using fastText Word Embeddings</td>
</tr>
<tr>
<td>mBERT</td>
<td>Multilingual BERT model</td>
</tr>
<tr>
<td>mPreBERT</td>
<td>Multilingual BERT with Pre-training</td>
</tr>
<tr>
<td>svBERT</td>
<td>Swedish BERT model</td>
</tr>
<tr>
<td>svLBERT</td>
<td>Swedish BERT with LAMB Optimizer</td>
</tr>
</tbody>
</table>

Table 4.1: A reference for the different abbreviations of the models presented in the Results.
4.1 Parkman Email Classification Task

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>F1</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>SVM &amp; fastText</td>
<td>86.98%</td>
<td>84.33%</td>
<td>187</td>
</tr>
<tr>
<td>mPreBERT</td>
<td>86.04%</td>
<td>85.16%</td>
<td>185</td>
</tr>
</tbody>
</table>

Table 4.2: Results for the best SVM and BERT model for Parkman email classification task.

In Table 4.2, the results for the best performing SVM and BERT model is shown. The accuracy and total number of correctly classified emails for the SVM with fastText were 86.98% and 187, respectively the accuracy and total number of correctly classified emails for the mPreBERT model 86.04% and 185. The SVM outperformed mPreBERT on these two metrics, however this was not the case with their respective F1 scores as mPreBERT outperformed the SVM by 85.16% to 84.33%. The F1 score is a common metric for NLP tasks as it gives an insight to the quality of misclassified answers and shows mPreBERT to be slightly better at the task of Parkman email classification than any SVM.

4.2 SVM

<table>
<thead>
<tr>
<th>SVM</th>
<th>Accuracy</th>
<th>F1</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf-idf</td>
<td>86.04%</td>
<td>84.03%</td>
<td>185</td>
</tr>
<tr>
<td>word2vec</td>
<td>86.98%</td>
<td>84.33%</td>
<td>187</td>
</tr>
<tr>
<td>fastText</td>
<td>86.98%</td>
<td>84.33%</td>
<td>187</td>
</tr>
</tbody>
</table>

Table 4.3: Best performance from the different SVM methods on the Parkman email classification task.
First, the SVM using tf-idf was applied to the raw text email text corpus from Parkman. It correctly predicted the class of each email with an accuracy of 86.04% with a total of 185 emails correctly classified, and a F1 score of 84.03%. Next, word2vec was used with the SVM with an accuracy of 86.98% with a total of 187 emails correctly classified, and a F1 score of 84.33%. Finally, fastText was used with the SVM with an accuracy of 86.98% with a total of 187 emails correctly classified, and a F1 score of 84.33%.

Since the word2vec and fastText scores were identical, Figure 4.1 plots the cross-validation scores for the 3 different SVM methods. It shows that fastText is the highest performing, followed by word2vec, and lastly tf-idf.

### 4.2.1 Text Filtering

<table>
<thead>
<tr>
<th>SVM</th>
<th>Accuracy</th>
<th>F1</th>
<th>Filtered</th>
</tr>
</thead>
<tbody>
<tr>
<td>tf-idf</td>
<td>75.14%</td>
<td>73.11%</td>
<td>False</td>
</tr>
<tr>
<td>tf-idf</td>
<td>86.04%</td>
<td>84.03%</td>
<td>True</td>
</tr>
</tbody>
</table>

Table 4.4: Increase in performance for SVM with text filtering.

First, the SVM using tf-idf classified the raw Parkman emails. This resulted in an accuracy of 75.14% and a F1 score of 73.11%. Next, the Parkman emails were filtered using the formatting program. Afterwards, the SVM tf-idf performed at an accuracy of 86.04% and a F1 score of 84.03%. This represents
a relative increase of 14.51% and a 10.9% total in accuracy, as well as a relative increase of 14.94% and a total increase of 10.92% in F1 score. Figures A.1 and A.2 demonstrate how the non filtered data affects the importance of words associated with the category.

![Confusion matrix](image)

(a) Raw Email Input Data

![Confusion matrix](image)

(b) Filtered Email Input Data

Figure 4.2: Confusion matrices for the SVM with tf-idf before and after the email text filtering for metadata. The biggest gain in performance was with the "Säga upp sitt avtal" category from 3 to 27 correctly classified emails out of a possible of 28 emails in that category.
4.2.2 word2vec and fastText

Figure 4.3: Confusion matrices for the SVM with word2vec and fastText were identical. The category that they both had the most difficult categorizing was "Beställa en hyrd plats via mail" which was 7 times mismisclassified as "Få mer info om möjliga parkeringsplatser".
4.3 mBERT

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
<th>F1</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>mBERT</td>
<td>79.53%</td>
<td>77.39%</td>
<td>171</td>
</tr>
<tr>
<td>mPreBERT</td>
<td>86.04%</td>
<td>85.16%</td>
<td>185</td>
</tr>
<tr>
<td>svBERT</td>
<td>29.30%</td>
<td>13.28%</td>
<td>63</td>
</tr>
</tbody>
</table>

Table 4.5: BERT results for Parkman email classification task.

The mBERT model without any modifications performed the email classification task to provide a baseline to compare various attempts at BERT optimization. The unmodified mBERT was able to achieve an accuracy of 79.53%, a F1 score of 77.39%, and a total of 171 emails classified correctly. The mPreBERT model that was optimized by pre-training on the unlabeled Parkman email corpus achieved an accuracy of 86.04%, a F1 of 85.16%, and a total of 185 emails correctly classified. Figures 4.4 and 4.5 depict reasonable training losses graphs for mPreBERT. The pre-training on the text corpus (mPreBERT) increased accuracy of the mBERT model by 8.19% and the F1 score by 10.34%. The svBERT model achieved an accuracy of 29.30%, a F1 of 13.28%, and a total of 185 emails correctly classified. It was unsurprising that the svBERT model performed so poorly. It had failed in learning to represent a general Swedish language model.
4.3.1 mPreBERT

Figure 4.4: Training loss for mPreBERT during pre-training of the Parkman email corpus. The hyper parameters used for training were a batch size of 32, a sequence length of 512, an $\eta$ of $3e^{-5}$, 5k warm up steps, 45k training steps, 15% tokens masked, and a maximum of 20 predictions per sequence.

Figure 4.5: Training loss for mPreBERT on the Parkman email classification task. The hyper parameters used for training were a batch size of 32, a sequence length of 128, an $\eta$ of $3e^{-5}$, and 10 epochs.
4.3.2 Added Vocabulary

<table>
<thead>
<tr>
<th>Model</th>
<th>Added Vocabulary</th>
<th>Accuracy</th>
<th>F1</th>
<th>Correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>mBERT</td>
<td>False</td>
<td>79.53%</td>
<td>77.39%</td>
<td>171</td>
</tr>
<tr>
<td>mBERT</td>
<td>True</td>
<td>5.12%</td>
<td>1.02%</td>
<td>11</td>
</tr>
<tr>
<td>mPreBERT</td>
<td>False</td>
<td>86.04%</td>
<td>85.16%</td>
<td>185</td>
</tr>
<tr>
<td>mPreBERT</td>
<td>True</td>
<td>85.11%</td>
<td>83.15%</td>
<td>183</td>
</tr>
</tbody>
</table>

Table 4.6: mBERT performance for added Parkman vocabulary.

These results show that adding domain-specific vocabulary without additional pre-training on the domain-specific text corpus dramatically reduces performance. mBERT with the additional vocabulary only scored a 5.12% accuracy and a 1.02% F1 score. However, the results were promising for this technique of using mPreBERT with the added vocabulary. It achieved an accuracy of 85.11% and an F1 score of 83.15%, which is just under the SVM with tf-idf. This shows that the randomly initialized weights that are in place of the new words were able to be learned to some degree.
Figure 4.6: Confusion matrices for Parkman email classification for mPreBERT with and without added domain specific vocabulary.
4.4 svBERT

The overall results for svBERT were underwhelming. Its performance on the Parkman task was 63.16% below the mBERT’s baseline score and on the svNLI task was 25.11% below the mBERT’s baseline score. The only interesting result was that of the use of the LAMB optimizer in place of the AdamW optimizer which showed a dramatic reduction, about 7 times less, in pre-training time.

4.4.1 Pre-training

![Image of loss graph]

Figure 4.7: Because of the nature of the preemptible [83] Google Cloud and TPU instances, pre-training would be interrupted regularly. This meant that loss graphs like this once were discarded each time this happened. However, the graph depicts what was typical in the pre-training loss plot. It would start at about 12 and never stay below 5. The BERT code repository suggests that the final loss should be below 1.
4.4.2 Optimizers

<table>
<thead>
<tr>
<th>svBERT</th>
<th>Optimizer</th>
<th>Steps</th>
<th>Batch</th>
<th>svNLI Score</th>
<th>Hours</th>
</tr>
</thead>
<tbody>
<tr>
<td>v2-8</td>
<td>AdamW</td>
<td>1000k</td>
<td>128</td>
<td>51.4%</td>
<td>121</td>
</tr>
<tr>
<td>v2-256</td>
<td>AdamW</td>
<td>1000k</td>
<td>256</td>
<td>33.3%</td>
<td>83</td>
</tr>
<tr>
<td>v2-256</td>
<td>LAMB</td>
<td>125k</td>
<td>4096</td>
<td>57.9%</td>
<td>12</td>
</tr>
</tbody>
</table>

Table 4.7: A comparison of using different optimizers on the TPU configurations on Google Cloud.

The pre-training using the AdamW Optimizer on the TPU v2-8 took the longest time, around 121 hours, because of the 64 GB memory limit could only allow for a batch size of 128, and achieved a svNLI score of 51.4%. The pre-training of the AdamW Optimizer on the TPU v2-256 reduced the training time 31.4% to 83 hours with an increased batch size of 256, but only scored a 33.3% score on the svNLI task, which is equivalent to random guessing.

In order to take advantage of LAMB, the batch size must be increased significantly which means using more memory, and that was only possible by using the TPU v2-256 Pod with 2 TB of memory. The batch size was increased 16 times and led to an 85.54% reduction in time spent training. An interesting side affect of using the larger batch size is that the model avoided getting stuck in a local minimum that is a common issue with Stochastic Gradient Descent and achieved an svNLI score of 57.9%.

4.4.3 svNLI Task

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>mBERT</td>
<td>77.3%</td>
</tr>
<tr>
<td>svBERT</td>
<td>33.3%</td>
</tr>
<tr>
<td>svLBERT</td>
<td>57.9%</td>
</tr>
</tbody>
</table>

Table 4.8: Performance of different BERT variations created by the author for this research on the svNLI task. The hyper parameters used were a batch size of 32, a sequence length of 128, an η of 5e⁻⁵, and 2 epochs.
mBERT vastly outperformed both svBERT and svLBERT on the svNLI task. It would appear that svBERT got stuck in a local minimum because of its relatively small batch size (256), while svLBERT avoided this problem because of its larger batch size (4096).

### 4.4.4 Pre-training on svNLI

<table>
<thead>
<tr>
<th>Model</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>mBERT</td>
<td>77.3%</td>
</tr>
<tr>
<td>mPreBERT</td>
<td>79.6%</td>
</tr>
<tr>
<td>svLBERT</td>
<td>57.9%</td>
</tr>
<tr>
<td>svPreLBERT</td>
<td>70.8%</td>
</tr>
</tbody>
</table>

Table 4.9: Performance of pre-training on the svNLI unlabeled corpus on the best performing svBERT and mBERT models. The hyper parameters used were a batch size of 32, a sequence length of 128, an $\eta$ of $5e^{-5}$, and 2 epochs.

The mPreBERT, svBERT, svLBERT, and standard mBERT models were all pre-trained on the raw svNLI text corpus. The pre-training hyper parameters used were a batch size of 32, a $\eta$ of $1e^{-4}$, 20 predictions per sequence, 5k warm up steps, and 50k training steps. Then all of the models evaluated the svNLI tasks. This resulted in a 2.98% increase in an accuracy, or a total accuracy of 79.6% for mPreBERT. This increase is marginal probably due to the fact that the svNLI "documents" only consisted of two short sentences, which is nothing compared to a full Wikipedia article. The svNLI greatly benefited from the pre-training with an increase of 22.28%, or a total accuracy of 70.8% for the svPreLBERT model.
4.4.5 svNLI Custom Vocabulary

Table 4.10: Adding to mBERT’s Vocabulary

<table>
<thead>
<tr>
<th>Vocab Size</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>119547</td>
<td>77.3%</td>
</tr>
<tr>
<td>157978</td>
<td>61.2%</td>
</tr>
</tbody>
</table>

Figure 4.8: Training Loss for mBERT for svNLI Task.

Figure 4.9 depicts the results of the experiment of expanding mBERT’s vocabulary by 38,431 of the most common words occurring in the svNLI corpus. This involved the custom modification of the Word Embedding layer of the mBERT model described in the methods section. This ultimately was detrimental to the performance with an accuracy of 61.2% much like the Parkman experiment because the additional words are simply initialized with random weights and haven’t had a chance to be learned from the pretraining process.
4.5 Mixed Precision

4.5.1 fp16

Table 4.11: mBERT svNLI Performance on fp16 and fp32

<table>
<thead>
<tr>
<th>fp</th>
<th>Batch</th>
<th>Sequence</th>
<th>RAM</th>
<th>Minutes</th>
<th>Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>16</td>
<td>128</td>
<td>64</td>
<td>7.7 GB</td>
<td>29</td>
<td>75.5%</td>
</tr>
<tr>
<td>16</td>
<td>64</td>
<td>128</td>
<td>8.2 GB</td>
<td>59</td>
<td>76.5%</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>128</td>
<td>5.7 GB</td>
<td>67</td>
<td>76.9%</td>
</tr>
<tr>
<td>32</td>
<td>32</td>
<td>128</td>
<td>8.1 GB</td>
<td>86</td>
<td>77.3%</td>
</tr>
<tr>
<td>16</td>
<td>32</td>
<td>256</td>
<td>9.2 GB</td>
<td>125</td>
<td>76.2%</td>
</tr>
</tbody>
</table>

Using half precision fp16 provided by NVIDIA’s apex package [35] instead of the regular fp32 precision did not increase accuracy of mBERT’s score on the svNLI task, but it is able to significantly reduce training time with marginal sacrifices in performance. The fastest time of 29 minutes was achieved with a batch size of 128 and a sequence length of 64 that resulted in a 75.5% accuracy which is only a 2.33% decrease in performance compared to the baseline fp32 mBERT accuracy of 77.3% and a 66.28% reduced task time. The loss of precision is apparent with the fp16 run with batch size of 32 and sequence length of 256 because it consumes over 13.58% memory, but still scores lower than the baseline fp32 mBERT model.
Chapter 5
Discussion

This thesis demonstrates that a deep learning NLP model can achieve an F1 score for a email classification task that is better than a classical machine learning model. This result attained by pre-training mBERT (mPreBERT) with a relatively small Swedish language domain-specific text corpus of only 3.67 million words. It is important because greater accuracy can be expected as the pre-training text corpus size increases. Other investigators have identified the need for the availability for a large, domain-specific text corpus. The BioBERT project from Lee et al. [84], which created a BERT model for biomedical text mining, also took this approach with great success. At the present time, the availability of a sufficiently large, modern Swedish language text corpus is a problem.

The result is also important because it confirms an idea proposed by Howard and Ruder [85] in their work on Universal Language Model Fine-tuning for Text Classification (ULMFiT). Their idea was that pre-training on a target task’s text corpus would lead to improved accuracy despite a small text corpus size. The Parkman Email text corpus was just that; the results show that it did in fact lead to markedly improved accuracy with a corpus size of only 3.67 million words.

It was also demonstrated that improved performance of mPreBERT is unable to be achieved with the addition of domain-specific vocabulary related to Parkman’s business terminology. In fact, the additional domain-specific vocabulary was seen to decrease performance by a small amount in this investigation. A reason for this is that each word in the vocabulary file corresponds to a single weight in the Word Embedding layer in BERT’s architecture. This lack of improvement in performance was probably due to insufficient training data to learn the proper weights associated with the added vocabulary words.
The classical machine learning approach of using a SVM with tf-idf was almost as accurate as mPreBERT. The hybrid SVM models with word2vec and fastText word embeddings further improved on this accuracy. While correctly classifying more emails than mPreBERT, the F1 performance of these two hybrids was less than mPreBERT. Nevertheless, the accuracy of mPreBERT and the accuracy of the hybrid SVM models are close enough for a relatively simple task such as the Parkman Email classification that it would be more cost effective to use the SVM solutions because they lack the specialized hardware requirements needed for running a BERT model.

To explore the question of whether BERT could provide an even more accurate classification of the Parkman emails, an attempt was made to pre-train a BERT model from scratch solely on a Swedish language text corpus (svBERT). It was hoped that svBERT would learn a better representative language model of Swedish. The vocabulary file of svBERT was a key factor in its learning ability. It not only dictates how words are tokenized but also the size of Word Embedding layer. In order to create this optimized vocabulary for svBERT, an in depth investigation into the work of Wu et al. [27] would have been required. Instead, an off-the-shelf vocabulary file creation script was used for svBERT’s vocabulary file. In retrospect, the script failed to properly split Swedish words, and this negatively affected its ability to reconstruct out of vocabulary words. This can be seen in Figure A.3. It depicts a 3-D PCA chart of the Word Embedding layer weights of both svBERT and mBERT. The mBERT chart shows a spherical and almost uniform distribution of weights in the vector space, while the svBERT chart only has smaller evenly distributed shape with two other highly concentrated areas far apart from it. This offers insight as to why svBERT was unable to differentiate the words from its text corpus dataset. Figure A.4 further illustrates this fact with two t-SNE representations of the learned weights of svBERT Word Embedding layer. Both of the graphs depict highly clustered areas, suggesting that many of the words were found to be similar. This attempt to create svBERT demonstrated that the size of the pre-training text corpus was insufficient to learn a proper representation of the Swedish language.

The svNLI task was important in the evaluation of svBERT’s Swedish language modeling ability because if svBERT had outperformed any other models on the Parkman Email Classification task, then it would be difficult to confidently conclude whether or not this was due to the fact that it had a better understanding of Swedish than mBERT. The task evaluates BERT’s Next Sentence Prediction training objective performance because it is a Sentence Pair Classification task. This means that svBERT had to make use of the
Segment Embeddings layer in BERT’s Word Embeddings, which is irrelevant in a single sentence classification like the Parkman email task. However, svBERT could not compete with mBERT’s performance on either the Parkman email classification nor the svNLI tasks.

A surprising finding of this thesis was the training time reduction by using the LAMB optimizer instead of the AdamW optimizer during the pre-training of svBERT. This reduced the pre-training time from days to hours. As mentioned previously, the LAMB paper is under scientific review, but even if it is not found to be a replacement for the AdamW optimizer it can be used as an effective prototyping method to develop new deep learning models quickly.

Other approaches to Swedish language NLP tasks might be considered as follows. It is interesting that mBERT could be so effective at a Swedish NLP task considering it was just one of a hundred languages that it was trained on. One explanation is that Swedish is a Germanic language. This is relevant because all of Germanic languages have similar sentence structure, grammar, and words. Of the top five largest Wikipedia dumps used in pre-training for mBERT, three are Germanic and account for 61.65% of the articles in the top five largest. In fact, Swedish is the third largest Wikipedia dump [86]. Therefore, consideration might well be given to whether it would be more effective to create a Scandinavian BERT (ScandicBERT) that is trained on Swedish, Danish, Norwegian, and Icelandic. The additional Scandinavian languages would boost the article count by 35.12% [86]. Between those five languages it would be possible to construct a considerably large text corpus besides their Wikipedia dumps that would be much larger than the one used in this research. It would also be interesting to attempt to create a BERT-large (24 Encoders and 16 Attention heads) version of svBERT given enough TPU resources. Another radical approach to Swedish NLP tasks would be to simply translate the Swedish text into English text and then use the English BERT for the task. Flawlessly effective Swedish to English translation would forgo the need for svBERT or even mBERT, and would instead guarantee the cutting edge performance of BERT-base on Swedish NLP tasks.
Chapter 6
Conclusions

While BERT seemed like a compelling solution to the Parkman email classification task, it was first and foremost not a practical one. The cost of running a dedicated GPU server for the BERT model is too expensive month over month and not something in the client’s budget. The goal of truly automating a response system like this also requires a near perfect accuracy which none of the BERT models could offer given the amount of pre-training. However, exploring a Swedish language version of BERT is very worthwhile if an appropriate size Swedish language text corpus for pre-training can be made available. Thus, using a classical machine learning method like a SVM is sufficient for this task.

The need for Swedish language BERT (svBERT) is suggested by this research in order to obviate the need for countless NLP task specific solutions by classic machine learning programs. A sufficiently large Swedish text corpus can expected to make it possible for an svBERT to achieve state of the art results that is demonstrated with the English language BERT models on English NLP tasks. A well trained deep learning language model should be a national and strategic goal for the country.
Bibliography


[37] Cloud Storage Documentation | Cloud Storage. URL: https://cloud.google.com/storage/docs/ (visited on 01/14/2019).

[38] Google Compute Engine Documentation | Compute Engine. URL: https://cloud.google.com/compute/docs/ (visited on 01/14/2019).


[40] Debian – The Universal Operating System. URL: https://www.debian.org/ (visited on 12/05/2019).


[58] PostgreSQL | The world’s most advanced open source database. url: https://www.postgresql.org (visited on 01/29/2019).


[62] spaCy · Industrial-strength Natural Language Processing in Python. URL: https://spacy.io/ (visited on 12/05/2019).


[66] 8 Sidor - Lättlästa nyheter. URL: https://8sidor.se/ (visited on 08/12/2019).

[67] Aftonbladet: De senaste nyheterna, Sveriges största nyhetssajt. URL: https://www.aftonbladet.se/ (visited on 08/12/2019).


[76] *Scale your pandas workflow by changing a single line of code. — Modin 0.6.3 documentation*. URL: https://modin.readthedocs.io/en/latest/# (visited on 12/05/2019).


Appendix A

Model Visualizations

A.1 Importance Vectors

Importance Features for SVM tf-idf without filtering

<table>
<thead>
<tr>
<th>Feature</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Beställa en hyrd plats via mail</td>
<td>1.3035 vb parkering</td>
</tr>
<tr>
<td>Uppsågning</td>
<td>0.9257 fw parkering</td>
</tr>
<tr>
<td>Kvitto</td>
<td>0.3103 garageplats</td>
</tr>
<tr>
<td>Uppsågning av</td>
<td>0.8982 vb parkeringsplats</td>
</tr>
<tr>
<td>Allégaraget ursprungligt</td>
<td>0.7818 garageplats ursprungligt</td>
</tr>
<tr>
<td>Allégaraget</td>
<td>0.7234 parkering</td>
</tr>
<tr>
<td>Lördag</td>
<td>0.6674 Åkersberga centrum</td>
</tr>
<tr>
<td>På lördag</td>
<td>0.6674 personalparkering Åkersberga</td>
</tr>
<tr>
<td>Parkering på</td>
<td>0.6323 parkeringsplats ursprungligt</td>
</tr>
<tr>
<td>Vb uppsågning</td>
<td>0.6248 Åkersberga</td>
</tr>
<tr>
<td>Vb kvitto</td>
<td>0.6076 personalparkering</td>
</tr>
<tr>
<td>Allégaraget</td>
<td>0.5841 centrum</td>
</tr>
<tr>
<td>Fw faktura</td>
<td>0.5824 fw</td>
</tr>
<tr>
<td>Faktura</td>
<td>0.5404 marievik ursprungligt</td>
</tr>
<tr>
<td>Parkering odenplan</td>
<td>0.5367 hyra</td>
</tr>
</tbody>
</table>

Importance Features for SVM tf-idf without filtering

<table>
<thead>
<tr>
<th>Feature</th>
<th>Importance</th>
</tr>
</thead>
<tbody>
<tr>
<td>Kvitto</td>
<td>1.0574 vad</td>
</tr>
<tr>
<td>Vb</td>
<td>0.9863 odenplan</td>
</tr>
<tr>
<td>Den</td>
<td>0.9530 platsen</td>
</tr>
<tr>
<td>Från</td>
<td>0.8673 hur</td>
</tr>
<tr>
<td>Uppsågning</td>
<td>0.8228 undrar</td>
</tr>
</tbody>
</table>

Figure A.1: SVM with tf-idf without filtered of email input data.
Figure A.2: SVM with tf-idf with filtered email input data.

A.2 mBERT Word Embeddings

Figure A.3: 3-D PCA Visualization of Word Embeddings Layer
Figure A.4: svBERT t-SNE Graphs of Word Embedding Layer