University of Padova

Department of Mathematics "Tullio Levi-Civita"

Master Thesis in Data Science

Overview of the Multi-Task Mutual Learning Technique: A Comparative Analysis of Different Models for Sentiment Analysis and Topic Detection

Supervisor
Prof. Tomaso Erseghe
University of Padova

Co-supervisor

Master Candidate
Matteo Posenato

Student ID
1228001

Academic Year
2022-2023

Abstract

This research aims to provide a clearer overview of a new technique called Multi-task Mutual Learning in the field of Natural Language Processing, specifically in sentiment analysis and topic detection. The objective is to understand whether employing different models within this technique may impact its performance. With the growing collection of natural language-based data, private companies, public organizations, and various entities are increasingly seeking to extract information from this vast amount of data, which can be in the form of audio, text, or video. This underscores the need to study systems that can analyze this data effectively and do so in the shortest possible time, providing a competitive advantage in the private sector and a social analysis of the current historical moment in the public domain. The method employed is Mutual Learning, and within this technique, we analyzed specific models, including Variational Autoencoder, Dirichlet Variational Autoencoder, Recurrent Neural Network, and Bidirectional Encoder Representation from Transformer. These methods were executed with two datasets: YELP, containing reviews of commercial activities, and IMDB, containing reviews of films. The main findings highlight the complexity of the model, the computational power required, and the customization of the model according to specific needs.

Contents

Listing of figures

Listing of tables

Listing of acronyms

NN . . . . . Neural Network

CNN . . . . . Convolutional Neural Network

RNN . . . . . Recurrent Neural Network

NLP . . . . . Natural Language Processing

MTL . . . . . Multi-task learning

AI . . . . . Artificial Intelligence

DNN . . . . . Deep Neural Network

KLD . . . . . Kullback–Leibler Divergence

VAE . . . . . Variational Autoencoder

MLP . . . . . Multilayer Perceptron

BERT . . . . . Bidirectional Encoder Representations from Transformers

MLM . . . . . Masked Language Model

ZB . . . . . Zettabyte

NLU . . . . . Natural Language Understanding

IR . . . . . Information retrieval

HMM . . . . . Hidden Markov Model

LDA . . . . . Latent Dirichlet Allocation

PCA . . . . . Principal Component Analysis

PDF . . . . . Probability Distribution Function

DirVAE . . . . . Dirichlet Variational Autoencoder

1

Introduction

In today's digital age, the proliferation of social networking sites and communication devices, such as smartphones, laptops, and PCs, has facilitated unprecedented levels of interaction among individuals, leading to the creation of massive amounts of big data. Notably, platforms like Twitter boast a vast network of 467 million users, generating a staggering 175 million tweets daily [2]. The sheer volume of data generated is astounding, where storing one second of high-definition video requires 2000 times more space than a page of plain text.

The International Data Corporation's 2011 report revealed that the world had already generated approximately 1 zettabyte (ZB) of data, and this exponential growth continued, reaching 7ZB by the end of 2014. Projections indicate that by 2020, the volume of data generated is expected to skyrocket to 44ZB, with textual data from social media technologies like Facebook, Twitter, and messaging apps such as WhatsApp and Telegram constituting at least half of this data.

Amid this data deluge, our focus turns to Natural Language Processing (NLP), a field that plays a pivotal role in making sense of the textual data inundating digital platforms. This chapter delves into the intricacies of NLP, exploring topics such as Topic Detection, Sentiment Analysis, Mutual Learning, and related works. As we navigate through these aspects, it becomes evident that the unprecedented growth in data, as outlined above, underscores the significance of advanced computational techniques in extracting meaningful insights from the textual sea of information.

1.1 Natural Language Processing

Natural Language Processing (NLP)[3] is a field of study that encompasses a range of computational techniques aimed at analyzing and representing texts found in natural language. These techniques operate at various levels of linguistic analysis and are designed to achieve language processing capabilities resembling those of humans. Importantly, NLP is not an end goal in itself but serves as a means to accomplish specific tasks or applications.

The primary objective of NLP is to achieve human-like language processing, where the term "processing" is carefully chosen over "understanding." While the field was initially called Natural Language Understanding (NLU), it is now widely accepted that complete NLU has not yet been achieved. A comprehensive NLU system would be able to paraphrase text, translate it into different languages, answer questions about its content, and draw inferences from the text. Although NLP has made progress in tasks such as paraphrasing, translation, and question-answering, the ability to draw inferences remains a goal.

NLP has practical applications tailored to specific needs, such as Information Retrieval (IR) systems that use NLP for providing precise and comprehensive information in response to user queries. In these systems, the goal is to represent the user's query accurately and match it with the content of documents, regardless of how the query is expressed.

The origins of NLP can be traced to various disciplines, including linguistics, computer science, and cognitive psychology. Linguistics contributes formal, structural models of language, computer science focuses on developing internal representations and efficient processing, while cognitive psychology examines language usage as a window into human cognitive processes.

The field of NLP is often divided into language processing and language generation. Language processing involves analyzing language to produce meaningful representations, while language generation focuses on producing language from a representation. Additionally, a traditional distinction is made between language understanding and speech understanding, with speech understanding incorporating elements of acoustics and phonology.

NLP approaches can be categorized into symbolic, statistical, connectionist, and hybrid methods. The statistical approach, for instance, employs mathematical techniques and large text corpora to create generalized models of linguistic phenomena. Hidden Markov Models (HMM) are frequently used in statistical models, particularly in tasks such as speech recognition, lexical acquisition, parsing, part-of-speech tagging, and machine translation. To do that its can be used differences types of Neural probabilistic language model [4], that have the goal of learn the joint probability functions of sequences of words in a language. This kind of models

have an intrinsically one problem: the dimensionality, in our work we encounter that problem because the model that we used have to manage a big number of data and a large size of input senteces that produce an high computational costs. A statistical model of language can be represented by the conditional probability of the next word given all the previous ones, since

$$\hat{P}(w_1^T) = \prod_{t=1}^{T} \hat{P}(w_t | w_1^{t-1})$$

where w_t is the t-th word, and writing sub-sequence $w_i^j = (w_i, w_{i+1}, ..., w_{j-1}, w_j)$. When building statistical models of natural language, one considerably reduces the difficulty of this modeling problem by taking advantage of word order, and the fact that temporally closer words in the word sequence are statistically more dependent. Thus, n-gram models construct tables of conditional probabilities for the next word, for each one of a large number of contexts, i.e. combinations of the last n−1 words:

$$\hat{P}(w_t | w_1^{t-1} \leftarrow \hat{P}(w_t | w_{t-n+1}^{t-1})$$

We only consider those combinations of successive words that actually occur in the training corpus, or that occur frequently enough. In our work as specified above one of the biggest issues is managing the dimensionality of our datasets we use the approach explained in the paper [4] citate before that can be summarize in three points:

- Associate with each word in the vocabulary a distributed word feature vector (a realvalued vector in R^n)
- Express the joint probability function of word sequences in terms of the feature vectors of these words in the sequence
- learn simultaneously the word feature vectors and the parameters of that probability function

The feature vector represents different aspects of the word: each word is associated with a point in a vector space. The probability function is expressed as a product of conditional probabilities of the next word given the previous ones.

1.2 Topic Detection

Topic detection uses the NLPs algorithms in order to understand the topic behind textual data. this common task is particularly usefull when you can have access to a large dataset but you know that a lots of data are useless of you needing, so that you extract the topic behind automatically. The research in this field is really important because in the recent year we can have access to an infinity amount of text data through for example the social network, that can be used for a bunch of scope such as Marketing for the private company, if you can analyze as much as possible text data we can reply rapidly to the customers question not only in the review but principle in the products adapting them to the needs of large consumer groups. For the public company can be usefull to understand the quality of the services or if needing others services to help the citizens in their life. The singular states can be understand what social issues there are in their country, nowadays the majority of people don't go out to protest but write a post on Twitter or other social network, due to that the government can be adapt their policies or they could be solve some problem that in normal scenario cannot because they didn't reach that information. Another big field in which topic detection is really important is the human-computer interactions, in recent year the expansions of the AIs needs to search the best model to understand the human language through computers, to do so the topic detection represent one good information for the model that need to understand what they talk about with a speed process. The model related to this task can be applied at different levels, first it can be used at document-level to understand and analyse the topic in a document, this document is a sum of different sentences; another level is the sentence-level in which we analyse the sentences singularly in a document, or the deepest level is word-level in which the single words are analysed singularly. The choice of the level is based on you scope. in our works we analyse our datasets based on document level, in which every document is a sum of some sentences, because the word-level information is passed by the classification parts. Probabilistic topic models have been used widely in NLP. [5]Typically, words are assumed to be generated from latent topics which can be inferred from data based on word co-occurrence patterns. The basic idea is to construct a neural network which aims to approximate the topic-word distribution in probabilistic topic models. Additional constraints, such as incorporating prior distribution[6], enforcing diversity among topics [7] or encouraging topic sparsity [8], have been explored for neural topic model learning and proved effective. However, most of these algorithms take the auto-encoder as the basis learner to fit the distribution function. Due to the drawback in integrals, the generalization ability of the auto-encoder is limited. More recently, the Variational Auto-Encoder

[9] has been proved more effective and efficient to approximating deep, complex and underestimated variance in integrals [10], [11]. However, existing supervised neural topic models treat the class labels as weights which are distributed across words during training. It thus ignores the rich contextual information such as dependencies among words/phrases in a sentence and the ordering of sentences in a document, which is important for many downstream NLP tasks including sentiment analysis and opinion mining.

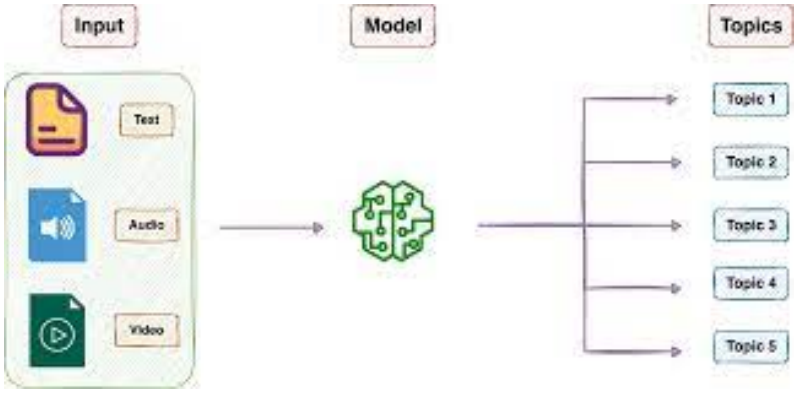


Figure 1.1: Topic Detection

1.3 Sentiment Analysis

In today's era of big data, the application of Opinion Mining and Sentiment Analysis has proven to be a valuable method for categorizing opinions and assessing the overall public mood. These techniques have evolved over the years and found applications across various datasets and experimental settings[12].

Research in sentiment analysis has expanded beyond unimodal text-focused approaches to include multiple modalities such as speech and video. It addresses various Natural Language Processing (NLP) subtasks, including aspect extraction, subjectivity detection, named entity recognition, and sarcasm detection.

The primary goal of sentiment analysis is to extract meaningful insights from people's opinions, providing valuable information to both consumers and manufacturers. Typically, sentiment analysis involves a classification process, with three main levels: document-level, sentence-level, and aspect-level sentiment analysis[12]. The main sources of data for sentiment analysis are product reviews, especially from review sites. While sentiment analysis is commonly applied to product reviews, its applications extend to stock markets, news articles, and political debates

[13]. A comprehensive survey on sentiment analysis algorithms and applications provides an overview of recent advancements in the field. This survey categorizes various algorithms and their contributions to sentiment analysis techniques, including sentiment classification, feature selection, emotion detection, and transfer learning [13]. Due to the complexity of sentiment analysis, which involves underlying concepts and expressions in text, the process encompasses multiple tasks. The primary tasks include sentiment or opinion detection, which classifies text as objective or subjective based on adjective examination, and polarity classification, which classifies opinions into opposing sentiment polarities [14]. Sentiment analysis involves assessing if a document or sentence carries emotional sentiment, usually categorized as positive, neutral, or negative. This information finds application in diverse contexts, such as evaluating customer satisfaction, discerning emotions in social media posts, or enhancing internal company processes. In recent years, sentiment analysis algorithms have evolved to associate various emotions, including anger, happiness, sadness, urgency, and more. In our work, specifically in Topic Detection, we employ a document-level approach. In the classification part, we adopt a word-level approach, leveraging sentiment methods to capture sentiment information for each label in our dataset. This information is exchanged with the topic parts and vice versa [5]. Sentiment classification can be achieved through supervised statistical learning models[15], [16], [17], traditional feature-engineering-based models, or deep learning models, including Convolution Neural Network (CNN)[18] and Recurrent Neural Network (RNN) [19]. Document-level sentiment classification involves modeling the hierarchical semantic composition of a document using hierarchical models like the Gated Recurrent Neural Network[20] and the Hierarchical Attention Network[21]. Recent advancements, such as training neural networks with large-scale pre-trained word embeddings like BERT[22], have significantly improved text classification[23]. In our specific approach, we utilize RNN and BERT.

1.4 Mutual learning

Mutual Learning [5] is an approach based on multi-task learning in which two different models are trained simultaneously, this two models share knowledge and their prediction during the training phase. This leads to have better information in the prediction because the two models receives information that they wouldn't have been able to find on their own. The Multi task Learning is an important machine learning mechanism that improves the generalisation performance by learning a task together with other related tasks, it usually has a common layers which learns a shared representation across tasks, then stack several task-specific upper layers

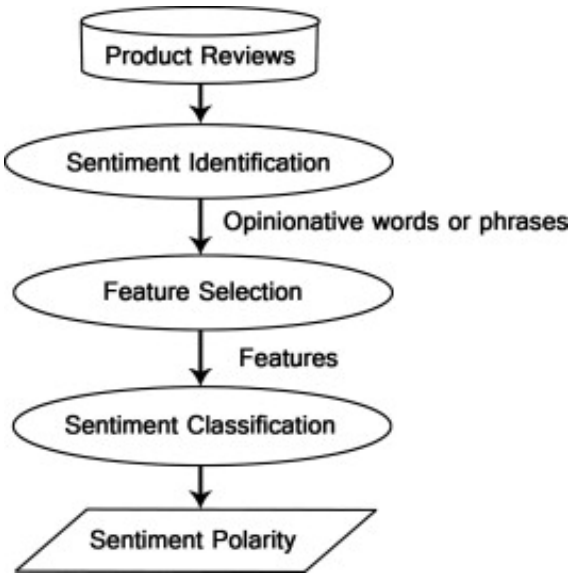


Figure 1.2: Sentiment analysis process

to learn task-relevant representations. MTL has been successfully applied in many NLP tasks including classification [24], [25] and sequence tagging [26], [27], [28], [29]. However, most MTL research focuses on supervised learning [30]. To the best of our knowledge, there is no MTL framework for jointly learning an unsupervised model such as a generative topic model and a supervised model such as a text classifier. The different from the MTL is that in the Mutual Learning are used both supervised and unsupervised models, the first one for the classification part and the second one for the part of topic detection. We use a novel MTL framework based on the following observation. The weights of the decoder in a neural topic model indicate the association probabilities between words and topics, while the attention signals in an attention-based classification model captures the importance of words/sentences contributing to the overall sentiment classification. Thus, if we could make these two distributions as similar as possible, we can potentially generate polarity-bearing topics and at the same time achieve higher sentiment classification results with the topical information incorporated. The key idea is to use latent topic distribution of each word obtained from the neural topic model to guide the calculation of word-level attention signals in the text classification model, which essentially incorporates the topic information into the classifier training. On the other hand, the word-level attention vector which potentially carries the word-level polarity information could be used to guide the learning of latent topic distributions in the neural topic model. The latent topic distribution for each word can be obtained by using the weights connecting the penul-

timate layer and the reconstruction layer in the neural topic model. The attention vector for each word in RNN is stored in u_i^j. Mutual learning is used to make the latent topic distribution of a word to be similar to the attention vector of the same word from RNN. The benefits of using such a strategy are:

- learning latent topics with word-level polarity information derived from classifier training without the need of using any external sentiment lexicons;
- incorporating the latent topic information into classifier training to improve the classification performance.

Below there is the pseudo-code of this model and the scheme of the algorithm which explains how the the Multi task mutual learning algorithm works in theory and the picture show how the two models exchange the information. The latent topic distribution for the ith word is

Algorithm 1.1 Multi-Task Mutual Learning

Require: Documents with labels $\}w_d, y_d\langle, d = \}1, 2, ..., D\langle$, pre-trained word embeddings, candidate word vocabulary $V = \}w_1^{'}, w_2^{'}, ..., w_{\|V\|}^{'}\langle$, the maximum training iterations T.
Ensure: Trained topic model and classifier
  Initialise model parameters
  for $j = 1$ to T or until convergence
    for each mini batch of training instances
      Minimise the loss function L_{final}
    end for
    for $t = 1$ to $\|\|V\|\|$
      Optimise the object function $\{$ for each $w_t^{'}$
    end for
  end for
  Fine tune the model by minimising the task specific loss function for each task

represented as $w_i^{'} = \}w_{i1}, w_{i2}, ..., w_{iK}\langle$ where K is the total number of topics, and the attention vector $u_i^{'}$ for the ith word is obtained from the scheme of RNN. We measure and maximise the similarity between the latent topic distribution of the ith word, $w_i^{'}$, and its attention vector $U_i^{'}$, during the training. We use the following similarity measurement metric:

$$O = \prod_i \frac{\|w_i^{'} \oplus u_i^{'}\|}{w_{i\ 2}^{'} \oplus u_{i\ 2}^{'}}$$

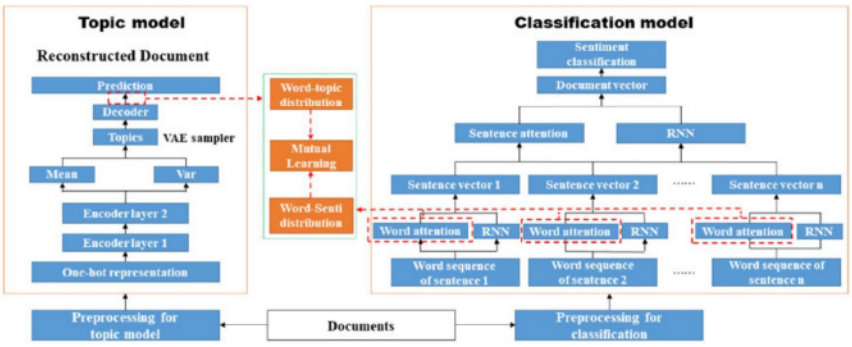


Fig. 4. Our proposed MTL with mutual learning for sentiment classification and topic detection.

Figure 1.3: MTL with Mutual Learning

that is based on cosine similarity. In order to make $w_i^{'}$ and $u_i^{'}$ comparable we set the dimension of the attention vector of $u_i^{'}$ to be the same as the number of latent topics. So as explained in line 4 and in line 7 we optimise the parameter of the equation:

$$L_{final} = \prod_{d} \alpha \oplus L_t(w_d) + \beta \oplus L_c(w_d)$$

where

$$L_c(w_d) = \prod p \oplus log[softmax(W_d \oplus w_d + b_d)]$$

$$L_t(w_d) \leftarrow \frac{1}{L} \prod_{l=1}^{L} \prod_{n=1}^{N_d} \log p(w_{d,n} || \hat{\beta}^{(l)}) \quad KL(q(z_d || w_d) \;\; p(z_d))$$

and the equation O separately. In each training epoch, we optimize the parameter in L_{final} iteratively for each minibatch of training instances to obtain topic and sentiment representations. Then, we update O at the end of current training epoch, so the L_{final} is update more frequently compared to O. This is the thery behind this medel, to have a complete vision about this techniques we modify the methods inside the model in order to see if the method exchange correctly the information and if there is a possibility to achieve better performance using model more sophisticate and newer in the literature. Basically we change the topic detection part from a Variational Autencoders(VAE) to a Dirichlet Variational autoencoders (DirVae), based on the results obtained using the LDA in sentiment and topic distribution [31], and in the sentiment analysis part from the Recurrent neaural network (RNN) to Bidirectional Encoder Represetnations from Tranformers (BERT). in the section 3. all this four model are explained

theoretically and our applications. Now we can explain how our code of MTL are composed: we have nine file:

- main.py: is one of the most important parts, in this file we set all the parameters for the model, preprocess the data for the topic part, we get vocabulary, load embedding,split the data in train and test, set the model, the loss function, the optimization function and last thing start training the model;
- train_model.py: this file contains all the stuff that we need in order to train the model and print the results. inside we have four function:
  - train_model: used to train the model inside we have a for loop based on the number of epochs, and a for loop based on the batch size. Inside this two loop we set all the variables that we need and we start to calculate all calling the model function, loss functions, once we calculate all we optimize it. under some specific conditions we evaluate it on the test set. after all we print the results;
  - evaluate: this function basically calculate all the things that are calculated in the training phase but in the test set. proceed always with a for loop based on the batch size;
  - get_reward_cv: this function create and return the classification vector based on the vocabulary the topic and the model;
  - print_topic: is a function used to print the resulting topic.

- model.py: is the most important part in this file are contained all the parts of the model, the first part start by setting all the parameters, the second one set the structure of the differences models. and the last one the part called forward part that are responsible to execute the code. all the functions are inside a class named JointModel;
- model_data.py: the main part in this file is the class DataIter, in which we modify the object in order to have iterable and modifiable the objects containes in the dataset and in all the others file;

- clf_data.py: here we have five functions:
  - get_vocabulary: used to create the vocabulary;
  - load_embedding: create the embedding matrix used in the model;
  - convert_words2ids: create the id list in order to complete the embedding;
  - load_clf: load the classification obtained by the sentiment analysis parts;
  - sort_key: organize the keys nad returns the document list, label list and the index list.

- topic_data.py: in this filed are contained two functions, the first one load and process the data in order to split ihe dataset in train and test, organize the label and all the stuff that are needed to preprocess the data, the second function is the preprocess of the data in order to obtain the correct structure and all the stuffs to train the model;
- embedding_code.py: cointains the embedding model
- loss.py: contains the loss function for the topic parts and the loss function shared by the two parts of the model which is used to optimize all the model;
- file_handing.py: is a library create by our own in which there are all the basic function to read, write, save the files.

This is how our files are organized, the singularly models are explained in section 3.

1.5 Objectives of this thesis

The aims of our work is basically two: the first one is see if the Mutual learning works better compared to the normal model of Topic detection, the second one is try to see if, changing the model we can achieve better performance, this for all the three vocabulary sizes (2000, 3000, 5000) To do so we compare all the model with some metrics:

- Complexity: Complexity in algorithms refers to the amount of resources (such as time or memory) required to solve a problem or perform a task. The most common measure of complexity is time complexity, which refers to the amount of time an algorithm takes to produce a result as a function of the size of the input. In our work we analyze the time complexity per epochs in the training phase. We do that to see the trade-off between the performance measures and the time because if we achieve a better accuracy but the improvement is very little and the time to run the new model is exponentially compared to the first one in the real scenario is not a good choices because we must take in consideration the time need to train the algorithm;

- Accuracy: the accuracy measure the number of the predicted values that are predicted correctly divided by the total predicted numbers. Accuracy has two definitions:

  - Is a description of only systematic errors, a measure of statistical bias of a given measure of central tendency; low accuracy causes a difference between a result and a true value; ISO calls this trueness.
  - ISO defines accuracy as describing a combination of both types of observational error (random and systematic), so high accuracy requires both high precision and high trueness.

  Accuracy in our work is defined as follow

$$(\prod correct\_prediction)/total\_sample$$

- KLD:[32] Kullback–Leibler divergence $KL(p, q)$ is the standard measure of error when we have a true probability distribution p which is approximate with probability distribution q. Its efficient computation is essential in many tasks, as in approximate computation or as a measure of error when learning a probability. in our work we use them to analyze the error in the generated distribution with respect to a normal distribution. We

define our KLD as follow

$$0.5 \oplus \prod_{i=1}^{N} \quad 1 + \log(\text{logvar}_i) \quad (\text{mean}_i^2) \quad \exp(\text{logvar}_i) \Big($$

This represent the loss for the topic detection part and is use it for calculate the loss

- Loss: [33]A loss function has two crucial roles in training a conventional discriminant deep neural network:

  - it measures the goodness of classification
  - generates the gradients that drive the training of the network.

  Conventional training of a DNN assumes a loss function that measures the "goodness" of the classification by comparing the prediction to the ground truth. Specifically, errors between the predicted and true labels are calculated over the training set. The errors are then combined into a scalar which is called loss. This phase of calculating the loss value from representation points is called forward propagation of the loss function. The training of the network actually occurs in the back propagation phase, in which the parameters of the network are updated proportionally to the gradient of the loss with respect to the parameters. As all the negative gradients are calculated by the chain rule that starts with the partial derivatives of the loss with respect to the representations, the derivatives of the loss function are the starting "forces" that drive the training of the network. We measure the loss by the cross entropy function [34] for the classification part and the KLD for the topic part. This two losses are multiplied by their weigths (two parameters that can be tuned if we want to give more importance to a part respect to the other) and then are summed up to compute the overall loss. Cross-entropy coincides with the (multinomial) logistic loss applied to the outputs of a neural network, when the softmax is used. It is known that the logistic loss is Bayes consistent [35] Thus, asymptotically, a nearly optimal minimizer of the logistic loss over the family of all measurable functions is also a nearly optimal optimizer of the zero-one classification loss. The Cross-entropy

are calculated as follow:

$$l(x, y) = L = \}l_1, ..., l_N\langle^T$$

$$l_n = \left(\prod_{c=1}^{C} w_c \log \right) \frac{\exp(x_{n,c})}{\prod_{i=1}^{C} \exp(x_{n,i})} \sum y_{n,c}$$

1.6 Related work

We start to work on the partial code that the researchers shared in the paper, the first part we try to recreate the missing part, and debug the others. The first missing part that we build-up is a library of basic function that read, write and save all the files that we use, we call it file_handling. One of the most important missing part was the parts of the embedding, we use a pre-built methods called "words2Vec" imported form gensim library: Word2Vec is a widely used algorithm based on neural networks, commonly referred to as "deep learning" (though word2vec itself is rather shallow). Using large amounts of annotated plain text, word2vec learns relationships between words automatically. The output are vectors, one vector per word, with remarkable linear relationships, that file is used in the function "load_embedding" to obtain the embedding matrix. Understanding the code we notice that in some part there is a theoretically problem so we fix that, for example the loss function, the researcher give some possibilities and studied all the theoretical implication for all the functions to see what works better, we make some correction to adapt to our modifications. We discover some problems in the structure of the code and in the flows of the algorithm, this work was pretty long because we cannot modified a lot of things in parallel so we use a partial dataset to see every modification how impact in the rest of the code. although we used a much smaller dataset each execution took a lot of time, we are talking about 50 minutes or even an hour. the changes that were made were many, once we reached the point where the code was running correctly we had to re-code all the data structures so that they would work with both datasets which have very different structures and sizes. First of all we start to analize the time consuming and we notice that the classification part is the most complex and is the part we required the majority of the time, Another time consuming part is the optimization in the training phase: the optimisation is act through the gradient descent in every batch, this could be really expensive when the batch size grows, we do that by PyTorch package that have pre-build functions to optimize the model, in our case

we use clip_gradient_norm that calculate the gradient of the loss and then calculate the norm on all the gradients together and modified the gradients in-place. At every iteration the model are optimize by the loss defined in the paragraph [1.5], and one time per batch we optimize the method with a KLD loss function but calculated with the word_attention_dictionary via this formula:

$$kld\_all+ = \frac{\| \rangle TM_{\text{attention}}, HAN_{\text{attention}} | \|}{HAN_{\text{attention}}{}^{2} \oplus TM_{\text{attention}}{}^{2}}$$

this calculated for all the instances in the word_attention_dictionary. where $HAN_{\text{attention}}$ is the value of attention and $TM_{\text{attention}}$ is the related weight in the model. When we understand that we are in fronts of a really complex method and it required hours to train it, we try to manage the different parameters in order to find the best combination that could make feasible the training in a good time. The parameters that can be modified are:

- Batch_size
- Vocabulary_size
- dt: can see as the number of topics we would predict
- tm_weight:
- clf_weight
- max_epochs
- min_epochs

For our first scope we concentrate two three parameters Batch_size, max_epochs and min_epochs. We try to tunes this values in the small subset of the data doing that for understand the behaviour and then we try in the whole datasets. We try to change the batch size parameter (1, 5, 10, 20, 100, 500, 1000) and see how much time required one epochs that is the sum of all iterations in the different batches and the overall time. We set a max number of epochs equal to 3 and a min equal to 1. Some values are not possibles for example with batch size equal to 1000 the computed haven't enough memory to analyze a batch with this dimensions. This take a lot of times because every run required hours to did it and the possibility to tuning this parameter are very much. Once we had found the best parameter for batch_size that would

allow us to execute the entire dataset in the shortest possible time, we focused on seeing how performance measures are affected with respect to vocabulary size. Once we had analysed all the results, which are reported in Chapter 4, we were able to understand what the behaviour of the model was like. Once we had completed the analysis of the basic model, which we will obviously take as standard, we started working on the first variation. We started to study theoretically how BERT's model worked for the sentiment analysis part, to see how we could adapt it to our standard model, making sure that the model would work with our data first, we created ad hoc functions to create the correct data structures to pass on to the model during execution, and then we tried to analyse what information was being exchanged between the two methods so that we could modify it to achieve our purpose. Having arrived at the point where the model with the BERT method worked, we selected the parameters from our standard model so that we could compare them, again the results obtained can be found in Chapter 4. Having performed the analysis of the MTL model with BERT, we moved on to the introduction of the second variant, the one using a Dirichlet Variational autoencoder for the topic detection part. The procedure was very similar to that used for the implementation of the BERT model: to study and understand the model thoroughly, to find a way to apply it correctly according to our data, to check the correct functioning in the exchange of information with the sentiment analysis part and finally to analyse the behaviour of the model with predefined parameters so that we could compare it with our basic model and finally to get an overview of the Multi task mutual learning method.

2

Dataset

We use two datasets the first one is a subset of the YELP dataset, which contains businesses reviews, contains the text of the reviews and the class associated. The class goes from 1 to 5, have 39 000 instances in the training set and around 16000 in the test set. The second dataset is the IMDB that contains movie's reviews, this dataset have ten classes from 1 to 10, contains 15000 instances in the training set and 9112 in the test set.

	#Class	#docs	Vocab.Size
YELP	5	39,923	53,823
IMDB	10	15,000	55,819

Table 2.1: Datasets

3 Models

3.1 Variational Autoencoder

Autoencoders are neural network architectures used for dimensionality reduction. They consist of two main components: an encoder and a decoder. The encoder takes the original features or data and produces a compressed representation, often referred to as "new features." The decoder then tries to reconstruct the original data from this compressed representation. This process is optimized iteratively through backpropagation, where the error between the encoded-decoded output and the initial data is used to update the network's weights.

The idea behind autoencoders is to create a bottleneck for data, allowing only the essential structured information to pass through for reconstruction. The architecture of the encoder and decoder networks defines the families of encoders (E) and decoders (D), respectively. The goal is to find encoder and decoder parameters that minimize the reconstruction error, typically achieved through gradient descent.

In the context of linear autoencoders with a single layer and no non-linearity, there's a clear connection to Principal Component Analysis (PCA). Both methods seek the best linear subspace for projecting data with minimal information loss. However, linear autoencoders can have multiple solutions, unlike PCA. Furthermore, they don't require the new features to be independent.

When both the encoder and decoder are deep and non-linear, a higher dimensionality re-

duction is possible while maintaining a low reconstruction loss. The more complex the architecture, the greater the compression potential. In theory, an encoder with infinite capacity could reduce any initial dimensionality to one, but this can lead to a lack of interpretable and exploitable structures in the latent space. The dimension of the latent space and the depth of autoencoders should be carefully controlled based on the specific dimensionality reduction objectives.

Variational autoencoders (VAEs) are a type of autoencoder that introduces regularization during training to ensure a regular and interpretable latent space. VAEs encode inputs as distributions over the latent space rather than single points. The training process involves encoding an input as a distribution, sampling a point from this distribution, decoding the point, and backpropagating the reconstruction error. The encoded distributions are typically set to be normal, with the encoder returning both the mean and covariance matrix.

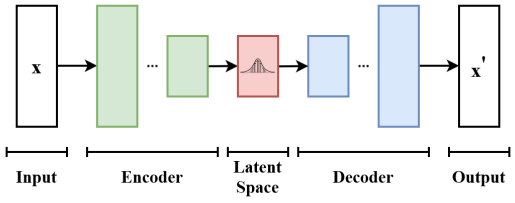


Figure 3.1: Variational Autoencoders

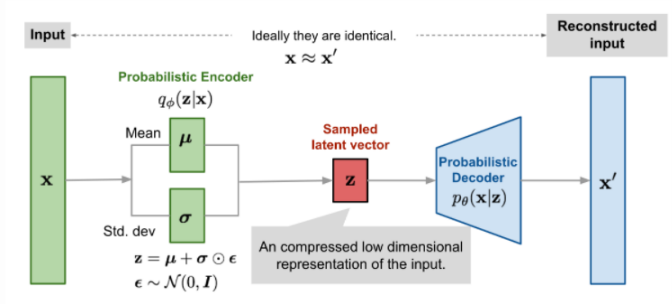


Figure 3.2: VAE

The regularity in the latent space, which is essential for generative purposes, is achieved through continuity and completeness. Continuity ensures that nearby points in the latent space result in similar decoded content, while completeness means that sampled points from the latent space yield meaningful content. VAEs address this regularity by enforcing the dis-

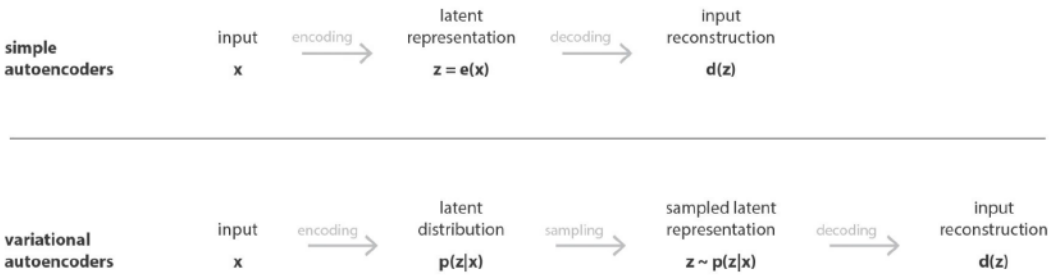


Figure 3.3: Autoencoders vs Variational Autoencoders

tributions to be close to a standard normal distribution, preventing punctual distributions or widely separated means.

The regularisation term in the VAE's loss function, represented by the Kulback-Leibler divergence, encourages the encoded distributions to meet these regularity conditions. This comes at the cost of a higher reconstruction error on the training data, and the trade-off between reconstruction error and the KL divergence can be adjusted based on specific requirements.

We review the topic detection model of [5]. For the generative model so the generation network we consider a corpus of D documents using a vocabulary of W words. Each document is represented by a (variable length) vector d_i, $i = 1, \ldots, D$ collecting the words occurrences in the document, so that $d_{i,n}$ $\mathcal{O}1, \ldots, W$. We let the corpus be organised in T topics, and denote with t_i (vector of length T) the latent topic representation of document d_i. Our reference generative model starts from an hidden prior variable z $\mathcal{O}$ R^T normally distributed, i.e., with probability distribution function (PDF) $p(z) = p_N(z; 0, I)$ where

$$p_N(x; m, \prod) = \frac{1}{\sqrt{det(2\pi \prod)}} e^{-\frac{1}{2}(x-m)^T \sum^{-1}(x-m)}$$

is the multivariate normal PDF. The latent topic representation t $\mathcal{O}$ R^T is approximated by a multilayer perceptron (MLP), to build a differentiable map of the form

$$t = \vartheta(z) = W_2 \tanh(W_1 z + b_1) + b_2$$

. The word-occurrence-pattern vector is then generated via softmax construction from the

latent topic representation, that is

$$\log(p_{d|t}(d|t) = \prod_n \log(s_{d_n}), s = softmax(W_3 t + b_3)$$

where s_{d_n} denotes the d_nth entry of s. Hence, we have

$$p_\theta(d|z) = p_{d|t}(d|\vartheta(z))$$

with parameters $\theta = \}b_1, b_2, b_3, W_1, W_2, W_3\langle$. For the approximation of posterior probability, so the inference network, we proceed as follows: the posterior probability $q_\phi(z|d)$ is approximated by the multivariate normal distribution

$$q_\phi(z|d) = p_N(z; \mu_\phi(d), diag(\sigma^2_\phi(d)))$$

where μ_ϕ and σ^2_ϕ are differential maps generated through two MLPs. Specifically, we have

$$\mu_\phi(d) = W_5 h + b_5, h = \tanh(W_4 d + b_4), log(\sigma^2_\phi(d)) = W_6 h + b_6$$

. For the target function, according to the VAE approach of [9] we define a variational lower bound $f_{\theta,\phi}(d) \geq log p_\theta(d)$ as

$$\begin{aligned} f_{\theta,\phi}(d) &= log p_\theta(d) \quad D_{KL}(q_\phi(z|d)|p_\theta(z|d)) \\ &= \sqrt{} dz q_\phi(z|d) log \Big) \frac{p_\phi(z|d)}{q_\phi(z|d)} \Big(\\ &= \underbrace{\sqrt{} dz q_\phi(z|d) log(p_\theta(d|z))}_{f_1} \quad \underbrace{\sqrt{} dz q_\phi(z|d) log \Big) \frac{p(z)}{q_\phi(z|d)} \Big(}_{f_2} \end{aligned}$$

with target function $f_{\theta,\phi}(d)$ to be maximized with respect to the parameters θ and ϕ. By exploiting the above formula for $q_\phi(z|d)$, the target function can be rewritten in the form

$$f_1(d) = \sqrt{} d\mu p_N(\mu; 0, I) log(p_\theta(d|\mu_\phi(d) + \sigma_\phi(d) \leq \mu))$$

$$= \frac{1}{L} \prod_{l=1}^{L} log(p_\theta(d|\mu_\phi(d) + \sigma_\phi(d) \leq \mu_l))$$

$$f_2(d) = \frac{1}{2} 1^T (1 + \mu_\phi^2(d) + \sigma_\phi^2(d) + log(\sigma_\phi^2(d)))$$

where $\leq$ stands for element-wise product, and where μ_l $\mathcal{O}$ $N(0, I)$ are independent normal samples. Below we report the Python implementation of the model

```
def encoder(self, x):
    if self.encoder_layers == 1:
        pi = F.relu(self.en1_fc(x))
        if self.encoder_shortcut:
            pi = self.en_drop(pi)
    else:
        pi = F.relu(self.en1_fc(x))
        pi = F.relu(self.en2_fc(pi))
        if self.encoder_shortcut:
            pi = self.en_drop(pi)

    #         mean = self.mean_bn(self.mean_fc(pi))
    #         logvar = self.logvar_bn(self.logvar_fc(pi))
    mean = self.mean_fc(pi)
    logvar = self.logvar_fc(pi)
    return mean, logvar

def sampler(self, mean, logvar, cuda):
    eps = torch.randn(mean.size())
    sigma = torch.exp(logvar)
    h = sigma.mul(eps).add_(mean)
    return h

def generator(self, h):
    if self.generator_layers == 0:
        r = h
    elif self.generator_layers == 1:
        temp = self.generator1(h)
        if self.generator_shortcut:
            r = torch.tanh(temp) + h
        else:
            r = temp
    elif self.generator_layers == 2:
        temp = torch.tanh(self.generator1(h))
        temp2 = self.generator2(temp)
        if self.generator_shortcut:
            r = torch.tanh(temp2) + h
        else:
            r = temp2
    else:
        temp = torch.tanh(self.generator1(h))
        temp2 = torch.tanh(self.generator2(temp))
        temp3 = torch.tanh(self.generator3(temp2))
        temp4 = self.generator4(temp3)
        if self.generator_shortcut:
            r = torch.tanh(temp4) + h
        else:
            r = temp4

    if self.generator_transform == 'tanh':
        return self.r_drop(torch.tanh(r))
    elif self.generator_transform == 'softmax':
        return self.r_drop(F.softmax(r,dim=1)[0])
    elif self.generator_transform == 'relu':
        return self.r_drop(F.relu(r))
    else:
        return self.r_drop(r)

def decoder(self, r):
    #         p_x_given_h = F.softmax(self.de_bn(self.de(r)),dim=1)
    p_x_given_h = F.softmax(self.de(r),dim=1)
    return p_x_given_h
```

Figure 3.4: VAE configuration

```
###topic model
mean, logvar = self.encoder(x)# batchsize*50
h = self.sampler(mean, logvar, cuda) # batchsize*50
r = self.generator(h)# batchsize*50
p_x_given_h = self.decoder(r) # batchsize*dv
```

Figure 3.5: VAE execution

3.2 Recurrent Neural Network

RNNs represent a fascinating class of neural network architectures designed primarily for identifying patterns within sequential data[36]. This type of data can encompass a wide range of applications, such as analyzing handwriting, decoding genomes, processing textual information, or handling numerical time series, often encountered in industrial contexts, like stock market data or sensor readings. Moreover, RNNs can even be applied to image data by decomposing images into patches and treating them as a sequence of information. On a broader scale, RNNs find extensive utility in tasks like Language Modeling, Text Generation, Speech Recognition, Image Description Generation, and Video Tagging. What truly sets Recurrent Neural Networks apart from their counterparts, such as Feedforward Neural Networks, also known as MLPs, is the way they manage the flow of information through the network. While Feedforward Networks convey data in a unidirectional manner without any feedback loops, RNNs embrace the concept of cyclic dependencies, allowing them to incorporate not only the present input X_t but also the historical input sequence $X_{0:t-1}$. This process of conveying information

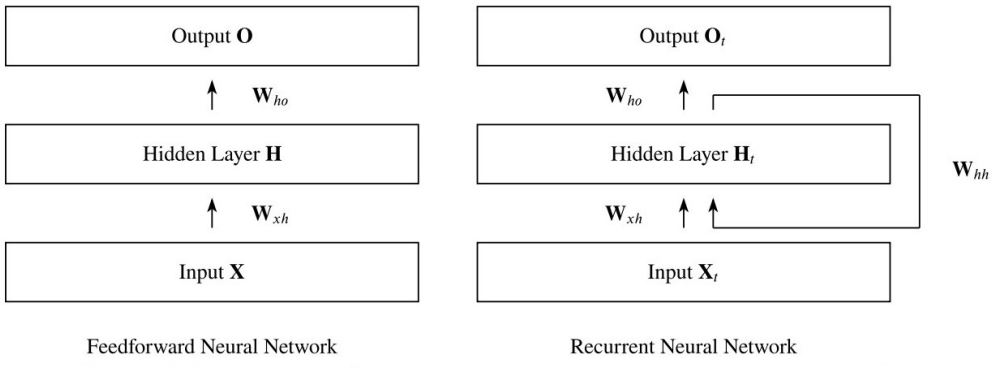


Figure 3.6: RNN vs FFN

from the previous time step to the current hidden layer can be described using mathematical notations introduced in the literature. In these notations, we denote the hidden state at time

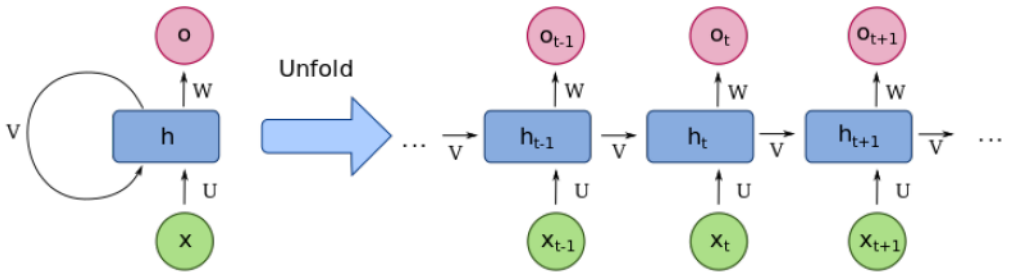


Figure 3.7: RNN

step t as $H_t \mathcal{O} R^{nxh}$ and the input as $X_t \mathcal{O} R^{nxd}$, where n represents the number of data samples, d signifies the dimension of the input for each sample, and h denotes the number of hidden units. We introduce essential weight matrices: $W_{xh} \mathcal{O} R^{dxh}$, representing the input-to-hidden-state transformation, W_{hh} inR^{hxh}, accounting for the hidden-state-to-hidden-state transitions, and a bias parameter $b_h \mathcal{O} R^{1xh}$. All this information is subsequently processed through an activation function, often a logistic sigmoid or tanh function. This step serves the purpose of preparing the gradients for efficient use in the backpropagation algorithm.

$$H_t = \phi_h(X_t W_{xh} + H_{t-1} W_{hh} + b_h)$$

$$O_t = \phi_o(H_t W_{ho} + b_o)$$

Crucially, due to the recursive nature of RNNs, the hidden state Ht at any given time step not only encodes information from the current input but also retains traces of all preceding hidden states up to Ht−1, which collectively imbues RNNs with a memory-like capability to model sequences effectively. In our work we use a hierarchical RNN, as proposed in [5] to model a document. Assuming that a document w_d contains M_d sentences, $w_d = \}s_1, s_2, ...s_{M_d}$, and the word embedding of jth word in ith sentence is w_i^j. Then, the representation of sentence s_i can be obtained by the following steps:

$$x_i^j = W \oplus w_i^j$$

$$\overleftarrow{h_i^j} = \overleftarrow{GRU}(x_i^j),$$

$$\overrightarrow{h_i^j} = \overrightarrow{GRU}(x_i^j),$$

$$h_i^j = \overleftarrow{h}_i^j \odot \overrightarrow{h}_i^j,$$

$$u_i^j = \tanh(W_w \oplus h_i^j + b_w),$$

$$\alpha_i^j = \frac{\exp(u^T \oplus u_i^j)}{\prod_t \exp(u^T \oplus u_i^t)},$$

$$s_i = \prod_{j=1}^{n} \alpha_i^j \oplus h_i^j$$

where $\overleftarrow{GRU}$ and $\overrightarrow{GRU}$ are bi-directional gated recurrent neural units for RNN, W, W_w, b_w are learned parameters in the classification model, and u_i^j is the attention vector of jth word in ith sentence, α_i^j is the attention signal captured by u_i^j, s_i is the learned representation for the ith sentence in document w_d. Then, we can learn the representation of document w_d with the similar architecture taking the input as a sequence of sentence representations. Finally, a softmax layer is stacked at the top to predict the class labels of documents by cross entropy loss between the predicted labels and the true labels.

$$L_c(w_d) = \prod p \oplus log[softmax(W_d \oplus w_d + b_d)],$$

where the output of $softmax(W_d \oplus w_d + b_d)$ is the distribution of predicted labels and p is the distribution of true labels. Below we report the Python implementation of the model

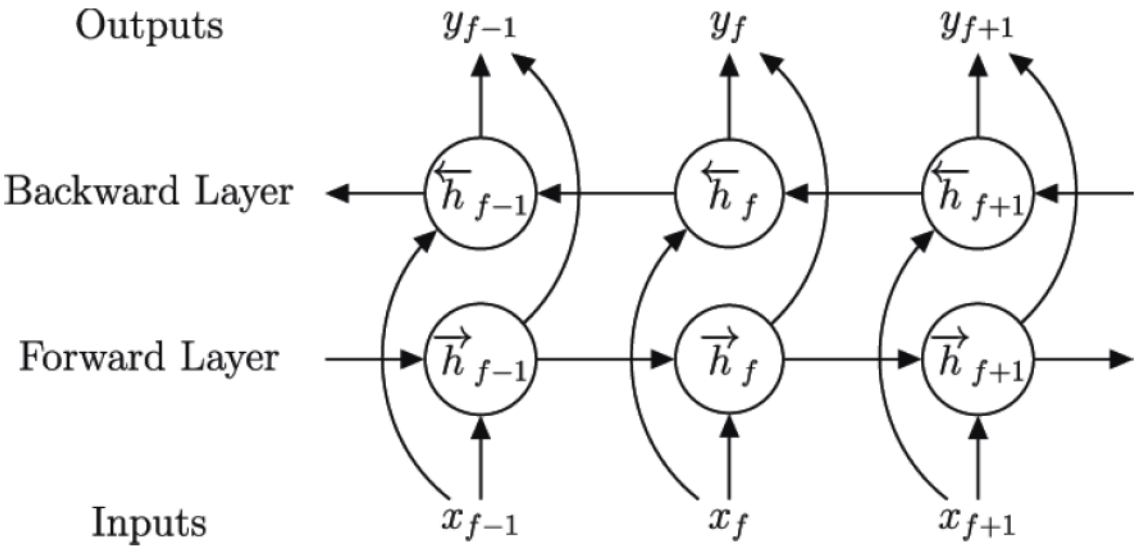


Figure 3.8: Bidirectional RNN

```python
def init_rnn_hidden(self, batch_size, level):
    param_data = next(self.parameters()).data
    if level == "word":
        bidirectional_multipier = 2 if self.word_rnn_bidirectional else 1
        layer_size = self.word_rnn_num_layer * bidirectional_multipier
        word_rnn_init_hidden = param_data.new(layer_size, batch_size, self.word_rnn_size).zero_()
        return word_rnn_init_hidden
    elif level == "context":
        bidirectional_multipier = 2 if self.context_rnn_bidirectional else 1
        layer_size = self.context_rnn_num_layer * bidirectional_multipier
        context_rnn_init_hidden = param_data.new(layer_size, batch_size, self.context_rnn_size).zero_()
        return context_rnn_init_hidden
    else:
        raise Exception("level must be 'word' or 'context'")
```

Figure 3.9: RNN configuration

```python
word_attention_dict = {}

num_utterance = len(input_list)  # one batch doucument_list
_, batch_size = input_list[0].size()

# word-level rnn
word_rnn_hidden = self.init_rnn_hidden(batch_size, level="word")
word_rnn_output_list = []
for utterance_index in range(num_utterance):
    word_rnn_input = self.embedding(input_list[utterance_index])
    word_rnn_output, word_rnn_hidden = self.word_rnn(word_rnn_input, word_rnn_hidden)
    word_attention_weight = self.word_conv_attention_linear(word_rnn_output)

    batch_data = input_list[utterance_index]
    for word_i in range(len(batch_data)):  # word_i word
        for clause_i in range(len(batch_data[word_i])):  # clause_i data(batch)
            word_index = int(batch_data[word_i, clause_i])  # word index
            if word_index < self.d_v:
                if word_index in word_attention_dict:
                    word_attention_dict[word_index] = (word_attention_dict[word_index] + word_attention_weight[word_i, clause_i,:]) / 2
                else:
                    word_attention_dict[word_index] = word_attention_weight[word_i, clause_i, :]

    ##HAN
    word_attention_weight = self.word_conv_attention_linear2(word_attention_weight)
    word_attention_weight = nn.functional.relu(word_attention_weight)
    word_attention_weight = nn.functional.softmax(word_attention_weight, dim=0)
    word_rnn_last_output = torch.mul(word_rnn_output, word_attention_weight).sum(dim=0)
    word_rnn_output_list.append(word_rnn_last_output)
    word_rnn_hidden = word_rnn_hidden.detach()
    # context-level rnn
context_rnn_hidden = self.init_rnn_hidden(batch_size, level="context")
context_rnn_input = torch.stack(word_rnn_output_list, dim=0)
context_rnn_output, context_rnn_hidden = self.context_rnn(context_rnn_input, context_rnn_hidden)
context_attention_weight = self.context_conv_attention_linear(context_rnn_output)
context_attention_weight = nn.functional.relu(context_attention_weight)
context_attention_weight = nn.functional.softmax(context_attention_weight, dim=0)
context_rnn_last_output = torch.mul(context_rnn_output, context_attention_weight).sum(dim=0)
classifier_input = context_rnn_last_output
logit = self.classifier(classifier_input)
```

Figure 3.10: RNN execution

3.3 BERT

The field of natural language processing has witnessed significant advancements through the implementation of language model pre-training techniques. These techniques have proven effective in enhancing a multitude of natural language processing tasks, ranging from tasks that operate at the sentence level, such as natural language inference and paraphrasing, to those that

demand fine-grained token-level analysis, like named entity recognition and question answering.

When it comes to employing pre-trained language representations for downstream tasks, two prominent strategies have emerged: feature-based and fine-tuning. In the feature-based approach, as exemplified by ELMo, task-specific architectures are constructed, incorporating pre-trained representations as supplementary features.

In contrast, the fine-tuning approach, as seen in models like the Generative Pre-trained Transformer (GPT), introduces minimal task-specific parameters. It trains on downstream tasks by fine-tuning all pre-trained parameters, effectively adapting the model to specific tasks. An inherent limitation of standard language models is their unidirectionality, which restricts the choice of architectures during pre-training. Here is where BERT, or "Bidirectional Encoder Representations from Transformers,"[22] emerges as a pivotal development. BERT alleviates the unidirectionality constraint by utilizing a "masked language model" (MLM) pre-training objective. This MLM objective randomly masks certain tokens in the input, challenging the model to predict the original vocabulary ID of the masked word solely based on its context. This approach empowers the representation to encapsulate both left and right context, paving the way for deep bidirectional Transformer pre-training. Fundamentally, BERT is a stack of Transformer encoder layers which consist of multiple self-attention "heads"[37]. For every input token in a sequence, each head computes key, value and query vectors, used to create a weighted representation. The outputs of all heads in the same layer are combined and run through a fully-connected layer. Each layer is wrapped with a skip connection and followed by layer normalization. The conventional workflow for BERT consists of two stages: pre-training and fine-tuning. Pretraining uses two self-supervised tasks: masked language modeling (MLM, prediction of randomly masked input tokens) and next sentence prediction (NSP, predicting if two input sentences are adjacent to each other). In fine-tuning for downstream applications, one or more fully-connected layers are typically added on top of the final encoder layer. One of the standout features of BERT is its unified architecture, applicable across a diverse array of tasks. BERT's model architecture is built upon a multi-layer bidirectional Transformer encoder, originally described in and released in the tensor2tensor library. This consistent architecture, adaptable for various tasks, has played a pivotal role in the success and widespread adoption of BERT in the field of natural language processing. after a careful study of the model, also based on what it says [38] we can summarise through a list, everything about the model's knowledge of language, such as semantics, syntax and knowledge about words

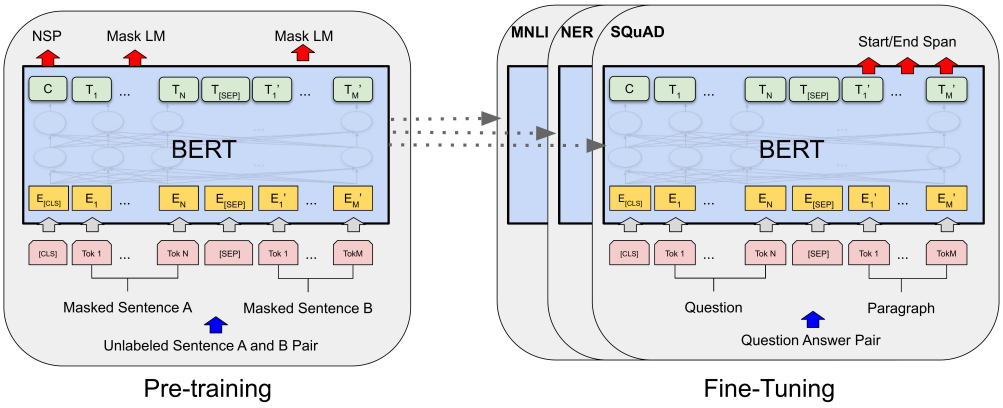


Figure 3.11: BERT

- Syntactic Knowledge:
  - BERT representations are hierarchical, suggesting a syntactic tree structure in addition to word order.
  - BERT embeddings encode information about parts of speech, syntactic chunks, and roles.
  - Syntactic information is captured in token embeddings and can be used to recover syntactic trees.

- Syntax Representation:
  - Syntactic structure is not directly encoded in self-attention weights.
  - BERT token representations can be used to recover syntactic dependencies in some cases.

- Semantic Knowledge:
  - BERT displays knowledge of semantic roles.
  - BERT has some preference for correct fillers related to the semantics.

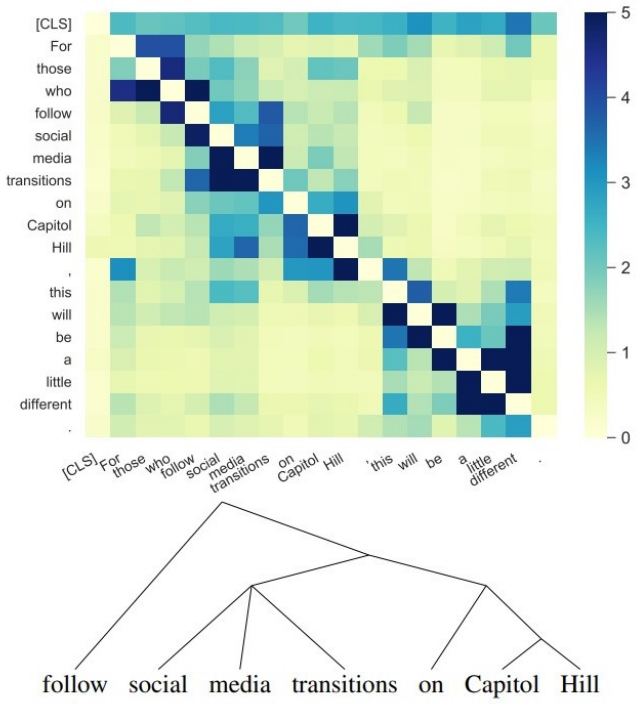


Figure 3.12: Parameter-free probe for syntactic knowledge: words sharing syntactic subtrees have larger impact on each other in the MLM prediction [1]

- World Knowledge:
  - BERT shows evidence of commonsense knowledge.
  - BERT can recall factual knowledge without fine-tuning.
  - BERT's world knowledge has limitations in reasoning and certain aspects of semantics.

Theoretically the figure 3.12 explain how the implementation of the BERT model for the classification part exchange information with the topic part. We use the same property of the RNN so we exchange the attention information's before the output layers. In our work we implement pre-trained Bert, as input we reconstruct the word of the VAE parts using a function that take the outputs of the detection part, that are tensors containing sequence of vectors for

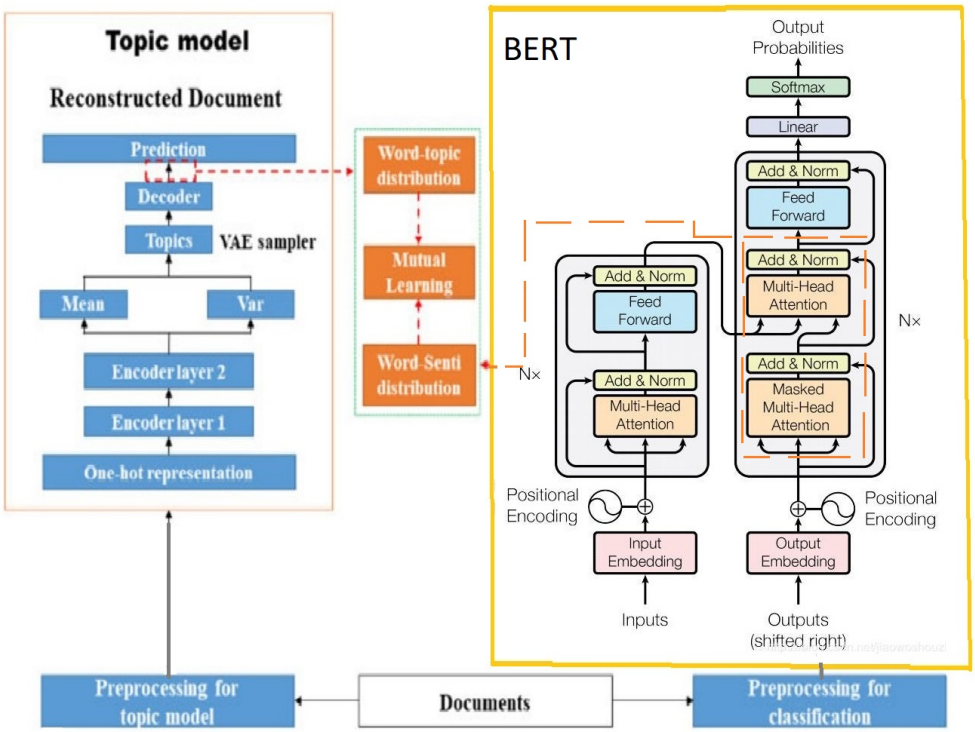


Figure 3.13: MLT with BERT

each sentence and each vector contains all the numbers associated to the words, once we have done it we merge the all the sentences in one list and we pass it to the pre-built BERT tokenizer.

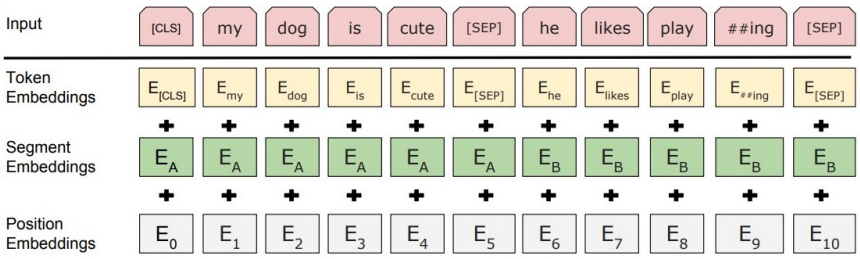


Figure 3.14: BERT Embeddings: In the context of sequence processing, each sequence commences with a distinctive classification token ([CLS]), while at the end, we demarcate individual sequences using a special token ([SEP]). The input representation for any given token is formed by the summation of the token's associated embeddings, including token, segment, and position embeddings.

Now we have the correct inputs for our model, we divide it into chunks for computations reasons and we set a specific token that specify to the model when a chunck start and when a chunk finished. When we have all the chunks ready we pass it to the BERT model already pre-trained with "bert-base-uncase" to obtain the final output. The bert-base-uncase model has 12 layers, 768 hidden, 12 heads and 110M parameters, it used as MLM, as explained above, and Next Sentence Prediction (NSP): the models concatenates two masked sentences as inputs during pretraining. Sometimes they correspond to sentences that were next to each other in the original text, sometimes not. The model then has to predict if the two sentences were following each other or not. the model not take in consideration the difference between a word that start with an uppercase letter and the same word written undercase. This way, the model learns an inner representation of the English language that can then be used to extract features useful for downstream tasks: if you have a dataset of labeled sentences, for instance, you can train a standard classifier using the features produced by the BERT model as inputs. Below we report the Python implementation of the model

```
    max_chunk_length = 64
    # Converte le rappresentazioni numeriche in testo utilizzando il tuo dizionario inverso
    tensor_inside_list = input_list[0]  # Assumendo che il tensore sia nel primo elemento della lista.

    # Converti il tensore in una lista di liste.
    list_of_lists = tensor_inside_list.tolist()

    # Ora puoi chiamare la tua funzione convert_ids2words.
    testo = self.convert_ids2words(list_of_lists, self.vocab)
    text_list = [' '.join(sentence) for sentence in testo]
    # Utilizza il tokenizer BERT per ottenere gli ID delle parole dai testi
    tokenized_inputs = self.tokenizer(text_list, padding=True, truncation=True, return_tensors='pt', max_length=max_chunk_length)
    # Calcola il numero di chunk che vuoi dividere
    num_chunks = tokenized_inputs['input_ids'].size(1) // max_chunk_length

    # Inizializza una lista vuota per contenere i chunk
    input_id_chunks = []

    # Dividi il tensore input_ids in chunk e aggiungili alla lista
    for i in range(num_chunks):
        start = i * max_chunk_length
        end = start + max_chunk_length
        chunk = tokenized_inputs['input_ids'][:, start:end]
        input_id_chunks.append(chunk)
    # Assicurati che ogni chunk abbia [CLS] all'inizio e [SEP] alla fine
    # ... (codice precedente)
    start_token = torch.tensor([101]).view(-1, 1)  # Inizializza il token di inizio con dimensione 1 x 1
    end_token = torch.tensor([102]).view(-1, 1)    # Inizializza il token di fine con dimensione 1 x 1
    input_id_chunks = [torch.cat([start_token.expand(chunk.size(0), -1), chunk, end_token.expand(chunk.size(0), -1)], dim=1) for chunk in input_id_chunks]
    # Ora chunks contiene la lista di chunk con lunghezza massima max_chunk_length o men
    bert_hidden_states_list = []
    for chunk in input_id_chunks:
        input_dict = {'input_ids': chunk,}
        bert_output = self.bert(**input_dict)
        if bert_output is not None and bert_output.hidden_states is not None and len(bert_output.hidden_states) > 0:
            print("the hidden states are:")
            print(bert_output.hidden_states)
            bert_hidden_states_list.append(bert_output.hidden_states)
    if len(bert_hidden_states_list) > 0:
        # Combina le rappresentazioni nascoste dei chunks
        combined_bert_hidden_states = torch.cat(bert_hidden_states_list, dim=1)
    else:
        # Nessuna rappresentazione nascosta valida disponibile, puoi eseguire un fallback o gestire l'assenza di input
        '''
        print("no hidden representation with bert ")
        '''
        combined_bert_hidden_states = torch.zeros_like(x)
# Utilizza le rappresentazioni nascoste di BERT come input al resto del tuo modello
    logit = self.classifier(combined_bert_hidden_states)
```

Figure 3.15: Bert execution

```
        self.tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')

        custom_config = BertConfig.from_pretrained('bert-base-uncased', output_hidden_state=True)
        custom_config.max_position_embeddings = 3000
        self.bert = BertModel(config=custom_config)
        # Definisci il resto della tua struttura, adattando l'input al tuo modello
        self.classifier = nn.Sequential(nn.Linear(3000, mlp_size), nn.LeakyReLU(), nn.Linear(mlp_size, num_label), nn.Tanh())

    def convert_ids2words(self, id_sequences, vocab):
        inverted_vocab = {v: k for k, v in vocab.items()}  # Inverti il dizionario
        return [[inverted_vocab.get(id, "<unk>") for id in seq] for seq in id_sequences]
```

Figure 3.16: Bert configuration

3.4 DirVAE

The dirVae model basically have the same scheme of a normal Variational autoencoder, there are a generative network and an Inference network. Is able to model the multi-modal distribution that was not possible with the Gaussian-Softmax and the GEM approaches. These characteristics allow DirVAE to be the prior of the discrete latent distribution, as the original Dirichlet distribution is[39]. Based on [40] we summarize the fundamental concept behind the Dirichlet Variational Autoencoder (DirVAE) is closely tied to the Dirichlet distribution, which is a multivariate generalization of the Beta distribution. The Dirichlet distribution represents a

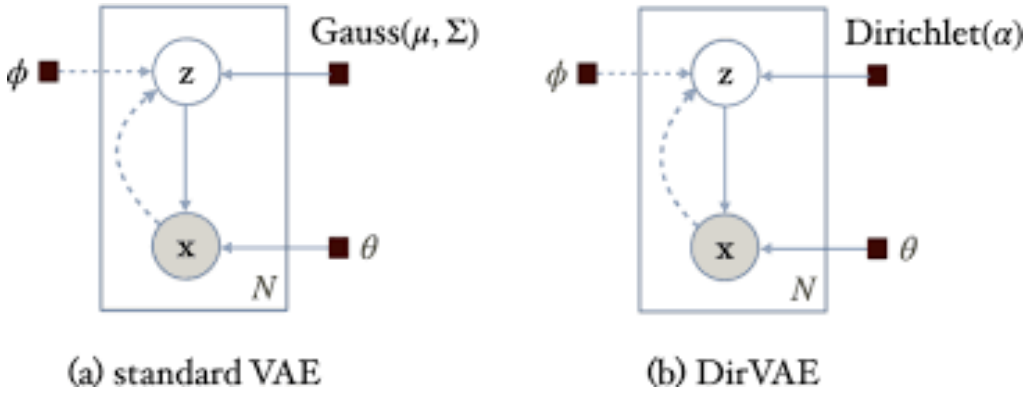


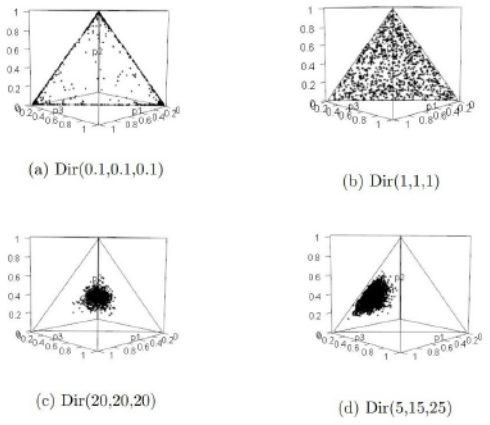
Figure 3.17: DirVae and standard VAE

multivariate continuous distribution of significant importance in the fields of probability and statistics. To better understand the DirVAE, we must first examine the Dirichlet distribution. This distribution is defined by a vector Y_k with k components, each of which is non-negative ($Y_i \sim 0$), and the sum of all components is equal to 1. It is also defined by a vector α_k with k components, each of which is strictly positive. The probability density function of the Dirichlet distribution is given by:

$$f(y_k) = \frac{\Gamma(\alpha_0)}{\sum_{i=1}^{k} \Gamma(\alpha_i)} \int_{i=1}^{k} y_i^{\alpha_i - 1}$$

Where $\alpha_0 = \prod_{i=1}^{k} \alpha_i, y_i > 0, y_1 + \ldots + y_{k-1} < 1$, and $y_k = 1 \quad y_1 \quad \ldots \quad y_{k-1}$. This distribution is denoted as $\text{Dir}(\alpha_1, \alpha_2, \ldots, \alpha_k)$.

It is important to note that the Dirichlet distribution is a distribution with k positive parameters α_k in a k-dimensional space. When $k = 2$, the probability density function is analogous to that of the Beta distribution with parameters α_1 and α_2. To better comprehend this distribution, we can visualize 1000 points generated from the Dirichlet distribution in a three-dimensional space with different values of parameter α_3. When α_1, α_2, and α_3 are all between 0 and 1, the density clusters around the edges of the simplex. This is observed when $\alpha_3 = (0.1, 0.1, 0.1)$, where the density congregates at the edges of the triangle, representing the sample space of Y_1, Y_2, and Y_3 in three-dimensional space.



(a) Dir(0.1,0.1,0.1)

(b) Dir(1,1,1)

(c) Dir(20,20,20)

(d) Dir(5,15,25)

Figure 3.18: Dirichlet distribution

However, as the value of α_3 increases to (1, 1, 1), the density becomes uniformly distributed across the entire triangle. When α_1, α_2, and α_3 are all greater than 1, the density becomes more concentrated at the center of the simplex, as seen in (c) when $\alpha_3 = (20, 20, 20)$. In (d), we observe that the density plot is not symmetric, as the values of α_1, α_2, and α_3 are not identical.

The Dirichlet Variational Autoencoder (DirVAE) was introduced in response to two specific situations:

- When Gaussian-Softmax or Softmax Laplace-based approaches fail to mimic the Dirichlet distribution.
- When non-parametric approaches may be influenced by biases that the Dirichlet distribution does not suffer from.

In these situations, DirVAE proves to be a valid solution, as it can model multi-modal distributions that cannot be accurately represented by other distributions or approaches.

In summary, the Dirichlet Variational Autoencoder (DirVAE) is an important innovation based on the fundamental Dirichlet distribution. This model is useful when modeling complex data distributions that cannot be accurately represented by other distributions or approaches. We choose to implement this model for two main reasons: Firstly, there are situations where conventional methods, such as Gaussian-Softmax approaches or Softmax Laplace approximations, fail to accurately mimic the Dirichlet distribution. This is where DirVAE steps in, offering a solution to capture more complex, multi-modal data distributions. Secondly, non-parametric approaches, while powerful, may introduce certain biases that the Dirichlet distribution inherently avoids. In scenarios where you need unbiased representations of data, DirVAE proves its mettle. Now we highlights the Pros and the Cons of the dirVAE method

- Pros:
  - Effective Multi-Modal Modeling: DirVAE excels in modeling multi-modal probability distributions, especially in high-dimensional spaces.
  - Fair Weight Distribution: It ensures fair weight distribution among components, which is crucial for various applications.
  - Flexible Hyper-Parameters: Dirichlet distribution's prior hyper-parameters can be adjusted to control and capture multi-modal characteristics.

- Cons:
  - Complexity: DirVAE can be more complex than traditional models, requiring a deeper understanding of Dirichlet distribution and its hyper-parameters.

– Learning Curve: Users may need to familiarize themselves with Dirichlet-based modeling techniques, which might involve a steeper learning curve.

– Computational Demands: The computational demands of DirVAE may be higher due to its ability to capture multi-modal data distribution.

In theory we have the same scheme of a normal VAE we emphasize the difference in the two networks: In the generative network the key difference is the prior distribution assumption on the latent variable z. Instead of using the standard Gaussian distribution, we use the Dirichlet distribution which is a conjugate prior distribution of multinominial distribution.

$$z \approx p(z) = Dirichlet(\alpha), x \approx p_\theta(x|z)$$

For the inference network probabilistic encoder with an approximating posterior distribution $q_\phi(z|x)$ is designed to be $Dirichlet(\alpha)$. The approximated posterior parameter α is derived by the MLP from the observation x in dataset D with positive output function such as softplus function, so the outputs can be positive values constrained by the Dirichlet distribution.In our work we implement the DirVAE in our algorithm, replacing the normal VAE to watch the performance in a real application. The figure 3.19 is the same of the standard MTL model the only difference is that in this model we change the distribution in the Dirichlet distribution. In the python application we add in the forward funcion a softmax layer in order to obtain a

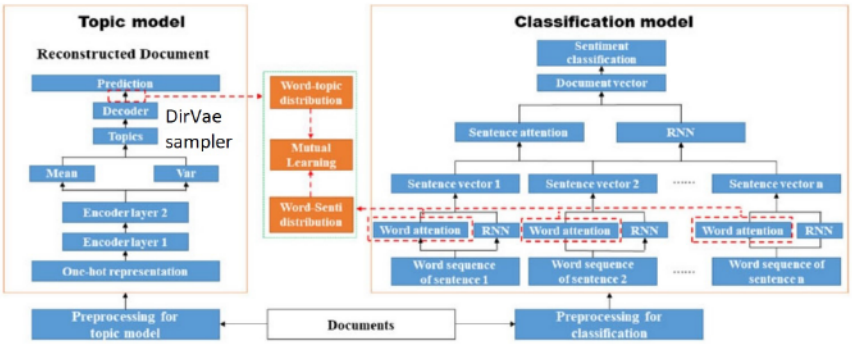


Figure 3.19: MLT with DirVae

Dirichlet distribution of our data[39]. we report below the python implementations

```
###topic model
mean, logvar = self.encoder(x)# batchsize*50
h = self.sampler(mean, logvar, cuda)# batchsize*50
#to apply the dirichlet distribution we need to apply a softmax layer to the distribution h in order to obtain a distribution that follows the dirichlet
h_dirc = F.softmax(h,dim=1)#now we have h that follow dirchlet distribution
r = self.generator(h_dirc)# batchsize*50
p_x_given_h = self.decoder(r) # batchsize*dv
```

Figure 3.20: DirVae execution

4 Results

4.1 Complexity

We evaluate the complexity in term of time taken to do a run of the algorithm based in 3 epochs of training. below the results for the the different algorithms

VAE + RNN								
	IMDB					YELP		
batch size	10	25	50	100	500	25	50	100
time(h)	6h	5h	4h	5h 30 min.	9h 15 min.	7h	5h 50 min	6h

Table 4.1: Complexity VAE + RNN

YELP		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
time(h)	5h 15 min	5h 15 min

Table 4.2: Complexity YELP BERT-DirVAE

We can see that at a certain point the value of the overall time restart growing this is due to the fact that there is a trade-off between the complexity of one iteration and the complexity of the overall computations. The time decrease increasing the batch size because the complexity of

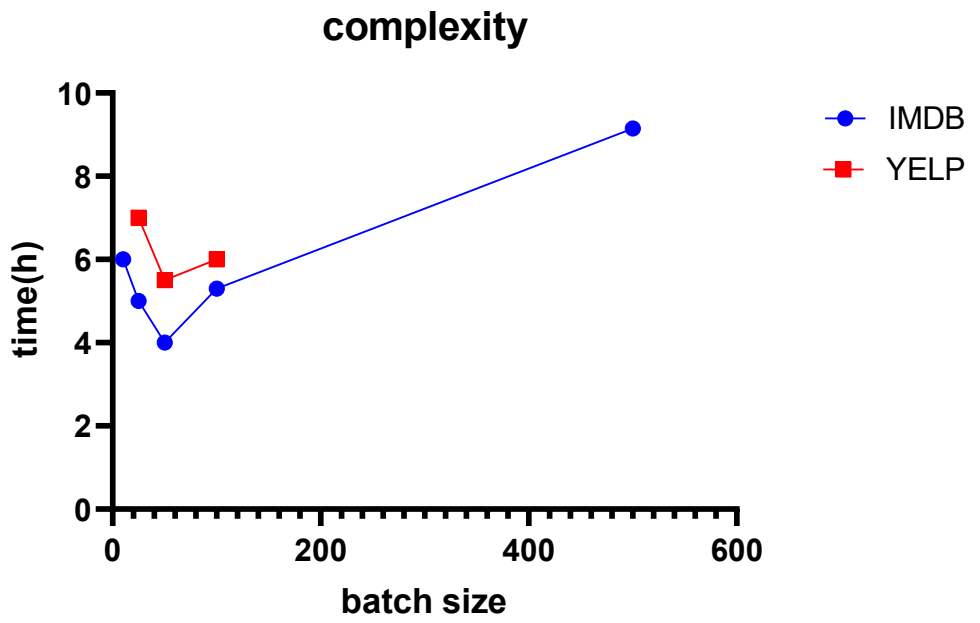


Figure 4.1: Complexity

IMDB		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
time(h)	4h	4h 10 min

Table 4.3: Complexity IMDB BERT-DirVAE

the computation reduce significatively dividing the entire dataset in batches. When we achieve the best parameter the time start to grows because the number of the operations are extremely large and this affect the benefits due to the easiest calculus. The two variations of the algorithm, one with Bert in the sentiment analysis and one with the DirVae in the topic detection, have similar overall time with respect to the original algorithm but we can notice that demand a little bit less of time in the YELP dataset, five hours and fifteen minutes instead of fifths minutes, in the IMDB datset the time is the same with the Bert model and ten minutes more with DirVae model. With this results we choose 50 as a value of batch size for the to others two model and to find the other parameters. We try also to see if the vocabulary size influence the complexity of the methods because in the preprocess of the data the model create a vocabulary, this part have no influece in the overall time because the process take only some minuts to creates differents vocabulary in order to respect the parameter that we want, is this the reason why we did not

put the relatively results.

4.2 Accuracy

We measure the performance based on the accuracy, as specified in the section 1, that metric evaluate the number of the correct predictions with respect to the number of the total predictions. below the table that report our results

VAE + RNN						
	IMDB			YELP		
vocab size	2000	3000	5000	2000	3000	5000
batch size	50	50	50	50	50	50
accuracy	0,17	0,192	0,127	0,229	0.227	0,227

Table 4.4: Accuracy VAE + RNN

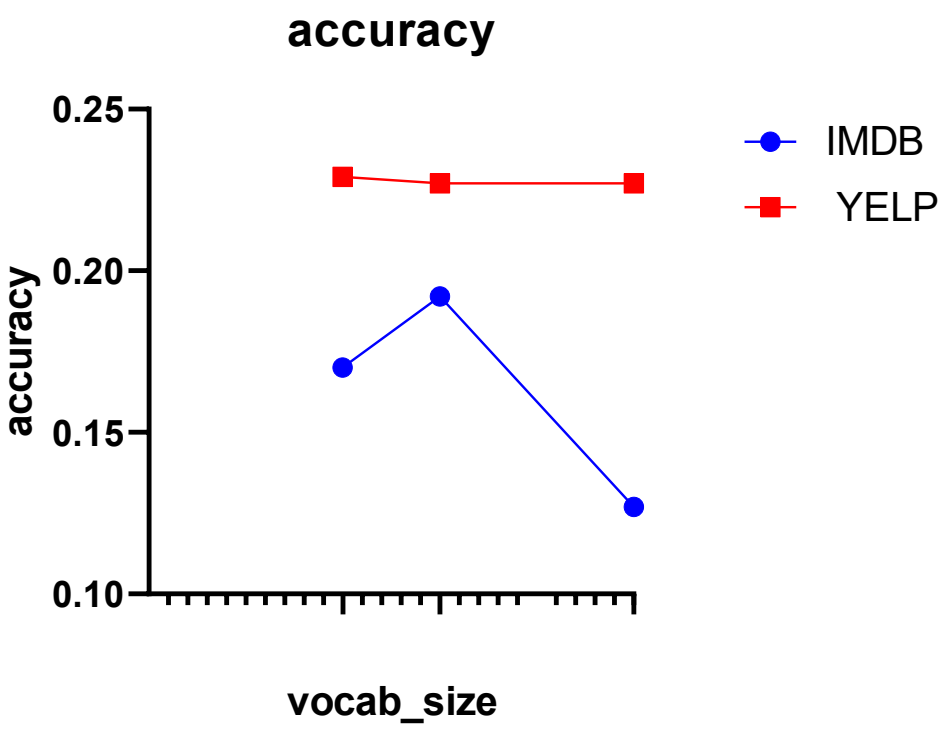


Figure 4.2: Accuracy

YELP		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
vocab size	3000	3000
accuracy	0,209	0,274

Table 4.5: Accuracy YELP BERT-DirVAE

IMDB		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
vocab size	3000	3000
accuracy	0,051	0,160

Table 4.6: Accuracy IMDB BERT-DirVAE

The first table show that in the IMBD dataset, the number of the vocabulary size impact more the performance of the model which respect to the YELP dataset. For the IMDB dataset the best value in terms of accuracy is with 3k of vocabulary size, instead the YELP with 2k. in the first dataset the variation is larger, round 2 5%, in the second the variation is very small, like 0.002%. So that we fix to 3k the vocabulary size for run the others two algorithms. For the IMDB dataset we have two different behaviours in terms of accuracy, the model with Bert part achieve very low performance with respect to the standard algorithm; this is due to the complexity of the model and the computational power that we have, the Bert model is a pretrained model that require a lot of data to work correctly. with the IMDB dataset we cannot give it a sufficiently amount of data, in fact if we compare to the same model in the YELP dataset that is twice bigger the model achieve an accuracy round the standard model. The DirVae model in the IMDB dataset achieve a bit less accuracy than the standard and in the YELP dataset achieve better performance with respect to the standard model.

4.3 Loss

We evaluate the Loss in order to see another measure of the performances. In our model due to the scheme and the purpose, we have two different loss, the first one is for the topic detection and the second one is for classification and sentiment analysis, as in the previous section we evaluate the standard model with three differences vocabulary size in order to see the behaviours and choose it to run the others two models

4.3.1 Topic Detection Loss

The first measure that we analyze is the topic detection loss, related to the topic detection part.

VAE + RNN						
	IMDB			YELP		
vocab size	2000	3000	5000	2000	3000	5000
batch size	50	50	50	50	50	50
topic loss	97,2639	34,4885	83,8601	12,3715	3,344	12,6933

Table 4.7: Topic Loss VAE + RNN

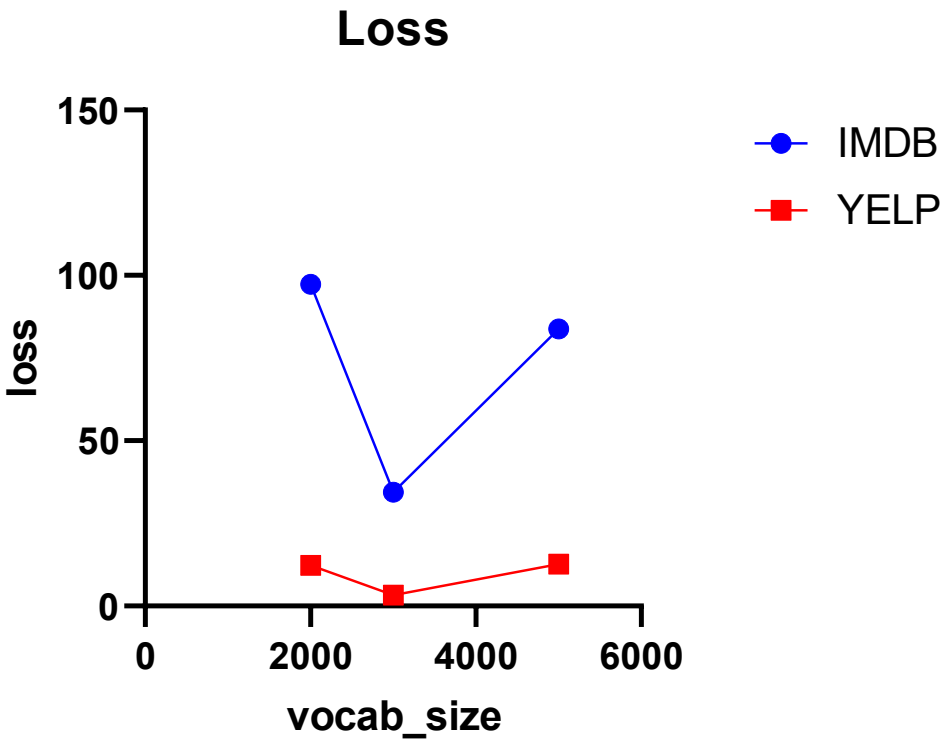


Figure 4.3: Topic Detection Loss

For the model composed by the RNN and VAE we have the same behaviour with respect to the vocabulary size, the best loss was achieved by 3 thousand vocabulary size, as in the accuracy measure the IMDB dataset have large variation between the differences measures of vocabulary instead of the YELP dataset that have this variation very low between 2 and 5 thousands

YELP		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
vocab size	3000	3000
topic loss	64,6515	20,3235

Table 4.8: Topic Loss YELP BERT-DirVae

IMDB		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
vocab size	3000	3000
topic loss	34,6483	119,4563

Table 4.9: Topic Loss IMDB BERT-DirVae

vocabulary size. Once we have found the best vocabulary size parameters we run the others two algorithms, we can notice that in this case the loss value in both datasets increase a lots in both algorithms. This is pretty strange because for example in the Bert model the topic part remain the same of the standard model but this prove how the two parts influence each others respectively.

4.3.2 Classification Loss

Another measure was the classification error, related to the sentiment analysis, below we report our results In the standard model we can see that the classification error are quite low in both

VAE + RNN						
	IMDB			YELP		
vocab size	2000	3000	5000	2000	3000	5000
batch size	50	50	50	50	50	50
classification loss	2,2078	2,1559	2,1086	1,6148	1,6305	1,6096

Table 4.10: Classification Loss VAE + RNN

datasets, in this case we highlights data the variation between the differences size of vocabulary not influence as in the topic loss, for this reason we choose as in the topic detection the 3k vocabulary to run the other variants of the algorithm. as in the standard model the variations not have a big difference, so that in this particularly metric the models works very similarly.

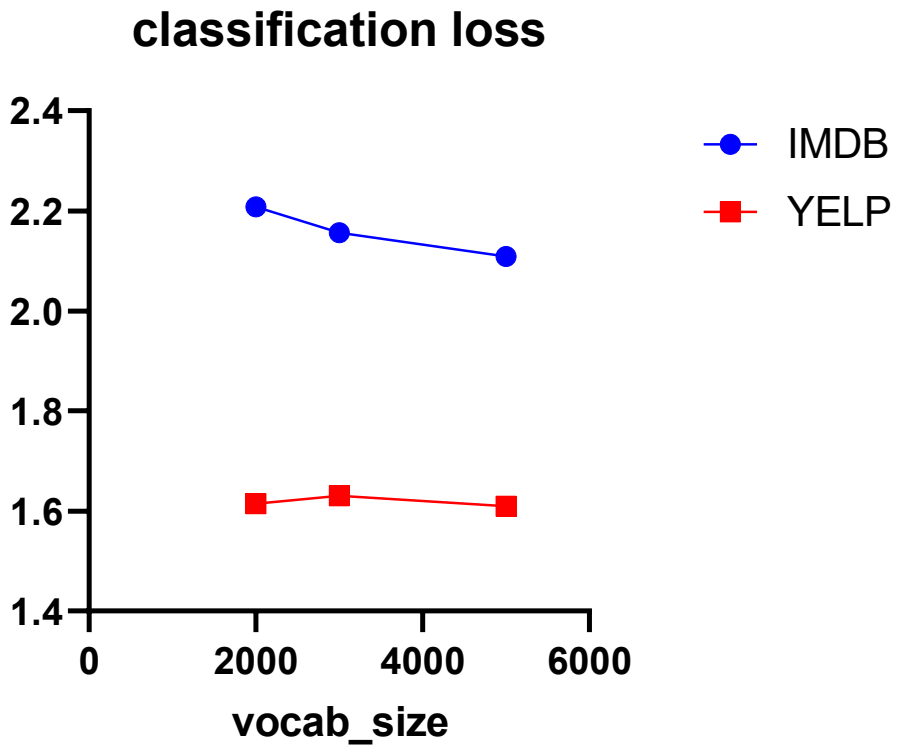


Figure 4.4: Classification loss

YELP		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
vocab size	3000	3000
classification loss	1,7592	1,6074

Table 4.11: Classification Loss YELP BERT-DirVae

IMDB		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
vocab size	3000	3000
classification loss	2,3015	2,1459

Table 4.12: Classification Loss IMDB BERT-DirVae

The quantity of the classification error are better in the Yelp dataset this is due as in the other permance evaluators because the dataset is bigger so the model have more information to take it decisions

4.4 Kullback leibler divergence

We calculate the KLD between the original distribution in the datasets and the distribution of the output in order to evaluate the dissimilarity between the two.

VAE + RNN						
	IMDB			YELP		
vocab size	2000	3000	5000	2000	3000	5000
batch size	50	50	50	50	50	50
kld	0,0031	0,0011	0,0009	0,0006	0,0002	0,0002

Table 4.13: KLD VAE + RNN

YELP		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
vocab size	3000	3000
kld	0,0060	0,0001

Table 4.14: KLD YELP BERT-DirVae

IMDB		
model	VAE + BERT	DirVAE + RNN
batch size	50	50
vocab size	3000	3000
kld	0,0004	0,0013

Table 4.15: KLD IMDB BERT-DirVae

For the KLD we have slightly differences between the two datasets, the first one achieved the best value with 5k vacabulary, and the second one with 3k and 5k. The difference between the distribution of the model and the original distribution is very low. As in the classification errors we choose 3k dictionary because the difference with the best result is very low and if we choose the same size of the dictionary as the other metrices we can have a complete overview between

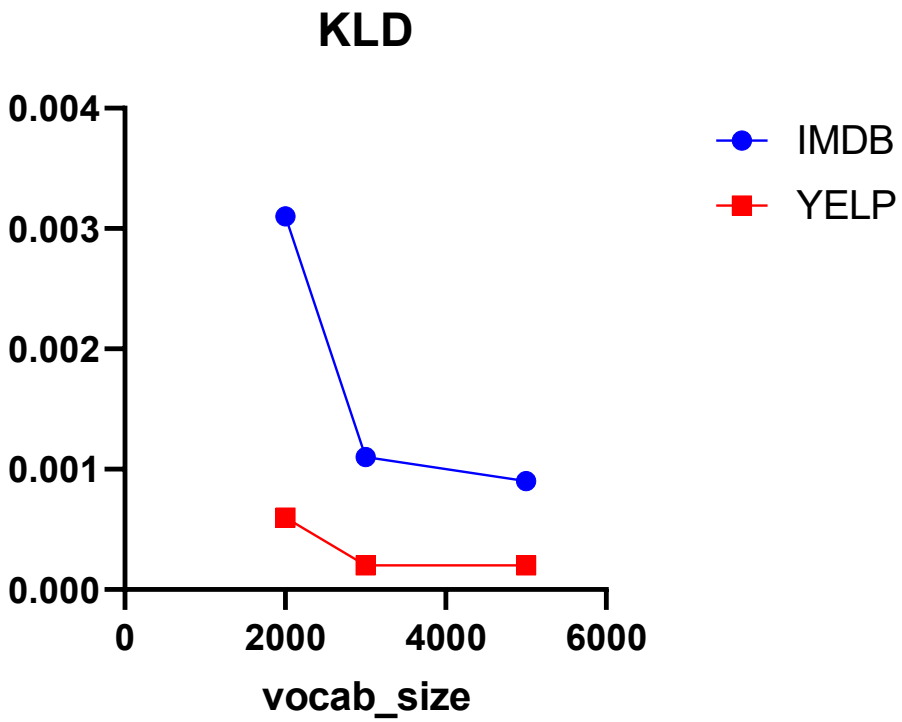


Figure 4.5: KLD

the standard model and the two variations. In the other two we have slightly differences in the results, in the IMDB dataset work better the dirchlet method with recurrent neural network, instead of the YELP dataset in which works better the model with BERT. this two methods work better with respect to the standard one because they reach better performance.

5
Conclusion

Summarizing the results, we have observed how the studied algorithms exhibit a significant need for computational power, and when unavailable, the execution time becomes notably extensive. Additionally, we noted that vocabulary size has a more pronounced impact on performance in smaller datasets compared to larger ones.

From an accuracy standpoint, our baseline model performs better with the larger dataset but fails to reach high levels due to the lack of computational power for more effective training. The DirVae model achieves accuracy levels similar to the baseline, and in the YELP dataset, it surpasses it. However, the BERT model shows a slightly lower performance with the YELP dataset and significantly drops with the IMDB dataset, attributed to the model's complexity and its demand for substantial data quantities.

Regarding loss in the topic analysis section, the baseline model, like other metrics, performs better with the larger dataset. However, the two modified models do not perform well, resulting in a substantial increase in the loss value.

In the sentiment analysis section, both the baseline model and its variations perform well, achieving low values. The results in the distribution analysis are excellent, and in certain cases, the modified models even outperform the standard model.

Concluding the results analysis, we can assert that they are heavily influenced by the computational power at our disposal. With few exceptions, the variations are minimal, making it challenging to declare one model significantly superior to another. This leads us to conclude that, depending on applications and needs, the MLT model with mutual learning is customiz-

able and adaptable to specific requirements.

Future work could involve rerunning the experiments with more powerful equipment, expanding the range of models for comparison to obtain a more comprehensive overview of mutual learning techniques.

In alignment with the objectives of our thesis, we can conclude that we have successfully completed the section related to general understanding, affirming that the system functions correctly. However, we cannot claim a substantial improvement in performance by using more recent models for individual tasks.

References

[1] Z. Wu, Y. Chen, B. Kao, and Q. Liu, "Perturbed masking: Parameter-free probing for analyzing and interpreting BERT," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, D. Jurafsky, J. Chai, N. Schluter, and J. Tetreault, Eds. Online: Association for Computational Linguistics, Jul. 2020, pp. 4166–4176. [Online]. Available: https://aclanthology.org/2020.acl-main.383

[2] A. Yasin, Y. Ben-asher, and A. Mendelson, "Deep-dive analysis of the data analytics workload in cloudsuite," 12 2014, pp. 202–211.

[3] E. D. Liddy, "Natural language processing," 2001.

[4] Y. Bengio, R. Ducharme, and P. Vincent, "A neural probabilistic language model," in *Advances in Neural Information Processing Systems*, T. Leen, T. Dietterich, and V. Tresp, Eds., vol. 13. MIT Press, 2000. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2000/file/728f206c2a01bf572b5940d7d9a8fa4c-Paper.pdf

[5] L. Gui, J. Leng, J. Zhou, R. Xu, and Y. He, "Multi task mutual learning for joint sentiment classification and topic detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 34, no. 4, pp. 1915–1927, 2022.

[6] R. Das, M. Zaheer, and C. Dyer, "Gaussian LDA for topic models with word embeddings," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. Beijing, China: Association for Computational Linguistics, Jul. 2015, pp. 795–804. [Online]. Available: https://aclanthology.org/P15-1077

[7] P. Xie, J. Zhu, and E. P. Xing, "Diversity-promoting bayesian learning of latent variable models," 2017.

[8] M. Peng, Q. Xie, Y. Zhang, H. Wang, X. Zhang, J. Huang, and G. Tian, "Neural sparse topical coding," in *Proceedings of the 56th Annual Meeting of the Association for*

Computational Linguistics (Volume 1: Long Papers). Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 2332–2340. [Online]. Available: https://aclanthology.org/P18-1217

[9] D. P. Kingma and M. Welling, "Auto-encoding variational bayes," 2022.

[10] Y. Miao, L. Yu, and P. Blunsom, "Neural variational inference for text processing," 2016.

[11] D. Card, C. Tan, and N. A. Smith, "Neural models for documents with metadata," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. Association for Computational Linguistics, 2018. [Online]. Available: https://doi.org/10.18653%2Fv1%2Fp18-1189

[12] S. Shayaa, N. I. Jaafar, S. Bahri, A. Sulaiman, P. S. Wai, Y. W. Chung, A. Z. Piprani, and M. A. Al-Garadi, "Sentiment analysis of big data: methods, applications, and open challenges," *Ieee Access*, vol. 6, pp. 37 807–37 827, 2018.

[13] W. Medhat, A. Hassan, and H. Korashy, "Sentiment analysis algorithms and applications: A survey," *Ain Shams Engineering Journal*, vol. 5, no. 4, pp. 1093–1113, 2014. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2090447914000550

[14] Y. Mejova, "Sentiment analysis: An overview," *University of Iowa, Computer Science Department*, 2009.

[15] B. Pang, L. Lee, and S. Vaithyanathan, "Thumbs up? sentiment classification using machine learning techniques," in *Proceedings of the 2002 Conference on Empirical Methods in Natural Language Processing (EMNLP 2002)*. Association for Computational Linguistics, Jul. 2002, pp. 79–86. [Online]. Available: https://aclanthology.org/W02-1011

[16] S. Wang and C. Manning, "Baselines and bigrams: Simple, good sentiment and topic classification," in *Proceedings of the 50th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Jeju Island, Korea: Association for Computational Linguistics, Jul. 2012, pp. 90–94. [Online]. Available: https://aclanthology.org/P12-2018

[17] E. Cambria, S. Poria, A. Gelbukh, and M. Thelwall, "Sentiment analysis is a big suitcase," *IEEE Intelligent Systems*, vol. 32, no. 6, pp. 74–80, 2017.

[18] Y. Kim, "Convolutional neural networks for sentence classification," 2014.

[19] A. Graves, "Generating sequences with recurrent neural networks," 2014.

[20] D. Tang, B. Qin, and T. Liu, "Document modeling with gated recurrent neural network for sentiment classification," in *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. Lisbon, Portugal: Association for Computational Linguistics, Sep. 2015, pp. 1422–1432. [Online]. Available: https://aclanthology.org/D15-1167

[21] Z. Yang, D. Yang, C. Dyer, X. He, A. Smola, and E. Hovy, "Hierarchical attention networks for document classification," in *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. San Diego, California: Association for Computational Linguistics, Jun. 2016, pp. 1480–1489. [Online]. Available: https://aclanthology.org/N16-1174

[22] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," 2019.

[23] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018.

[24] P. Liu, X. Qiu, and X. Huang, "Adversarial multi-task learning for text classification," 2017.

[25] M. Isonuma, T. Fujino, J. Mori, Y. Matsuo, and I. Sakata, "Extractive summarization using multi-task learning with document classification," in *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*. Copenhagen, Denmark: Association for Computational Linguistics, Sep. 2017, pp. 2101–2110. [Online]. Available: https://aclanthology.org/D17-1223

[26] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa, "Natural language processing (almost) from scratch," 2011.

[27] A. Søgaard and Y. Goldberg, "Deep multi-task learning with low level tasks supervised at lower layers," in *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Berlin, Germany: Association for Computational Linguistics, Aug. 2016, pp. 231–235. [Online]. Available: https://aclanthology.org/P16-2038

[28] H. M. Alonso and B. Plank, "When is multitask learning effective? semantic sequence prediction under varying data conditions," 2017.

[29] K. Singla, D. Can, and S. Narayanan, "A multi-task approach to learning multilingual representations," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 214–220. [Online]. Available: https://aclanthology.org/P18-2035

[30] Y. Zhang and Q. Yang, "A survey on multi-task learning," 2021.

[31] C. Lin and Y. He, "Joint sentiment/topic model for sentiment analysis," in *Proceedings of the 18th ACM Conference on Information and Knowledge Management*, ser. CIKM '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 375–384. [Online]. Available: https://doi.org/10.1145/1645953.1646003

[32] S. Moral, A. Cano, and M. Gómez-Olmedo, "Computation of kullback–leibler divergence in bayesian networks," *Entropy*, vol. 23, no. 9, 2021. [Online]. Available: https://www.mdpi.com/1099-4300/23/9/1122

[33] L. Li, M. Doroslovački, and M. H. Loew, "Approximating the gradient of cross-entropy loss function," *IEEE Access*, vol. 8, pp. 111 626–111 635, 2020.

[34] A. Mao, M. Mohri, and Y. Zhong, "Cross-entropy loss functions: Theoretical analysis and applications," 2023.

[35] T. Zhang, "Statistical behavior and consistency of classification methods based on convex risk minimization," *The Annals of Statistics*, vol. 32, no. 1, pp. 56 – 85, 2004. [Online]. Available: https://doi.org/10.1214/aos/1079120130

[36] R. M. Schmidt, "Recurrent neural networks (rnns): A gentle introduction and overview," 2019.

[37] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2023.

[38] A. Rogers, O. Kovaleva, and A. Rumshisky, "A primer in bertology: What we know about how bert works," 2020.

[39] W. Joo, W. Lee, S. Park, and I.-C. Moon, "Dirichlet variational autoencoder," 2019.

[40] J. Lin, "On the dirichlet distribution," *Department of Mathematics and Statistics, Queens University*, pp. 10–11, 2016.

Acknowledgments

We use this python libraries:

- numpy
- optsparse
- pytorch
- re
- time
- json
- scipy
- joblib
- transformers
- os
- gc
- copy
- progressbar
- codecs
- collections
- spacy

- gensim
- sys
- other libraries create by our own