POLITECNICO DI TORINO

Master's Degree in Artificial Intelligence and Data Analytics

Master's Degree Thesis

Fine-tuning Deep Language Models for Zero-Shot Text Classification

Supervisors
Garza Paolo
Bongiovanni Lorenzo

Candidate
Fontana Elia

Academic year 2022-2023

Abstract

High quality Zero-Shot Text Classification is one of the holy grails of NLP as it allows to avoid the difficult, time-consuming and expensive process of collecting and labelling data for supervised training. Deep language models have shown remarkable capabilities in various natural language processing tasks, but their effectiveness in Zero-Shot Text Classification remains an area of exploration. Surely, large language model (LLMs), e.g., GPT4 and LaMDA, have undoubtedly shown stunning generalization capabilities but they are not open-source and anyway intractable with normal computing resources.
The aim of this thesis is to go deeper and analyze this task, in the context of tractable, open-source language models.
In particular, we focus on MPNet, a language model pre-trained on extensive general corpora and specialized on the task of Semantic Text Similarity (STS). We explore the advantages of implementing a supervised contrastive learning objective during the fine-tuning phase to address the challenge of Zero-Shot Text Classification.
The main focus of this work is centred on enhancing the model's Zero-Shot capability by generating a better-suited vector-based representation for short sentences like noun phrases, used as labels.
Given a document, such as scientific paper or journal article, consisting of a title and a description, noun phrases are extracted from title. The framework aim is to generate embeddings for this short-text keywords in such a way that they are as close as possible in the semantic vector space to the embedding of the associated long-text description.
Furthermore, an analysis of alignment and distribution uniformity within these generated vectors is conducted to gain a deeper understanding of the semantic vector space generated by MPNet during fine-tuning.
By shedding light on these aspects, this thesis contributes to a deeper understanding of Zero-Shot Text Classification and presents novel insights that may pave the way in enhancing the performance and capabilities of deep language models in the context of Zero-Shot Text Classification.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Zero-Shot Text Classification

In last decades supervised text classification has reached great success through the usage of enormous amount of training data and the introduction of trasformer-based Deep Language Models. Similarly, Zero-Shot Text Classification (0SHOT-TC) is an application that has great potential in real word scenarios and it is somehow less explored than traditional supervised methods. In conventional text classification the labels of each class are typically associated with class numerical indices 0,1,2,...,n which does not permit to exploit the semantic of labels. This approach cannot be exploited in 0SHOT-TC where labels can be different every time and it is not possible to assume the presence of labeled examples for training the model. Moreover, we can categorize a text base on different aspects. For example as depicted in figure 1.1 a phrase can be interpreted on different levels of meaning, such as the topic (for example this text discuss a finance or sport argument) or the emotion (for instance this text express joy or sadness).
For humans, associating multiple and diverse meaning words with a text for the purpose of categorization, is a straightforward task. This because we have the capability to interpret word meaning and its contextualization within a sentence. The idea of Zero-Shot Text Classification is to design a framework that, much like humans, can determine, for any forthcoming word acting as a label, whether it can be associated to a text.
To summarize, we define Zero-Shot Text Classification as the task of categorizing a text according to one ore more labels that have never been seen by the model during training, relying only on semantics of both the text and labels themselves. In a way, we can think of this method as a case of transfer

learning, where a model pre-trained on one task is employed for a different application.

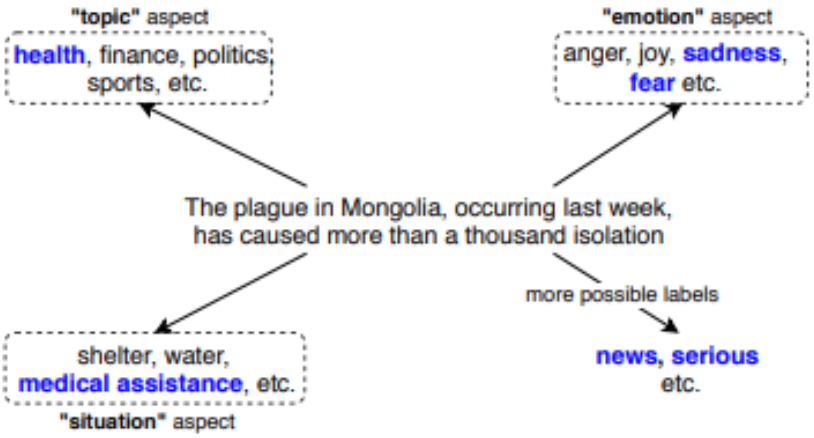

Figure 1.1. Text can be annotated with labels that describe its various facets, from the topic treated, to the sentiment that it want to convey. Positive labels are highlighted in blue. Image from [19]

1.2 Semantic Similarity

One of the key concept in the Zero-Shot Text Classification task is the semantic similarity.
Semantic similarity represent a distance measure between the semantic meanings of pair of words, sentences or document. It is defined over a set of terms or documents where the concept of distance between items is based on their similarity in meaning or semantic content rather than their lexicographical likeness.
In fact, two pieces of text can express very similar concepts despite the words in them being completely different.
Moreover, semantic similarity is often associated to semantic relatedness. This concept goes beyond just measuring the semantic similarity (similarity in meaning), but also involves a wider analysis of the common semantic characteristics/properties between (two) words. For example, the words 'coffee' and 'mug' may be related to one another closely, but they are not considered semantically similar whereas the words 'coffee' and 'tea' are semantically similar. The idea is to bring closer entailed word vector representations while

push away not correlated vectors.

1.3 Objective of thesis

Annotation of a dataset is a labor-intensive and time-consuming process. On top of that, a dataset labelled according to a certain set of labels becomes useless when new labels are introduced, which is a common in many real word scenarios and particularly in the scientific field where the introduction of new terms to describe specific concepts is unavoidable.
This is the main challenge that this work wants to address by leveraging a Deep Language Model, pre-trained for Semantic Text Similarity (STS), and specializing it to make it more suitable for the task of Zero-Shot Text Classification.
To this end, we devise a fine-tuning objective where the model has to learn to associate some text to a set of relevant keyword related to it, where these keywords are different for each text. We collect a big dataset of documents, each one composed by title and abstract/description, where the title is typically a good summary of the content of the abstract.
Keeping this in mind, we extract noun-phrases from each title, and those are almost certainly guaranteed to be relevant keywords for the respective abstract.
The first step consists in extracting noun-phrases (keywords) from the title of each document, that in a certain way summarized and categorized the article.
Next, the idea is to train the model to generate a vector-representation for these keywords in a manner that captures their semantic relevance with respect to the corresponding article.
At the end of this process we want to achieve the following result:
given a text and a list of keywords, we want out model to be able to generate vector representations such that the more semantically related are the keywords to the text, the closer are their vectors representations.
To achieve this goal a first analysis was conducted on the performance of the raw MPNet architecture, without fine-tuning. Then we fine-tune the model and demonstrate the effectiveness of a contrastive learning approach in generating embedding representation of keywords that are semantically close to long-text description of the corresponding abstract.

Chapter 2

Related Works

In this section, we provide an in-depth exploration of the diverse methodologies employed in the creation of word embeddings, that play a crucial role in Semantic Similarity and, consequently, in Zero-Shot Text Classification.

2.1 Word2Vec

In the field of Natural Language Processing (NLP), representing words as indices in a vocabulary has been a simple and robust way to encode information, making it machine-readable. However, this approach lacks the ability to define properties like similarity between words, limiting its utility for many tasks. Furthermore, until the introduction of this technique, existing models struggled to handle large datasets efficiently, resulting in slower processing speeds.
In the realm of word vector representations, earlier research had already revealed that words exhibit various degrees of similarity. For instance, words with similar endings often clustered in the same subspace of the original vector space. Researchers also found that similarity extended beyond mere syntactic regularities; for example, they demonstrated that operations like vector(King) – vector(Man) + vector(Woman) produced a vector close to the Queen vector.
The paper by Mikolov et al. [11] aimed to develop a novel model that not only maximized the accuracy of vector operations like the one mentioned above, but also improved the learning of both syntactic and semantic regularities. To achieve these goals, they introduced two single-layer architectures: Continuous Bag of Words (CBOW) and Skip-gram.

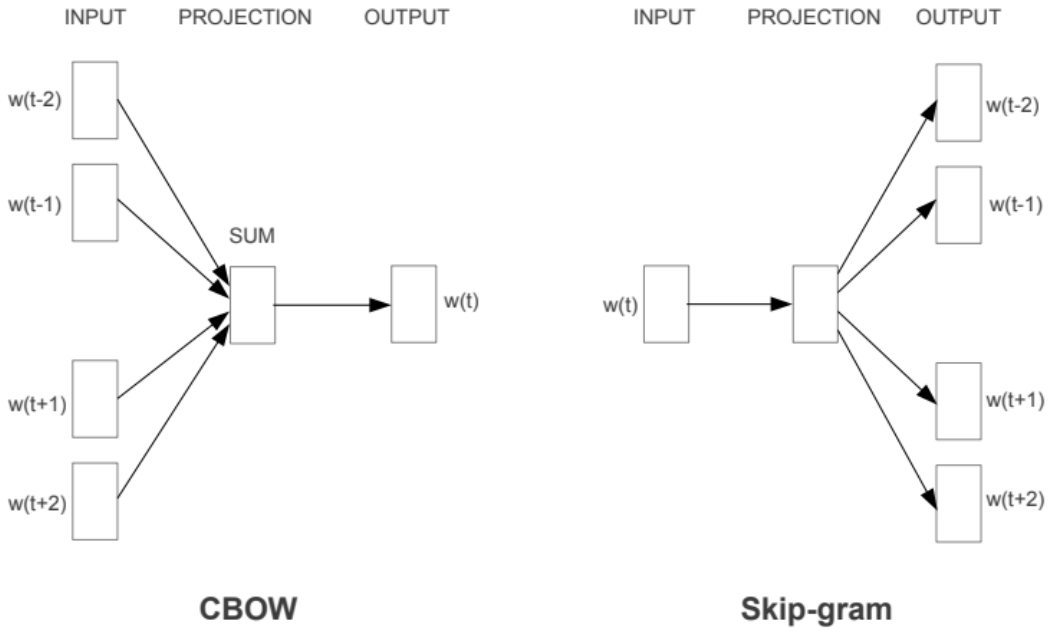

Figure 2.1. Schema of the 2 method proposed methods. The CBoW (left side) technique makes prediction for the current word based on the surrounding context, on the contrary Skip-gram (right side) model predicts the surrounding words given the current word. Image from [13]

2.1.1 Continuous Bag-of-Words (CBOW)

The CBOW method is a modality for learning Word2Vec embeddings that consists in training a simple feed forward network where it is removed the non-liner hidden layer and the projection layer is shared by all words. As results the projection is not influenced by the order in which words are presented. As shown in figure 2.1 the architecture, given a window of n words, attempts to predict the current word (in the middle) based on the other words that surround it, in essence by leveraging the context. In particular the architecture exploit both word from past as well as from future (the words before and after the current). It is called continuous because the representation learned, i.e. the word embeddings, are continuous dense representation of the original words. The window input consists of N future and N past words.

2.1.2 Skip-gram

The other method to learn Word2Vec embeddings is Skip-gram. As shown in 2.1, it works in the opposite way with respect to CBOW. Given a current word as input, the model predicts words before and after within a certain window range. It increase the quality of the generated vector but also the computational complexity. Moreover, inside a window, words are weighted based on their distance to the current (central) word, following the principle that the more words are distant the less they are related. In this context, the weight represents how often we sample a word and use them for the training. The architecture employs a range C, in particular, R words form past and R words form future are sampled in the range $< 1, C >$.

2.2 GloVe

The main drawback of Word2Vec methods is the inability to capture the global context. In general, they are not able to explicitly preserve global information about the statistics of word occurrences because they focus the attention only on the local context, a small portion (window) of the entire corpus.
For this reason they totally lost the advantage to consider the vast amount of data repetition within text or documents, fails in creating statistics on that. The overcome the aforementioned limitations, the authors of [13] propose

Global Vectors for Word Representations (GloVe), a global log-bilinear regression model that combines the advantages of both local context window methods and global matrix factorization.

Before going into detail about how the model works lets define some notation:

- X: word-word co-occurrence count matrix
- X_{ij} : contains the number of time the word j occurs in the context of word i
- $X_i = \sum_k X_{ik}$: number of recurrences of word i within the context of all the other word
- $P_{ij} = P(j|i) = X_{ij}/X_i$: probability that given the word i, the word j appears in the same context

From simple statistics about words co-occurrence it is possible to extract certain aspects of meaning. An example is proposed to explain better this concept and clarify in a simple way what are the core elements of the GloVe architecture. We can consider two word, i = ice and j = steam, and examine their relationship by studying how often they appear together with respect to different probe words k. In particular, by analyzing the ratio of co-occurrence probabilities. As shown in the table 2.2, for a given word k = solid we expect a strong relation with respect to ice (i) but not steam (j). As consequence, the ratio P_{ik}/P_{jk} will be large. On the contrary, for a word k = gas, the ratio should be small as the word has a stronger correlation with steam. Moreover, for k words like water or fashion, which in the first case are either related to ice and steam and in the second to neither, the ratio is close to 0, showing that this word are irrelevant in the discrimination between the words i, j.

In previous works training was based on probabilities co-occurrence. The innovation, as illustrated in the example, is to leverage ratio of co-occurrence probabilities instead.
Based on this principle the aim is to find an objective function that depends on the three words i,j,k that we can write in a generic form as

$$F(w_i, w_j, \tilde{w_k}) = P_{ik}/P_{jk}$$

where w represents word vectors and $\tilde{w} \in \mathbb{R}^d$ are separate word vectors. In fact during training the model generates and train two sets of word vectors W $\tilde{W} \in \mathbb{R}^d$. This two vectors differ only on their random initialization. The

Probability and Ratio	$k = solid$	$k = gas$	$k = water$	$k = fashion$
$P(k\|ice)$	1.9×10^{-4}	6.6×10^{-5}	3.0×10^{-3}	1.7×10^{-5}
$P(k\|steam)$	2.2×10^{-5}	7.8×10^{-4}	2.2×10^{-3}	1.8×10^{-5}
$P(k\|ice)/P(k\|steam)$	8.9	8.5×10^{-2}	1.36	0.96

Figure 2.2. The table presents a comparison of word occurrences for both 'ice' and 'steam' in relation to other words. Specifically, it analyzes the conditional probability of these two words and their ratio. It's evident that the ratio provides a better extrapolation of information regarding word relevance concerning the two specified words. In the raw ratio, a higher value indicates a stronger correlation with 'ice' in the numerator, while a lower value implies a stronger correlation with 'steam' in the denominator. A value close to one signifies that the word 'k' is not discriminative for either of the two words. Image from [13]

final word vectors is obtained by merging W $\tilde{W}$ through a simple addition. This ticks improve performances reducing overfitting and noise.

Given some consideration about linearity of the vector space it is possible to encode the information of the ratio P_{ik}/P_{jk} leveraging vector differences. Moreover to make both left and right sides compatible, since one represents vector and the other a scalar value, vectors can be combined through dot product. We derive the following function:

$$F((w_i, w_j)^T \tilde{w_k}) = P_{ik}/P_{jk}$$

Next, F must be homomorphic between $(\mathbb{R}, +)$ and $(\mathbb{R}_{>0}, \times)$. The solution is picked the exp as F function, applying then the logarithm; after some transformation and adding a bias $\tilde{b_k}$ for $\tilde{w_k}$ we obtain:

$$w_i^T \tilde{w_k} + b_i + \tilde{b_k} = \log X_{ik}$$

From the previous equation a weighted least squares regression is finally defined as:

$$J = \sum_{i,j=1}^{V} f(X_{ij})(w_i^T \tilde{w_j} + b_i + \tilde{b_j} - \log X_{ij})^2$$

were V is the size of the vocabulary and f is the weighting function that must respect the following properties:

- $f(0) = 0$. f must approach zero rapidly as x approaches 0.
- $f(x)$ should exhibit non-decreasing behavior giving excessive importance to infrequent co-occurrences
- $f(x)$ should be relatively small for large values of x, preventif that frequent co-occurrence are not overweighted.

The function, displayed in figure 2.3, that satisfies the above properties and works well can be parameterized as:

$$f(x) = \begin{cases} (x/x_{max})^\alpha & \text{if } x < x_{max} \\ 1 & \text{otherwise} \end{cases}$$

where α is an hyperparameter an is set to $\alpha = 3/4$, while x_{max}, fixed to 100, is the cutoff.

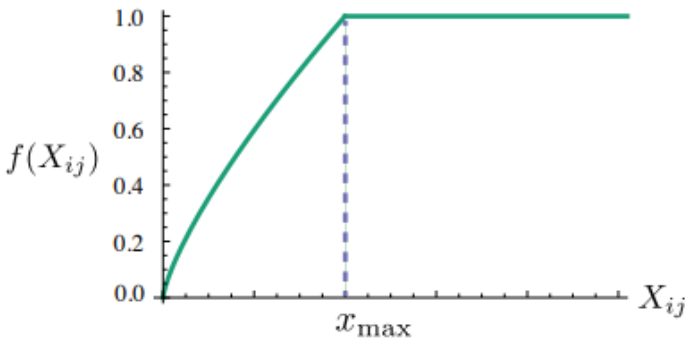

Figure 2.3. Weighting function f with $\alpha = 3/4$. Image from [13]

2.3 Transformers

Originally designed for language translation, the Transformer architecture [16] marked a paradigm shift, a revolution in the realm/landscape of machine learning. Transformers employ an attention mechanism, called Self-Attention, that learns contextual relationships between words within a text. It has become the basis for state-of-the-art models in virtually all NLP tasks. Moreover, Transformer architecture has been successfully employed to achieve the state of the art even into other domains such as Computer Vision, time-series forecasting, Reinforcement Leaning and many other machine learning fields.

2.3.1 Model Architecture

The model, as depicted in figure 2.4, is composed by 2 main blocks, encoder and decoder. The encoder takes as input a sequence of word (represented as embedding) and generates a series of continuous representation. Given the output of the encoder the decoder generates an a sequence of word (embedding) one element at a time. The model use the output of the previous step to generate the next so it's auto-regressive.

2.3.2 Encoder

Contains a stack of N identical sub-blocks. Each of which is composed by a multi-headed self-attention layer and a position wise fully connected feed forward layer. Moreover residual connection are introduced at the end of each sub-block, combined with a layer normalization.

2.3.3 Decoder

Similarly to the encoder, it is composed by M identical sub-blocks. Each sub-block introduces a layer that performs a multi-head attention using also the output of the encoder stack, in addition to the other two blocks appearing in the encoder. Likewise the encoder, residual connection and layer normalization are applied. The decoder's multi-head attention differs from the encoder in that its tasks is to predict the following token, and therefore it should be blind to future elements of the sequence that is trying to generate. To achieve this, it uses a masked input in order to do not take into consideration subsequent position. In this way the current element, at position i,

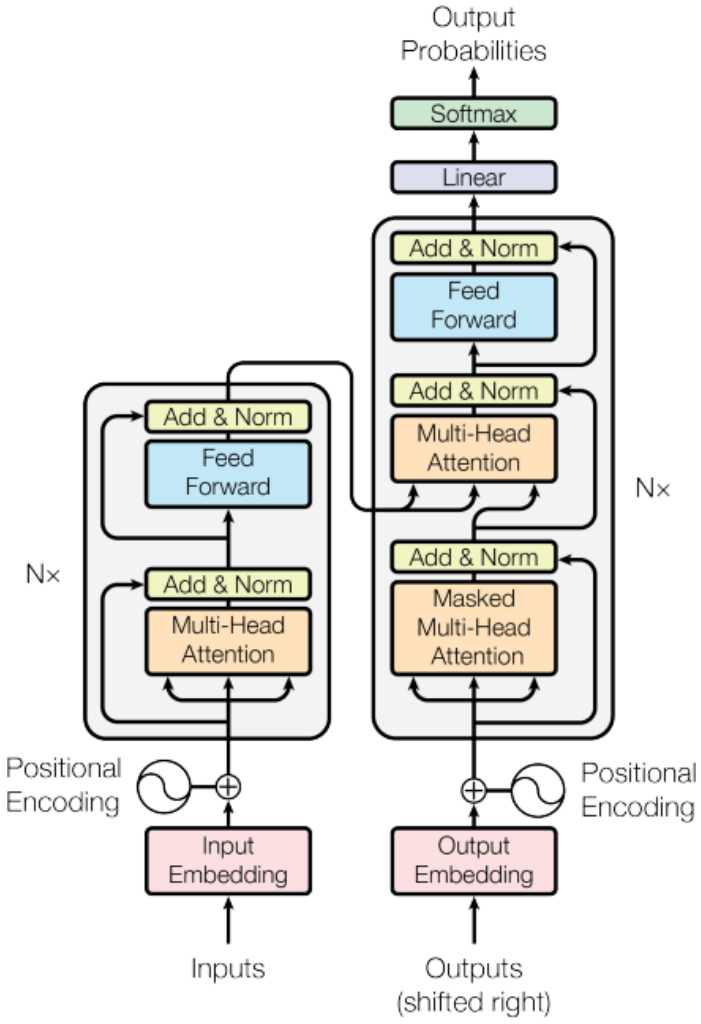

Figure 2.4. Transformers architecture. Image from [16]

depends only on all the previous elements but not on the following ones.

2.3.4 Positional Encoding

It is a straightforward but effective way of attaching the information about word positioning and, as consequence, distances between words. By adopting this approach, the complexity of understanding word positioning is moved from the structure of the network (which does not contain convolution neither recurrence) to the data itself. The concept is store word order as data, not structure. To add the information to the input, this vector is summed to the corresponding embedding as illustrated in figure 2.6 before injecting data into the network. The positional encoding has the same dimension of the embedding and it is generated using cosine and sine functions:

$$PE_{(pos,2i)} = \sin\left(pos/10000^{2i/d_{model}}\right)$$

$$PE_{(pos,2i+1)} = \cos\left(pos/10000^{2i/d_{model}}\right)$$

pos is the word's position, i is the dimension, while d_{model} is the embedding dimension. The choice of sine and cosine functions is favored for their versatility in handling sequence lengths greater than those encountered during training.

2.3.5 Self-Attention

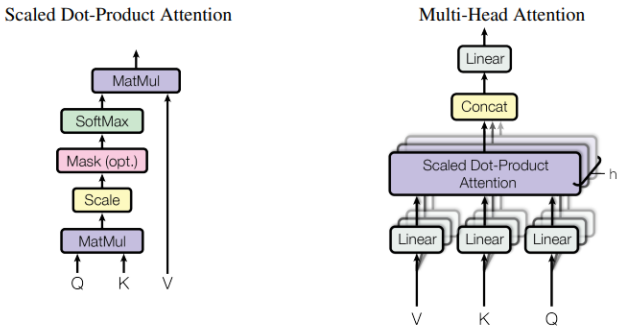

Figure 2.5. Scaled Dot-Product Attention (left side) and Multi-Head Attention (right side) that consists of multiple attention layers running in parallel. Image from [16]

Used to understand pattern languages or underling meaning. Thanks to this approach, the model is able to build up a better internal representation

of the language, as the classical conv network does with images. In fact, as well as the layer of a conv network recognize edges, eyes..., a self-attention head is able to recognize part of speech, syntactic or semantic rules, learning gender, plurality and other grammar and syntactic rules.

The approach adopted is called Scaled Dot-Product Attention.
As explained in [9], In the first step, from each of the inputs of the encoder (input are embedding) the model generate 3 vectors Key (K), Value (V), Query (Q), by multiplying the input embedding with each corresponding matrix Wk, Wv, Wq that are trained during the training process. The second step consists of calculating a score for each word with respect to all the other words in the sentence. The score determines given a word encoded in a certain position, how much focus is placed on other parts of the input sentence. Given a word, for example the word in position 1, we compute the dot product between Q1 (of a fixed word) and the K vectors of the other sentence words (K1 x Q1, K1 x Q2 ... K1 x Qn). Then divide by the square root of the dimension of the K vectors (dk) to have a more stable gradient and, after that, pass the results through a softmax to normalize the scores. The softmax score plays a crucial role in determining the level of expression for each word at this particular position.

Then multiply for each V vector the respective softmax score (in order to make less relevant value vectors with lower scores) and finally sum up the weighted value vector. This produces the output value for the first word (Z1 = V1 x Score1 + V2 x Score2 + ... + Vn x Scoren). All these steps are done in a matrix form to speed up calculation. This leads to the following expression:

$$Attention(Q, K, V) = sofmax(\frac{QK^T}{\sqrt{d_k}})V$$

2.3.6 Multi-headed Attention

Expand the model's ability to focus on different positions, instead the simple attention in which the main focus is on the position of the word itself. Thanks to the multi-head system, each head creates a different representation subspace. And as the kernel is done, each head learns a particular pattern, focusing on a specific feature to learn, such as different grammar or syntactic/semantic rules.

$$MultiHead(Q, K, V) = Concat(head - 1, ..., head_h)W^O$$

$$\text{where } head_i = Attention(QW_i^Q, KW_i^K, VW_i^V)$$

The attention is calculated by the different heads ($head_i$) that are randomly initialized and different from each other. After computing the $h = 8$ *Attention* vectors, they are merged through a concatenation and finally multiplied by a W^O matrix, in order to be reshaped and fit the feed-forward layer.

2.4 BERT

Published by Google AI in 2018 [6] represents a key point in the NLP scenario. It set a new standard for state of the art results for a wide variety of NLP tasks such as Question Answering (SQuAD), Natural Language Inference (MNLI) and many others. The powerful thing is the versatility of the model that, with small adjustments, is able through transfer learning (fine-tuning) to obtain great performances on a variety of different tasks.

Previous training language models used a directional approach in the next sentence prediction goals. This method intrinsically limits the context learning, since the model has no knowledge about the whole sentence. BERT was designed to get through this problem by exploiting two pre-training strategies: MLM and NSP.

2.4.1 Input

As shown in the figure 2.6 before being fed to the model, the input is processed to handle representations for both single or pairs of sentences based on the task. The input representation is constructed by combining 3 parts:

- About token embeddings, a [CLS] token is inserted at the beginning of the sequence. It is a special classification token that aggregates sequence representation, useful for classification tasks. Moreover, a [SEP] token is placed at the end of each sentence, such as a delimiter.

- A sentence embedding is added to each token, indicating which sentence each word belongs to.

- As in the Transformers framework, positional embeddings are added to each token to indicate its position in the sequence.

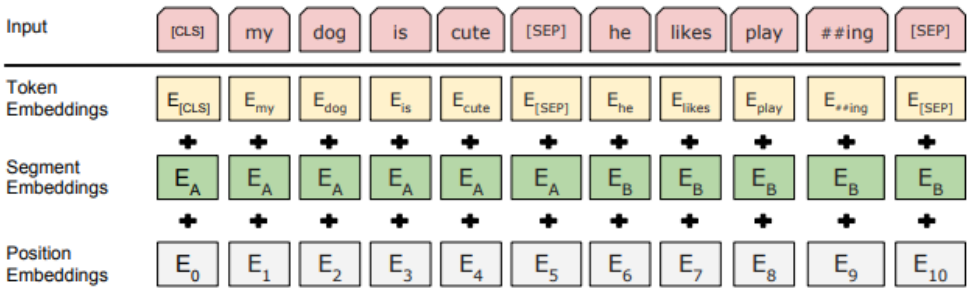

Figure 2.6. BERT input representation. The input is constructed by combining the token embeddings, segmentation embeddings, and position embeddings through summation. Image from [6]

2.4.2 Masked Language Model (MLM)

In order to train a deep bidirectional representation , a percentage of the words (15%) fed as input to BERT are replaced with a [MASK] token.

Then the model's aim is to predict the original value of the masked word by leveraging the context provided by remaining non-masked words in the sequence. The task of word prediction is the same as the basic Transformer architecture. However, the BERT Cross Entropy loss function does not consider the entire set of predictions, but rather focuses solely on the prediction of the masked words.

Moreover, the 15% tokens picked to be masked are not always replaced with the [MASK] token. This happens 80% of the time. In 10% of cases, it is substituted by a random token, while in the residual 10% the token remains unaltered. This technique mitigates mismatch problems with the fine-tuning step, creating a more robust framework.

From a technical point of view, predicting the output words necessitates:

1. Add on top of the encoder output a classification layer
2. Adjusting the output vector dimension to match the vocabulary size (multiplying the output vectors by the embedding matrix)
3. Computing, though a sofmax function, the probability of each word in the vocabulary

Figure 2.7 provides a schema of the previously explained technique.

2.4.3 Next Sentence Prediction (NSP)

Since some important downstream tasks are based on understanding the relationship between two sentences, the idea was to pre-train the model to solve a binary next sentence prediction task. Specifically, the architecture receives as input pairs of sentences and learn to predict the correlation between them. Concretely, given two sentences A and B, it learns the probability of B being the subsequent sentence that follows A. During training, 50% of the time pairs are related (labeled as IsNext) while for the other 50% the second sentence is drawn randomly from the corpus, in order to obtain an uncorrelated pair (labeled as NotNext). The main steps of this phase are the following one:

1. The input sequence goes through the architecture.

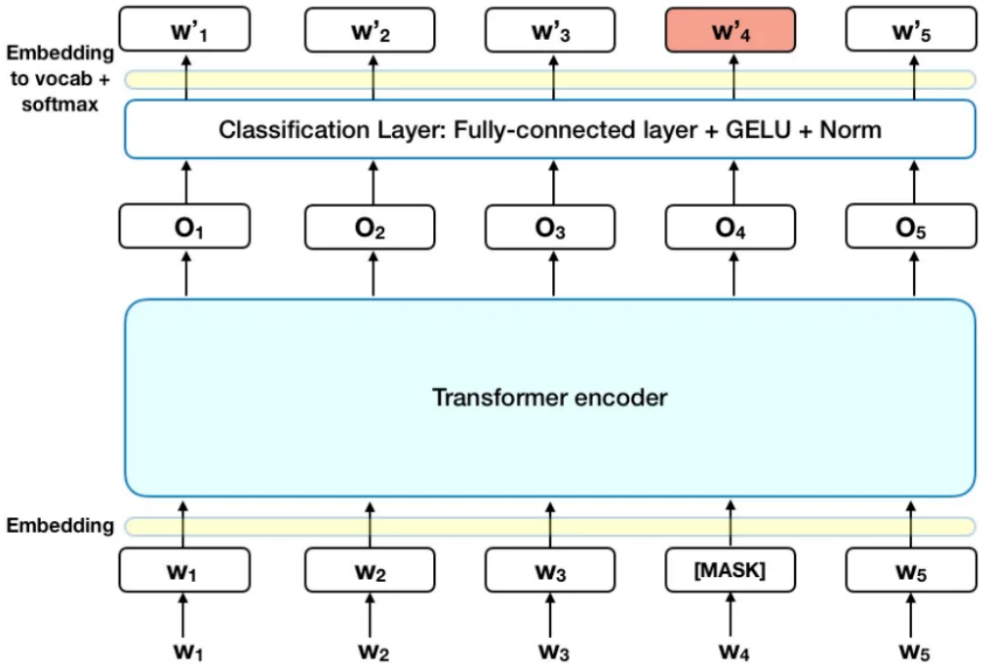

Figure 2.7. BERT Masked Language Model (MLM). Image from [4]

2. The output of the [CLS] token is transformed into a 2x1 shape vector (true and false value scores) using a classification layer.

3. The probability of IsNextSequence is computed with a softmax.

2.5 MPNet

Despite its powerful and effective on a lot of natural language tasks, BERT suffers from some drawbacks. MLM can perceive the positional information of the full sentence, however it is not able to capture the dependency among the predicted tokens. To address this problem, XLNet [18], an auto-regressive language model based on Transformes, introduces Permuted Language Modeling (PLM). This approach consist of a random permutation of the sequence. The permuted sequence is split into a left non-predicted part and a right predicted part. Then tokens on the right side are predicted in an auto-regressive way. For example, as shown in the figure 2.8(b), given a sequence $x = (x_1, x_2, x_3, x_4, x_5)$, the permuted $x = (x_1, x_3, x_5, x_2, x_4)$ is generated. Next, PLM predicts x_2 and x_4 auto-regressevely conditioned on (x_1, x_3, x_5). While this technique can capture the dependencies among the predicted tokens, it is unable to leverage the positional information of a sentence, which induces a mismatch between pre-training and fine-tuning. The objective of MPNet (Masked and Permuted language model) [14] is to establish a unified model that inherits the advantages of both MLM and PLM avoiding their weaknesses and limitations, and built a stronger pre-trained model.

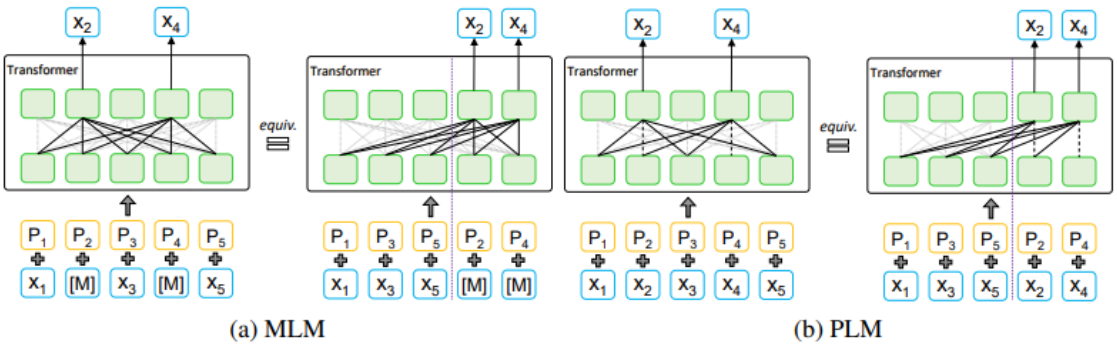

Figure 2.8. MLM (a) and PLM (b) models. Image from [14]

2.6 Contrastive learning

Contrastive learning is a versatile technique employed in both computer vision and natural language processing tasks. Its core objective is to learn representations of data samples (in our context sentences or words), in a manner that brings similar samples closer together in the vector space, while simultaneously pushing dissimilar ones further apart.
Particularly, the fundamental concept, illustrated in 2.9, revolves around the process of convergence between an anchor, a reference sample, and "positive" samples in the embedding space, while simultaneously creating separation between the anchor and multiple "negative" samples. On an unsupervised task, given the absence of explicit labels, a positive pair typically comprises augmented versions of the same sample, while negative pairs are constructed by pairing the anchor with randomly chosen samples from the batch, for this reason, it is often referred to as self-supervised training. Notably, some studies [10, 8] introduce a supervised learning approach, which extends the principles of the contrastive self-supervised literature by incorporating the valuable label information at hand.
Leveraging this techninque, as illustrated in 2.10, positives are drawn from samples of the same class as the anchor, in contrast to being generated as data augmentations of the anchor, as is typically done in self-supervised learning. Supervised contrastive learning is especially advantageous in NLP. In fact in computer vision tasks, the augmentation, such as crop, flip or rotation, almost always preserves the image structure. On the other hand, in NLP, since we deal with sentences, a sequence can be easily distorted during the augmentation phase. The exploitation of labels avoid any issues of that nature.

In our context, the supervised contrastive learning setup is achieved by considering, for each document, the abstract as anchor, the noun phrases extracted from the relative title as positive samples and the remaining are labeled as negatives.

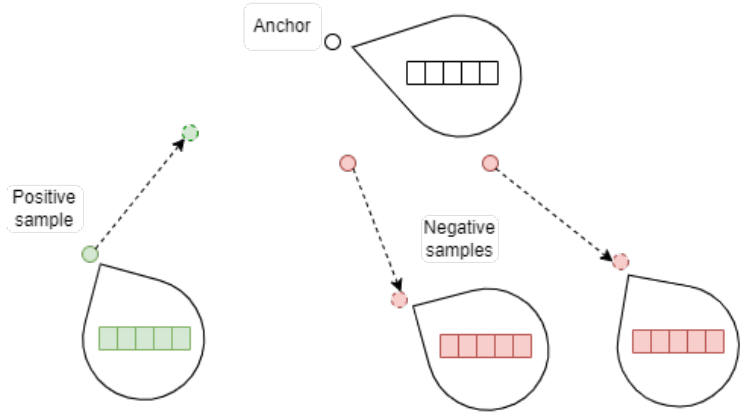

Figure 2.9. A visual representation that elucidates that explains how contrastive learning works. Circles represent embeddings within a vector space. Positive/entiled samples (in green), are bring close the anchor (the white circle), while negative samples (in red), are pushed away. The dashed circles represent the final (and ideal) position of embeddings after the model's training leveraging contrastive learning

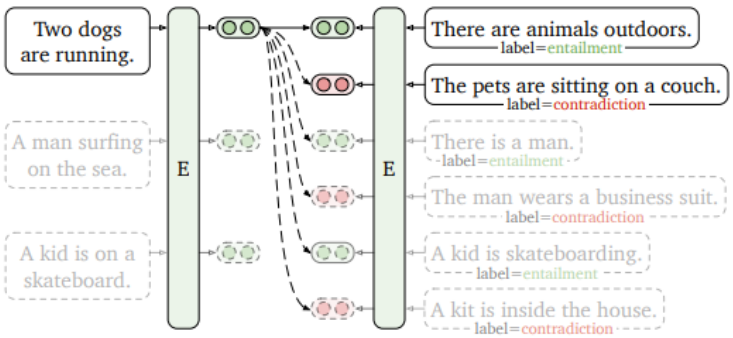

Figure 2.10. Image that better illustrates the supervised contrastive learning. For each anchor (left side), a set of other samples (right side) are drawn from the dataset, forming pairs (anchor, sample to compare with). Entailed phrases (in green) are labeled as positive samples, while contradictory one (in red) are marked as negative samples. Image from [8]

Chapter 3

Dataset

3.1 ArXiv

ArXiv, established in 1991 by Paul Ginsparg and currently maintained and operated by Cornell University, stands as a remarkable testament to collaborative funding and community support. Over nearly three decades, ArXiv has steadfastly served both the public and the global research communities by offering unrestricted access to an extensive array of scholarly articles. Its comprehensive coverage spans a vast spectrum, encompassing the diverse realms of physics, the multifaceted subdisciplines within computer science, and a plethora of other academic domains. Whether one's interests lie in mathematics, statistics, electrical engineering, quantitative biology, economics, or the myriad fields that fall in between, ArXiv provides an invaluable repository of knowledge. The richness and depth of this corpus of information are truly unparalleled. However, this extensive wealth of knowledge can, at times, prove daunting and overwhelming to navigate. To address this challenge and enhance accessibility, a dedicated repository has been meticulously curated on Kaggle [2]. This repository, constantly updated, comprises 1.7 million papers, each tagged with relevant features that facilitate easy discovery, exploration and manipulation. These features include article titles, author information, category classifications, full-text PDFs, and a wealth of additional metadata.

The repository's objective is to generate a more manageable dataset, that can serve a diverse range of machine learning tasks.

The ArXiv papers have a hierarchical category classification structure. There are 8 main categories: Computer Science, Economics, Electrical Engineering and Systems Science, Mathematics, Physics, Quantitative Biology,

Quantitative Finance, Statistics. Each macro-category group different categories as shown in the figure:

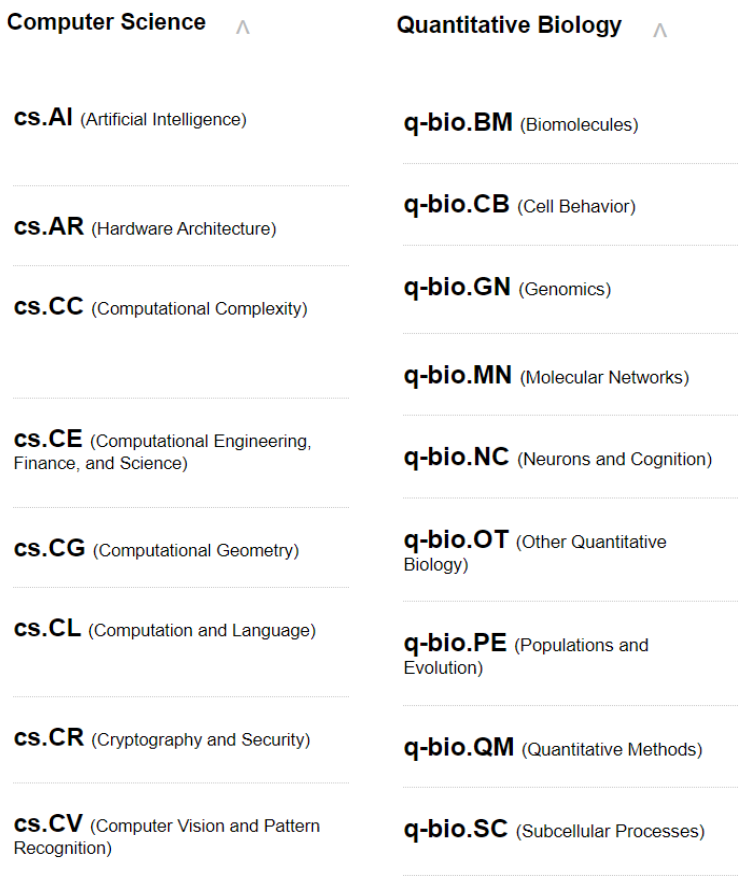

Figure 3.1. Example of hierarchical taxonomy for the macro-classes Computer science and Quantitative Biology. Snapshot from [3]

Each paper belongs to one ore more categories, depending on the faced argument.
As already mentioned the dataset contains a lot of information for each paper. For our purpose the following properties are extracted:

- title: title of the paper
- noun phrases: keywords extracted from the title
- abstract: abstract of the paper, is a summary that explain the paper's work
- categories: list of categories, topics covered/ field of interest of the paper

3.2 AG News

AG is a collection composed of 1 million news articles gathered from more than 2000 different news sources. The information were gathered in more than 1 year through ComeToMyHead [5], an academic news search engine. The AG news classification dataset [1], exploited in this work, is created by Xiang Zhang as text classification benchmark for the paper [20].

The dataset is constructed by choosing 4 largest classes: "World", "Sports", "Business", and "Science". It comprises a total of 120,000 training samples and 7,600 testing samples, Samples are equally balanced across the classes. Specifically, each class includes 30,000 training samples and 1,900 testing samples, ensuring equitable representation for robust evaluation.
Clearly, within this context, our primary focus is not on class classification. However, given inherent structure of the dataset, it seamlessly aligns with our specific objectives.

- title: title of the article
- noun phrases: keywords extracted from the title
- description: article describing the news in more detail
- category: general topic of the article (can be: World, Sports, Business, or Science)

To partially clean the dataset from noise, meaningless words such as the journal names that frequently appear at the end of titles or articles are filtered out.

3.3 Noun phrases extraction

Our focus lies in optimizing the creation of embeddings for short sequences of words. To achieve this, we employ a preprocessing step involving the extraction of noun phrases from the paper titles. This process is facilitated by the utilization of the spaCy library [15], an open-source natural language processing (NLP) tool for Python. It is designed to perform various NLP tasks such as tokenization, part-of-speech tagging, named entity recognition, syntactic parsing, and more. It is also known for its speed and efficiency, making it a popular choice among developers and researchers for processing

and analyzing text data.
Within this context, we leverage spaCy to extract noun phrases from the titles. Subsequently, we filter out stop words such as articles from the beginning to the end of each noun phrase. This step is finalized at reducing noise within the data, by eliminating irrelevant words. Moreover, an additional strategy to reduce noise consists in deliberately discard single words.
This decision is driven by the understanding that such isolated terms may lack the necessary contextualization and, instead of contributing valuable information, could introduce further noise. Samples without valid noun phrases are discarded from the dataset.

Below an example is shown:

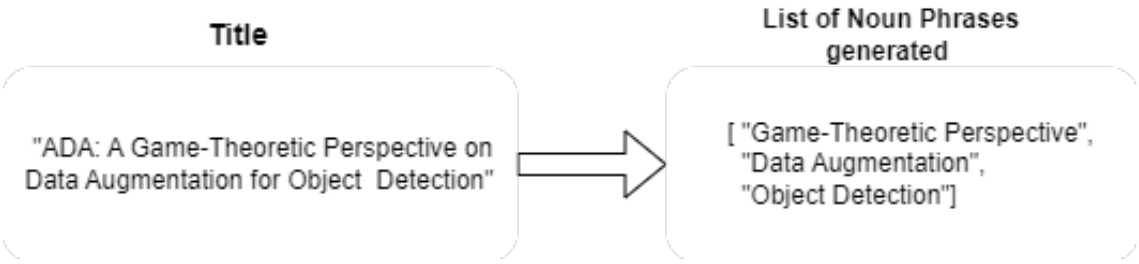

Figure 3.2. From the title, through spaCy, 4 noun phrases are extracted: "ADA", "Game-Theoretic Perspective", Data Augmentation", "Object Detection". Since the single word "ADA" is meaningless by itself, it is discarded.

Chapter 4

Methods and metrics

4.1 Problem Statement

The objective of this study is to create a framework to enhance the Zero-Shot Text Classification capabilities of deep Language Models by creating an appropriate self-supervised task, from which the model can learn without humanly annotated labels, and by harnessing the power of contrastive learning.
In particular, we fine-tune the MPNet deep Language Model to address this task, evaluating its performance on both the ArXiv and AGNews datasets. We explore different data subdivisions 5.2, from a less restrictive setup to one that closely mirrors real-world conditions.
The primary goal is to refine the generation of vector representations for keywords, ensuring they are semantically aligned with the associated text embeddings.
Our main contributions are summarized as follows:

- setting up a framework for self-supervised fine-tuning of a deep Language Model for the task of Zero-Shot Text Classification, in 3.3 and 4.5;
- introducing a weighted normalization approach for the objective function, detailed in 4.2;
- conducting an in-depth analysis of the spatial distribution of the generated embeddings within the vector space, as discussed in 4.3;
- assessing our framework's effectiveness in a category classification task, thereby demonstrating its versatility in another Zero-Shot Text Classification scenario where the model has to categorize text according to an

unseen taxonomy, in 5.3.4.

4.2 Weighted Normalized Supervised Contrastive Loss

Taking into account the concepts exposed in 2.6, we take inspiration from the contrastive loss used in the literature [10],[8] and tailor it to our specific needs. Particularly, our custom loss differs from the previously mentioned one as it remains invariant to the number of negative samples. This is because, in this context, we do not have knowledge a priori of the exact number of negative samples for each anchor in every batch. The relevance of negative samples, as we will discuss shortly, is regulated by the hyperparameter α.
The objective function that we want to optimize given a sample x_i considered as anchor is:

$$l_i = -\log \frac{\frac{1}{N_i^+} \sum_j^{N_i^+} e^{sim(x_i, x_{i,j}^+)/\tau}}{\frac{1}{N_i^+} \sum_j^{N_i^+} e^{sim(x_i, x_{i,j}^+)/\tau} + \alpha \cdot \frac{1}{N_i^-} \sum_j^{N_i^-} e^{sim(x_i, x_{i,j}^-)/\tau}}$$

Were τ is a temperature value, sim represent the cosine similarity between two embeddings, x_i is the anchor, $x_{i,j}^+$ are positive samples while , $x_{i,j}^-$ are the negative one, finally α is an hyperparameter that balances the weight of negative samples with respect the positive one.
This loss is computed for each anchor, the final results is obtained by calculating the mean value.

4.3 Alignment and Uniformity

Both properties serve as quality metrics for assessing the goodness of the generated embeddings as explained in [17]

Alignment guarantees that in a positive pair, two samples are mapped to closely located features. It calculates the expected distance between positive embeddings.

$$l_{align} = \mathbb{E}_{x,x^+ \sim p_{pos}} \|f(x) - f(x^+)\|^2$$

where p_{pos} denotes the distribution of positive pairs.

Uniformity measures how uniformly are distributed embeddings within an hypersphere space, ensuring the preservation of as much information of the data as possible.

$$l_{uniform} = \log \mathbb{E}_{x,y\sim\ p_{data}} e^{-2\|f(x)-f(x^+)\|^2}$$

where p_{data} represents the distribution over all data (both positives and negatives), $e^{-2\|f(x)-f(y)\|^2}$ is the Gaussian potential kernel, in particular the Radial Basis Kernel.

Summarizing, alignment is minimized by that assigning similar features to similar samples, while uniformity promotes the feature being distributed over the whole semantic vector space, rather than collapsing into a narrow subspace. The figure 4.1 better exposes and visualizes this concepts.

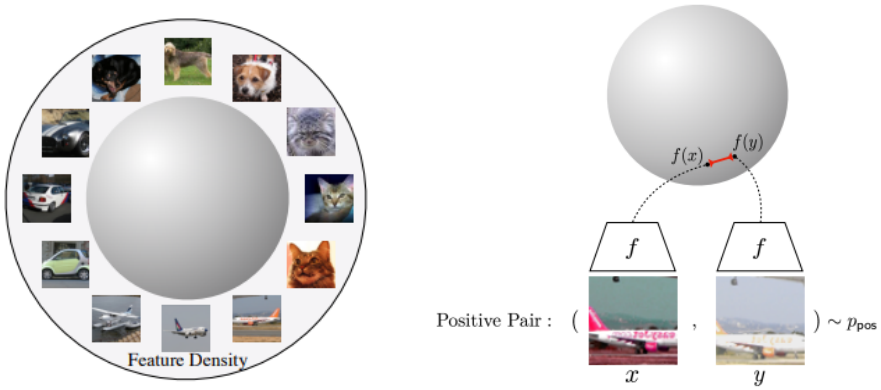

Figure 4.1. Visualization of the Alignment (on the right) and Uniformity (on the left) of feature distributions on the output unit hypersphere. The image illustrates an ideal scenario where, concerning Uniformity, samples exhibit a well-distributed pattern across the hypersphere. Regarding Alignment, samples with similar features are positioned closely within the feature space. Image from [17]

Recent studies, such as [7], have identified an anisotropy problem in the semantic vector spaces learned by deep Language Models, that is naturally connected to uniformity. In particular, these models learn and generate embeddings that occupy a narrow cone in the vector space. This is thought to limit their expressiveness and consequently affect the performance on the semantic similarity task. While numerous approaches have been suggested to address this issue, our primary emphasis lies in the investigation of the contrastive setup.

4.4 Cosine Similarity

Cosine similarity is a metric that evaluates the similarity between two vectors defined within a linear vector space. It is essentially the cosine of the angle formed by these vectors, calculated as the dot product of the vectors divided by the product of their L2 norm:

$$sim(x_1, x_2) = \frac{x_1 \cdot x_2}{\|x_1\|_2 \cdot \|x_2\|_2}$$

It is important to highlight that cosine similarity is solely determined by the angle between the vectors, not by their norm. This metrics consistently falls within the range of $[-1, 1]$.
In this study cosine similarity is employed to measure distance between embeddings. The more the value approaches to 1 the closer are embeddings in the vector space, indicating, in our context, a higher degree of semantic similarity.

4.5 Normalized Discounted Cumulative Gain (nDCG)

Normalized Discounted Cumulative Gain (nDCG) is a measure of ranking quality, used in information retrieval or recommendation system. Given a query and a set of items provides an evaluation of a ranking algorithm in presenting relevant items in the ranking list.
In our context, the query is represented by an abstract while the items are represented by a set of noun phrases/keywords/labels some related to the abstract (i.e., extracted from the title relative to that abstract), and some unrelated (i.e., extracted from titles relative to different abstracts).
These noun-phrases are ranked based on the cosine similarity of their semantic embeddings with respect to the given abstract embedding. We, then, look at the position in the ranking of the noun-phrases that are related to the abstract and the higher their position the better $nDCG$ score is obtained.

$$DCG = \sum_{i=1}^{n} \frac{rel_i}{\log_2(i+1)}$$

Where n is the number of samples taking into account, rel_i is the relevance of the sample at position i, it is 1 for a related noun-phrase and 0 for an unrelated one. As we move down to the ranking positions, the contribution

that a related noun-phrases gives to DCG decreases, meaning that DCG is highest when all the noun-phrases relevant to a given abstract are ranked in the top positions.

$$nDCG = \frac{DCG}{IDCG}$$

$nDCG$ is calculated by dividing the discounted cumulative gain (DCG) by the $IDCG$. Ideal DCG establish the ideal order of samples. It represent the best possible value of DCG of the ideal ranked list for a query. $nDCG$ varies between 0 and 1, greater values correspond to better better-ranked list and as consequence better performance.

4.6 Precision@K, Recall@K, F1@K score

This set of metrics provides an alternative way to evaluating the model's performance in a classification task where each sample is associated with more then one label. We adopt a specific variant of Precision and Recall metrics, that are commonly employed to measure the effectiveness of recommendation system, i.e. Precision@K and Recall@K.
In essence, these metrics involve sorting the labels based on a specific criterion, in this context the cosine similarity between noun phrases and abstracts. Subsequently, the top K elements are designated as recommended, representing predictions of relevance made by the model. Notably, this metrics share similarities with the nDCG.
First of all it is important to clarify our criteria for defining a sample as relevant. For our purpose, a sample, better a keywords, is considered relevant when it is associated to the corresponding abstract's paper.

We define the Precision@K as the proportion of recommended samples in the top-K set that are relevant:

$$p@K = Precision@K = \frac{\text{\# of our recommendation that are relevant}}{\text{\# of items we recommended}}$$

The Recall@K is the proportion of relevant samples founded within the top-K recommendations:

$$r@K = Recall@K = \frac{\text{\# of our recommendation that are relevant}}{\text{\# of all the possible relevant items}}$$

The F1@K metric is a combination of the previous metric:

$$F1@K = 2\frac{p@K \cdot r@K}{p@K + r@K}$$

4.7 Model architecture

The main architecture, as depicted in 4.2, is quite simple. Both noun phrases and abstracts are input into the frozen MPNet network, the pre-trained model that generates embeddings for both long and short text sequence. On the noun phrases side, an additional feed forward layer is introduced and trained on the top of the model.
This layer remap each embedding through a series of linear and non-linear transformations, ultimately producing a new vector representation. Subsequently, both the long text (abstract) and short text (keyword) embeddings are employed in the computation of the contrastive loss objective.

Throughout the training process, the FFN undergoes updates with each epoch, progressively refining the vector representation for short texts pushing them closer to the respective abstract and contributing to the enhancement of the model's performance.

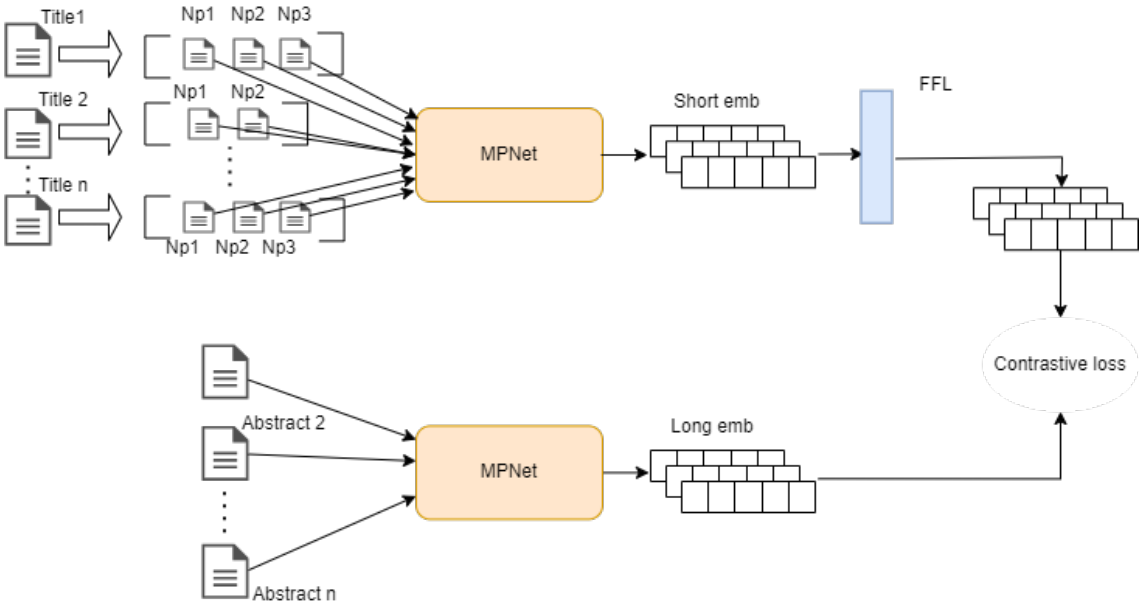

Figure 4.2. pipeline used

Chapter 5

Experiments and Results

5.1 Hyperparameter settings

We implement the pipeline described above leveraging the pre-trained MPNet, available through the Hugging Face platform [12].
After an exhaustive exploration of various configurations, our final decision for the Feed Forward Layer led us to a straightforward linear layer, characterized by an input and output size set at 768, aligned with the output dimensions of the MPNet model.
To ensure an efficient and effective training of the model, early stopping is employed. This technique basically stops training when there is no substantial improvement in performance, calculated on a validation set, over a set of consecutive epochs. Our experimentation revealed that, under optimal conditions, 10 training epochs are quite sufficient to achieve significant enhancements in model performance, avoiding the risk of overfitting.
The model is fine-tuned exploiting Stochastic Gradient Descent (SGD), with a momentum of 0.9, a weight decay set to 0.01 and a learning rate of 0.001. Further considerations led us to the determination of an optimal batch size, setting on a value of 128.
As better explained in the subsequent section 5.3.2, the optimal values for the crucial hyper-parameter alpha is explored, identifying a balanced compromise with a value of 200.

5.2 Dataset subdivision

Random split: Simply subdividing random the dataset without taking into account any categories subdivision. As results sample from a category in the train set may appear in the test set. It is important to underling that we compute and evaluate the metric just on noun phrases, as consequence the task remain a Zero-Shot Text Classification. (heterogenous dataset)
Unseen category: For this dataset subdivision a category is isolated, and samples belonging from that category are drawn/selected/picked as test test. Since in the arxiv dataset papers may have different categories belonging also to different discipline/subject (aka macro-category) the dataset split is constructed taking care of removing samples that contains at least another macro-category that is different from the one we want to isolate. This is the worst scenario, since test samples belongs to the isolated category the model do not ever see also a similar terminology.

5.3 Results

In this initial section, our attention is directed towards a relatively less restrictive scenario. We specifically consider the dataset's random split division, trying to extract valuable insights and evaluating the architecture's efficacy.

5.3.1 Uniformity and Alignment analysis

As discussed in the preceding chapter 4.3, uniformity and alignment metrics prove to be valuable measures in contrastive learning, offering a reliable instrument to assess the model's effectiveness in the embeddings generation.
Going into detail, alignment calculates expected distance between embeddings of noun phrases with respect to the corresponding abstract embedding. On the other hand, uniformity measures how well the embeddings are uniformly distributed within the vector space.
Specifically, concerning uniformity, we introduce three distinct measures:

- Uniformity Short: Focusing on uniformity between short phrase embeddings, also known as noun phrases.
- Uniformity Long: Concentrating on uniformity among long phrase embeddings, specifically the abstracts.

- Uniformity Cross: Evaluating uniformity between short and long phrase embeddings.

Since all these measures are designed as losses, the lower are values the better they are.

The subsequent plots provide a visual representation of the evolution of these properties across epochs. Notably, the loss 5.1 experiences a rapid decline during the first epochs, followed by a more gradual decrease that reach a sort of plateau after the 10th epoch.
The alignment 5.2 exhibits a decreasing trend, following the behavior of the contrastive loss 5.1, in contrast the uniformity metrics 5.3, 5.4 present an ascending trajectory as the epochs progress.
It is notable that the behavior of uniformity stands in stark contrast to that one of alignment. This observation suggests that the model and in particular the designed contrastive loss, primarily focuses on the task of bringing close associated samples together, with less emphasis on maintaining a uniform data distribution.

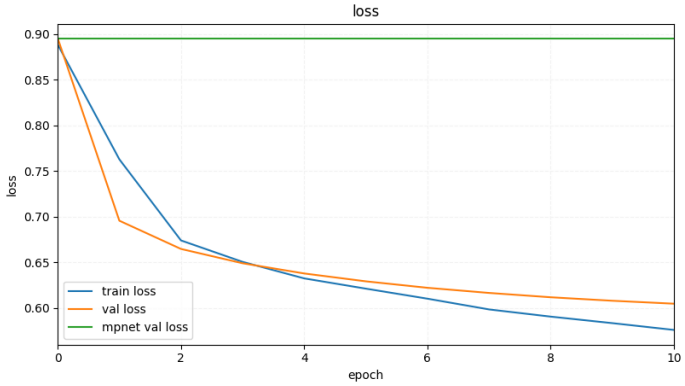

Figure 5.1. Plot of Loss, with epochs on the x-axis and loss values on the y-axis. The horizontal green line corresponds to the value computed through MPNet without fine-tuning, the blue line represents the value calculated on the training set, while the orange line reflects the value determined on the validation set.

Table 5.1 provides a detailed overview of the metrics, including numerical values, both before and after the model's fine-tuning process.
Remarkably, the MPNet architecture, even without fine-tuning, consistently achieves remarkable values on both ArXiv and AG News datasets.

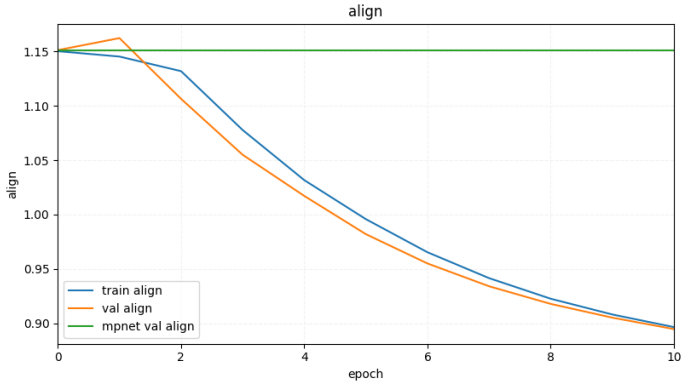

Figure 5.2. Plot of Alignment, with epochs on the x-axis and alignment values on the y-axis. The horizontal green line corresponds to the value computed through MPNet without fine-tuning, the blue line represents the value calculated on the training set, while the orange line reflects the value determined on the validation set.

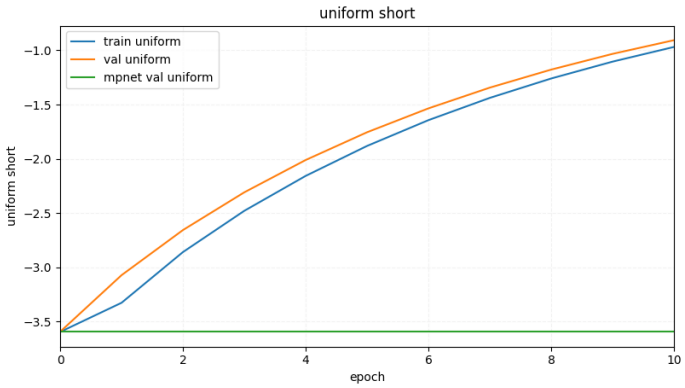

Figure 5.3. Plot of Uniformity short, with epochs on the x-axis and uniformity values on the y-axis. The horizontal green line corresponds to the value computed through MPNet without fine-tuning, the blue line represents the value calculated on the training set, while the orange line reflects the value determined on the validation set.

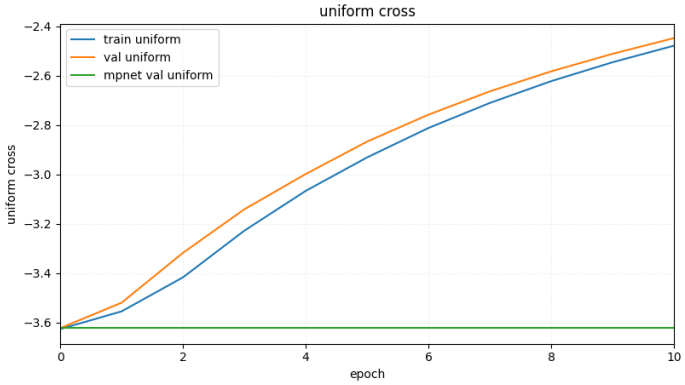

Figure 5.4. Plot of Uniformity cross, with epochs on the x-axis and uniformity values on the y-axis. The horizontal green line corresponds to the value computed through MPNet without fine-tuning, the blue line represents the value calculated on the training set, while the orange line reflects the value determined on the validation set.

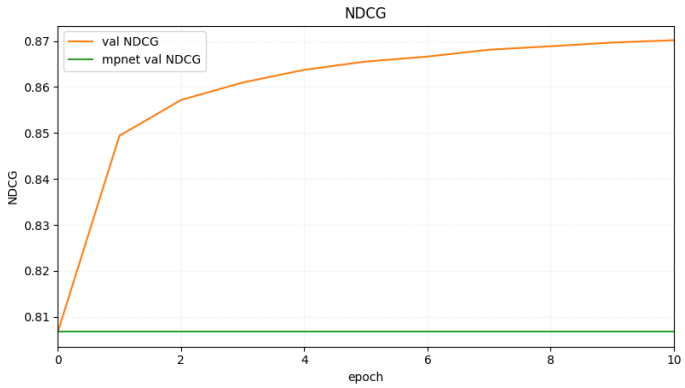

Figure 5.5. Plot of nDCG, with epochs on the x-axis and nDCG values on the y-axis. The horizontal green line corresponds to the value computed through MPNet without fine-tuning, the blue line represents the value calculated on the training set, while the orange line reflects the value determined on the validation set.

After the fine-tuning phase, we observed a diminishing in the alignment. This reduction implies that the embeddings of noun phrases are effectively brought closer to their respective abstract representations.
However, both cross and short uniformity metrics exhibit an increase. This phenomenon may be attributed to the issues related to anisotropy, as previously explained in 4.3. In particular, during the fine-tuning process, the samples are remapped into a narrow cone space, thereby reducing uniformity. It's important to note that, while relevant, this topic falls outside the scope of our current study, as our focus is directed towards other aspects.
Lastly, just for completeness, we also present the uniformity long metric. Obviously since the embeddings for the abstract sentences are not remapped, the uniformity value remains unaltered through fine-tuning process.

Metrics	MPNet raw	Fine-tuned model
ArXiv		
Alignment	1.151	**0.895**
Uniformity shorts	**-3.592**	-0.907
Uniformity cross	**-3.622**	-2.447
Uniformity long	-3.493	-3.493
AG News		
Alignment	1.213	**0.879**
Uniformity shorts	**-3.535**	-0.99
Uniformity cross	**-3.724**	-2.485
Uniformity long	-3.74	-3.74

Table 5.1. Alignment and Uniformity analysis on both Arxiv and Ag News dataset. We can notice a similar behaiviour across datasets. In both cases, Alignment improves, while Cross and Shorts Uniformity degrade. Since these metrics are losses lower values indicate better performance.

5.3.2 Contrastive hyperparameter α

As already explained in the previous section 4.2, the alpha parameter choice plays a critical role in the behavior of our framework, since it balances the weights of negative samples with respect to positive ones in the contrastive loss.

At approximately 60, the improvement levels off, and after 300 further

increases in the value leads to a slight degradation in performance. After an exploration of various hyperparameter configurations, we concluded that an alpha value of 200 best suits our requirements.

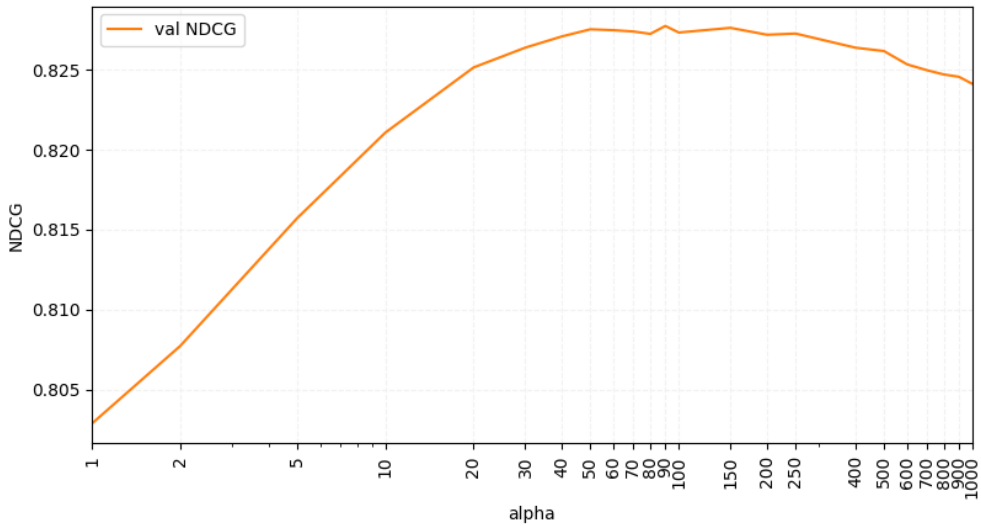

Figure 5.6. Evaluation of nDCG (y-axis) on the variation of alpha (x-axis). The plot shows an increase in nDCG values, reaching a stable value around 60. Further increases of alpha beyond 300 do not yield benefits, leading, in fact, to a decrease in performance.

5.3.3 nDCG Results

First of all we analyze the model behaviour on a simpler situation, leveraging a model trained and tested on an heterogeneous dataset.
The bar chart illustrates the results of nDCG over the test set of both arXiv and AG News datasets. We can noticed as we already seen analyzing the alingment and uniformity metrics that the base MPNet performs quite well on the embedding generation, obtaining an already higher classification score (based on nDCG). Overall we obtain an improvement of performance after the fine-tuning phase for both datasets. This show the effectiveness of the adopted approach in solving the presented task.

This bar chart depicts a comparison between two datasets using a series

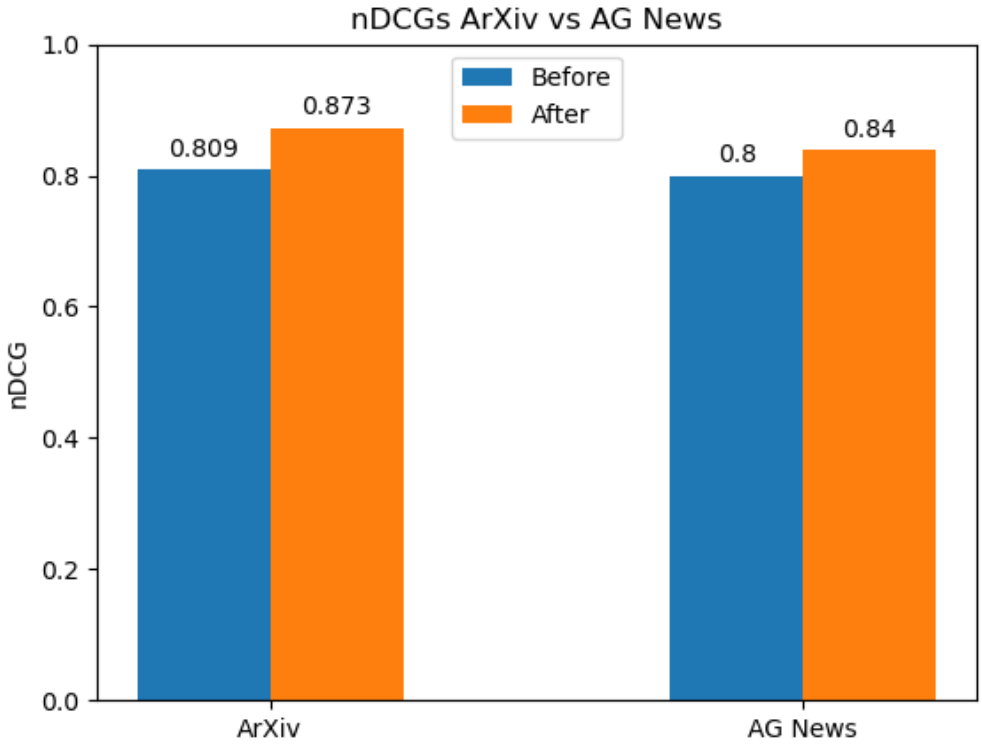

Figure 5.7. Comparison of nDCG (on the y-axis) measures between models trained and tested on ArXiv and AG News dataset (on the x-axis) both utilizing a 'random split' subdivision. Blue bars represent the nDCG values before MPNet fine-tuning, while orange one depict the nDCGs after the model's fine-tuning.

of bars. Each dataset is represented by a distinct color. The chart is organized into categories or groups along the horizontal axis, while the vertical axis shows the scale for the measured values. The height or length of each bar corresponds to the value of a specific variable or parameter within each category. This visual representation allows for a clear comparison between the two datasets, making it easy to identify any differences or trends in the data.

Furthermore, we focus our attention on a more restrictive and less ideally situation. We exploit the already mentioned "unseen" split, isolating sample from a specific category, used as test set and training the model on the remaining categories. The model performances are evaluated on the test

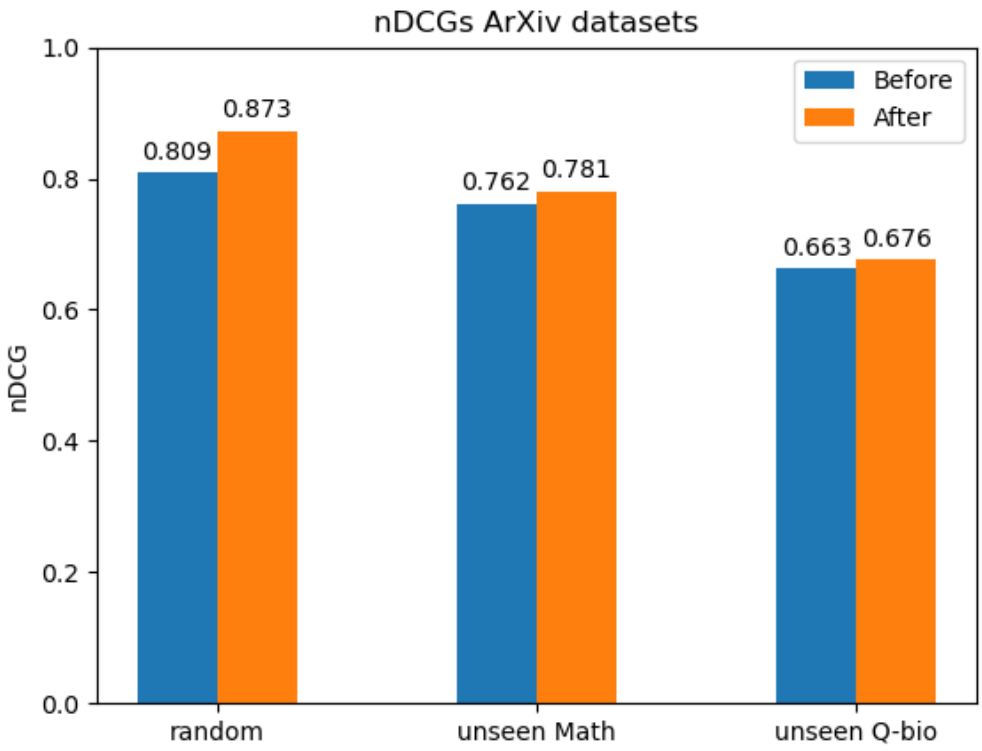

Figure 5.8. Comparison of nDCG (on the y-axis) measures between models trained on ArXiv dataset utilizing different data subdivision (on the x-axis). The models labeled as 'unseen Math' (Mathematic) and 'unseen Q-bio' (Quantitative biology) are trained by isolating the specific category, that is used for evaluation. On the other hand, the 'random' model employs the 'random split' strategy for data subdivision. Blue bars represent the nDCG values before MPNet fine-tuning, while orange one depict the nDCGs after the model's fine-tuning.

dataset, in this totally Zero-Shot situation.
In figure 5.8 are compared results obtained using a random split training dataset and results obtained on an unseen categories. Also for the 2 unseen configuration we can notice an improvement, however, as expect, is less relevant than the random configuration. In particular, the model is less able to generate an high resolution vector representation for the "Quantitative biology" (Q-bio) category samples. As consequence the classification performances are lower for that configuration. This lack may be caused by the

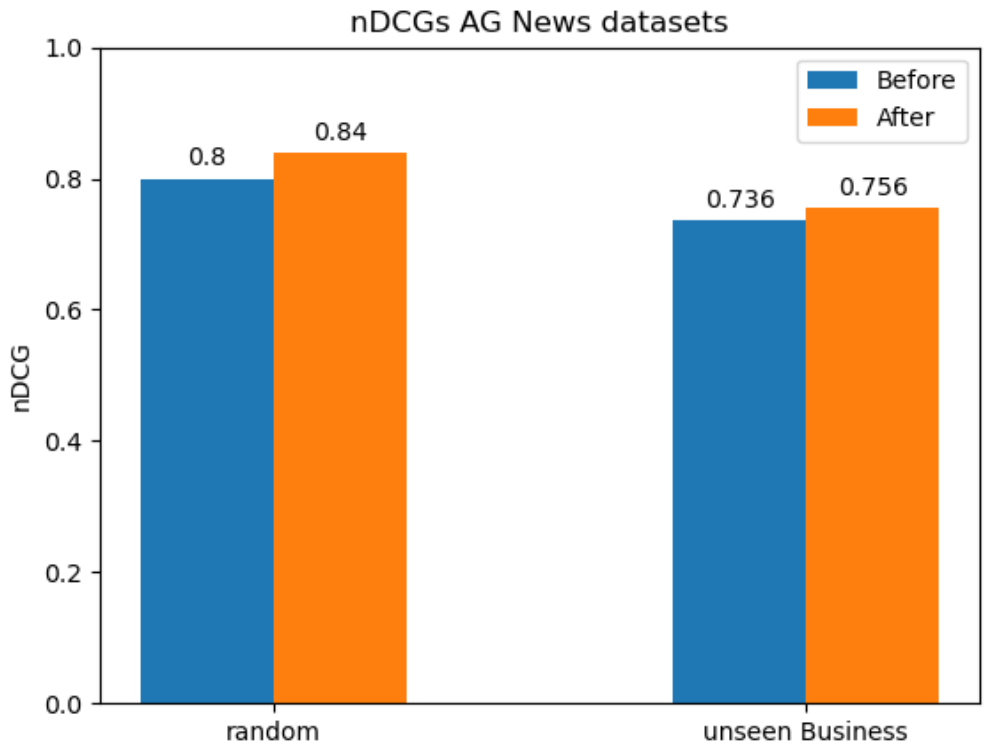

Figure 5.9. Comparison of nDCG (on the y-axis) measure between models trained on AG News dataset utilizing different data subdivision (on the x-axis). The model labeled as 'unseen Business' is trained by isolating the 'Business' category, that is used for evaluation. On the other hand, the 'random' model employs the 'random split' strategy for data subdivision. Blue bars represent the nDCG values before MPNet fine-tuning, while orange one depict the nDCGs after the model's fine-tuning.

usage for that paper of a more complex and specific language related to the argument. On the other hand, for samples under the Mathematic (Math) category, the score is higher also before the fine-tuning of the model, pointing out a better model capacity in the embeddings generation for that word or phrases. Again this situation may be caused by the usage of a language that is of course specific but maybe more general that the one used in the Q-bio samples.

For the AG News dataset and its different training settings, we can make the same consideration done for the ArXiv dataset. As we can see from the

figure, also in this case the improvement over the "unseen category" samples is lower with respect to the random configuration but is still relevant. Also in this case, for the Business category samples, we start from a decent score, sign of the model capacity in generating good vector representation.

5.3.4 0-Shot Text Classification Results on Taxonomy

We now turn our focus to the main Zero-Shot Text Classification task, where the model's objective is to categorize text according to a set of previously unseen labels. With this objective in mind, we delve into the task of categorizing academic papers within the ArXiv dataset, according to the taxonomy defined by ArXiv itself, and comparing our results with the authors' manual classifications. In practice, we leverage the model self-trained according to the framework described in Chapter 4 and 5, to Zero-Shot classify documents according to ArXiv pre-defined taxonomy. In our evaluation of the model's performance within this classification task, we employ the metrics introduced in 4.6. As mentioned earlier, these metrics provide distinct perspectives on the model's accuracy in selecting relevant categories within the top-k ranked predictions. Relevance, once again, is determined by the cosine similarity between the category embeddings and the vector representation of the abstract. Higher values in these metrics indicate the model's improved ability to predict relevant results.
To provide a comprehensive evaluation, we also present the results in terms of nDCG, offering additional insights into the relationships between these metrics.

	MPNet raw	**Fine-tuned model**
NDCG	0.4682	**0.4807**
Precision@5	0.123	**0.1315**
Recall@5	0.4035	**0.4289**
F1@5	0.1884	**0.2011**

Table 5.2. The table illustrates results of nDCG, Precision@K, Recall@K and F1@K on the unseen taxonomy classification task. An enhancements across all metrics is observed.

The table 5.2 presents the evaluation results on a top-5 rank (K = 5). An overall improvement in all metrics, particularly the recall, is clearly demonstrated. These metric improvements provide strong evidence of the framework's efficiency in enhancing classification performance on an unseen taxonomy.

Chapter 6

Conclusion

In conclusion, this study presents an exploratory method for specializing a deep Language Model into the task of Zero-Shot Text Classification. It aims to improve text classification performance in cases where the set of classification labels used has never been seen before by the model. Our method starts by defining a self-supervision framework where the model learns to associate keywords appearing in document title to their relative document abstract. Our investigation, then, probed the efficacy of supervised contrastive learning in addressing this challenging task.
Furthermore, we introduced a normalization strategy for the contrastive objective function. This modification is pivotal, as it renders the framework invariant to the variable number of positive and negative samples. It proves to be a valuable adaptation, particularly suited for scenarios where the number of samples is not fixed. This flexibility significantly enhances the applicability of our approach and stands as an important contribution to the landscape of Zero-Shot Text Classification.
Our primary objective was to leverage the power of supervised contrastive learning to enhance the vector representations of keywords. The desired outcome was to bring these keywords closer to their associated text, effectively improving the efficiency and precision of Zero-Shot Text Classification. To achieve this result we fine-tune a transformer-base architecture, MPNet, specializing it on this downstream task.
The analysis, with a specific focus on alignment and uniformity, provides evidence of the framework's effectiveness.
The alignment analysis clearly demonstrates that our approach succeeds in narrowing the semantic gap between the abstracts or descriptions and the noun phrases extracted from titles. This reduction in the gap translates to

enhanced performance in the downstream task.
However, this alignment improvement did not come without trade-offs. In fact, as underlined by the uniformity analysis, these metrics revealed a decline, indicating a less desirable distribution of samples within the vector space. This insight reveals the complexity of the Zero-Shot Text Classification task and opens the door to future research.
Finally, we evaluate our model on a standard Zero-Shot Text Classification task, where the goal is to categorize text according to an unseen taxonomy. Once again, we observed performance improvements, further demonstrating the robustness and effectiveness of this approach.

6.1 Future works

In this section, we outline potential avenues for further research and improvements in our model and methodology.

The following are some of the key areas that could be explored in future works:

- **Enhancing data distribution**: One promising aspect to enhance the quality of generated embeddings pertains to their data distribution. The integration of the uniformity loss into the contrastive objective, could help in achieving a more uniform distribution of data within the vector space, which is crucial for our downstream tasks. Investigating how this addition affects the overall performance of the model would be a valuable endeavor.

- **Addressing anisotropy issue**: This point can be directly related to the previous one. As we discussed, anisotropy in language representations has been identified as a challenge. Future research can delve deeper into better understanding and mitigating this problem to enhance model performance. Developing techniques or modifications to the training process that address anisotropy could lead to more expressive and well distributed embeddings and, consequently, better model results.

- **Incorporating Categories**: Another avenue to explore is the inclusion of categories within the training set. By incorporating category information, the model can learn to distinguish and associate different categories with the abstracts. This approach could help in better understanding

the relationships between categories and abstracts, ultimately enhancing the model's performance in tasks related to category-based analysis.

These are some of the potential directions for future work. Further research and experimentation on these aspects may lead to opportunities for improving the performance and capabilities of the model.

Bibliography

[1] Kaggle ag news dataset. https://www.kaggle.com/datasets/amananandrai/ag-news-classification-dataset.

[2] Kaggle arxiv dataset. https://www.kaggle.com/datasets/Cornell-University/arxiv.

[3] Arxiv taxonomy. https://arxiv.org/category_taxonomy.

[4] Bert explained state of the art language model for nlp. https://towardsdatascience.com/bert-explained-state-of-the-art-language-model-for-nlp-f8b21a9b6270.

[5] Cometomyhead. http://www.di.unipi.it/~gulli/AG_corpus_of_news_articles.html.

[6] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*, 2019.

[7] Kawin Ethayarajh. How contextual are contextualized word representations? Comparing the geometry of BERT, ELMo, and GPT-2 embeddings. In Kentaro Inui, Jing Jiang, Vincent Ng, and Xiaojun Wan, editors, *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 55–65, Hong Kong, China, November 2019. Association for Computational Linguistics.

[8] Tianyu Gao, Xingcheng Yao, and Danqi Chen. Simcse: Simple contrastive learning of sentence embeddings. *ArXiv*, abs/2104.08821, 2021.

[9] illustrated transformer by jalammar. https://jalammar.github.io/illustrated-transformer.

[10] Prannay Khosla, Piotr Teterwak, Chen Wang, Aaron Sarna, Yonglong Tian, Phillip Isola, Aaron Maschinot, Ce Liu, and Dilip Krishnan. Supervised contrastive learning. *ArXiv*, abs/2004.11362, 2020.

[11] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*, 2013.

[12] sentence-transformers/all-mpnet-base-v2. https://huggingface.co/sentence-transformers/all-mpnet-base-v2. Accessed: 2022-12-07.

[13] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *EMNLP*, volume 14, pages 1532–1543, 2014.

[14] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *ArXiv*, abs/2004.09297, 2020.

[15] spacy. https://spacy.io.

[16] Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Neural Information Processing Systems*, 2017.

[17] Tongzhou Wang and Phillip Isola. Understanding contrastive representation learning through alignment and uniformity on the hypersphere. In *International Conference on Machine Learning*, 2020.

[18] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime G. Carbonell, Ruslan Salakhutdinov, and Quoc V. Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Neural Information Processing Systems*, 2019.

[19] Wenpeng Yin, Jamaal Hay, and Dan Roth. Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach. *ArXiv*, abs/1909.00161, 2019.

[20] Xiang Zhang, Junbo Jake Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Neural Information Processing Systems*, 2015.