UNIVERSITÀ DI PISA

Dipartimento di Informatica

Corso di Laurea Magistrale in Data Science and Business Informatics

TESI DI LAUREA

Label Selection in Text-to-Text Neural Language Models for Classification

Relatore:
Prof.ssa Anna MONREALE

Correlatore:
Prof. Felice DELL'ORLETTA

Candidato:
Michele PAPUCCI

ANNO ACCADEMICO 2022/2023

This work contains a set of preliminary experiments with the objective of exploring and optimizing the use of reasonably small text-to-text Transformers to solve classification tasks. The broader objective is to see if we can use text-to-text Language Models, that aren't costly to train and deploy like the ones that are currently very popular (e.g. ChatGPT or LLaMa), as a unifying framework to solve any Natural Language Processing tasks. Contrary to what we need to do with larger models, with reasonably sized Transformers we need to find optimal way of casting the tasks into a text-to-text form, i.e. having a textual input, and expecting a textual output from the model. This thesis focuses on classification tasks, and in particular on the problem of how to represent class names into the best possible strings that maximize performances for the model. First, we evaluated whether this smaller models can obtain reasonable performances in classification tasks. Then, we tested the importance of label representation in this settings, finding that is, indeed, important to maximize the model performances. Finally, we presented and evaluated a novel technique to extract label representation from the training set of a classification task based on Attention attribution explainability methods.

Contents

List of Figures

List of Tables

Introduction

In recent years, the world of Natural Language Processing (NLP) underwent a revolution that shook the foundation of the field. This brought results that were never achieved before and lead to the creation of products that became commercial hits and are now popular even among non-expert people that use them daily.

This all started in 2017, with the now classic publication *Attention is all you need* by Vaswani et al., and the introduction of a new kind of Language Model: the Transformer [Vaswani et al., 2017]. By leveraging Self-Attention, a new variation of the Attention mechanism, the Transformer was able to learn continuous representation of words, that captured both semantic and syntactic contextual information. The Transformer was performing better and was faster than the previous generation of models built by encoder/decoder architecture with Attention, that instead relied mainly on LSTMs, CNNs or GRUs. This because, while the Transformers contained a lot more parameters than the previous models, it could do the majority of its calculation in parallel, something that models based on the Recurrent Neural Network architecture couldn't do.

One of the most successful iterations of this architecture was BERT, introduced in [Devlin et al., 2019], which ditched the decoder part of the model and placed a classification head, a fully connected layer, on top of the encoders stack of the Transformer. Using the contextual representation obtained by the encoder as input for the classification head, BERT reached new state-of-the-art performances in all kind of text classification tasks.

In the following years, generative models, i.e. models that take text in input and produce text in output (text-to-text), started to get traction: in 2018, the first iteration of GPT is published [Radford et al., 2018]; in 2019 GPT-2 [Radford et al., 2019], BART [Lewis et al., 2019] and T5 [Raffel et al., 2019] are released; in 2020, Open-AI publishes GPT-3 [Brown et al., 2020], etc.

This trend has yet to stop, and new model architectures for text generation are published by the day, with some of the most recent, and famous ones, being GPT-4 [OpenAI et al., 2023] and LLaMA [Touvron et al., 2023].

One of the attractive thing of the text-to-text paradigm is having a single model capable of being trained to do any task that you can cast to be textual in both input and output. In the T5 paper we read: ≪The basic idea underlying our work is to treat every text processing problem as a "text-to-text" problem, i.e. taking text as input and producing new text as output. [...] Crucially, the text-to-text framework allows us to directly apply the same model, objective, training procedure, and decoding process to every task we consider≫. By assuming that we can cast any problem in a form where we input text and accept text as output, we can use a text-to-text model as a unifying framework, where we can compare the effectiveness of different learning objectives, unlabeled datasets for pre-training and many more factors. The problem lies in the assumption that we can cast *any problem* into a text-to-text form, or that we can do so *effectively*.

Another important issue that needs to be solved is the current size of generation models. Right now, generative models have a huge number of parameters to train, some sources seem to estimate that GPT-4 has 1.76 trillion parameters [Maximilian, 2023]. This makes using this kind of universal model really costly at best, and inaccessible to the majority of the people at worst. We believe that it's important to find ways to either reduce the number of parameters of these Large Language Models (see as an example [Bansal et al., 2023]) or to find ways to make the smaller language models better (see as an example [Schick and Schütze, 2021]).

In this thesis we'll focus on how to use relatively small text-to-text models to solve classification tasks effectively. In particular, we chose to focus on how to cast them in a text-to-text form, by trying to solve the issue of how to choose the words that represent the classes of our classification task. These are the words that our model should produce when predicting, e.g. if we are doing Sentiment Polarity classification, which word should the model produce to tell us that a certain sentence have a Positive polarity? Should it generate the word *positive*, or the word *good*? Or maybe something else entirely, that doesn't have semantic connection to the task. Does it even matter which word the model is going to be fine-tuned with?

We decided to focus on three specific research questions, that this thesis will try to provide the answers for:

- Can text-to-text models classify successfully?
- How important are label representation for performances?
- Can we find a way to choose the optimal label representations?

In the first chapter, the main theoretical concept needed to understand the experiments are going to be presented, starting from traditional language models, up to the Transformer architecture and the Attention attribution explainability method that have been used for the experiments.

In the second chapter the dataset and models that have been used for the experiments are going to be presented. For the dataset, the pre-processing steps applied to it are going to be discussed, while for the models, some of the modifications to the standard architectures that were necessary to make the model work in our experimental settings are going to be presented.

Finally, in the third chapter the experiments will be presented: first we evaluated whether a relatively small generative model could be used for classification, then an analysis on the importance of label representation choice for the performance of the model will be presented, and, to conclude, a novel heuristic to choose representation for the task's classes based on Attention attribution techniques will be presented and evaluated.

Some of the results of this thesis have also been published as articles for the proceedings of two different conferences: [Papucci et al., 2022] has been published to the Proceedings of the Sixth Workshop on Natural Language for Artificial Intelligence (NL4AI 2022) co-located with 21th International Conference of the Italian Association for Artificial Intelligence (AI*IA 2022) the 21st of November, 2022; [Papucci et al., 2023] has been published at the Proceedings of the 9th Italian Conference on Computational Linguistics (CLiC-it 2023) the 28th of November, 2023.

1. State of the Art

In this chapter the key theoretical concepts needed to understand the rest of the chapters are introduced, including the experiments and the considerations made on them. In particular, this chapters is going to introduce what Language Models are, starting with some historical models that were used to solve Natural Language Processing tasks, and finishing with the latest Large Language Models based on Transformers. In particular, we'll see two specific Transformers architectures that are the ones that have been used for the experiments. Finally, an explainability technique based on Attention attribution, Value-Zeroing, will be explained, that will provide the theoretical background to understand the Label Representation Selection method presented in Chapter 3.

1.1 Traditional Language Models

A Language Model is a model that assigns probabilities to sequences of words, so, given a sequence of words $X = \{x_1, x_2, ..., x_n\}$ the Language Model (LM) outputs the probability $p(X) = p(x_1, x_2, ..., x_n)$ of observing that sequence of words, in that order.
Of course, actually modelling the human language is a hard and computationally intensive task, so some simplifications needed to be made. The standard solution was, instead of creating a model of the language from zero, trying to learn the probability distribution of words by estimating them from a large text corpora[1]. The idea is that by using big enough corpora we obtain a good estimate of the distributions of words in a language.

1.1.1 Probabilistic Language Models

The first iterations of Language Models were built using probabilistic models trained on large corpora of texts. Some notations will be introduced that will be useful for the rest of the chapter. To represent the probability of a random variable X_i taking on a value like

[1] A collection of texts. In the context of language analysis and natural language processing these are used as training data for the models.

Luca, or $P(X_i = Luca)$ the simplification $P(Luca)$ will be used; also, given a sequence of words $w_1, ..., w_n$ (or $w_{1:n}$), the joint probability of each word in the sequence having a particular value $P(X_1 = w_1, X_2 = w_2, ..., X_n = w_n)$ will be represented as $P(w_1, w_2, ..., w_n)$. Using these notions, we can represent the joint probability of observing a particular words sequence by applying the **Chain Rule of Probability** and decomposing it to a series of conditional probabilities:

$$\begin{aligned} & P(w_1)P(w_2|w_1)P(w_3|w_{1:2})...P(w|w_{1:n-1}) \\ & = P(w_1)\prod_{k=2}^{n} P(w_k|w_{1:k-1}) \end{aligned} \tag{1.1}$$

The problem with equation 1.1 is that we don't know how to compute the exact probability of a word given a long sequence of preceding words $P(w_n|w_{1:n-1})$. This is because we can't just estimate it by counting every time that a word appears after a certain sequence, because language is creative and we could encounter contexts that have never appeared before. There is also some computational constraints for calculating the probability of a word given a long context. For these reasons probabilistic models had to find **astute approximations** to learn word distributions.

1.1.2 N-grams

The idea behind N-grams is to limit the amount of context we use to compute the probability of observing a certain word. This is a **Markov** assumption, i.e. we assume that the stochastic process is **memory-less** and that we can predict the next state by using as information only the current state. In practice, most of the time we use **Markov Chains** [Markov, 1913] of order greater than one so, instead of using just the current state as information, we can look back at N-1 steps where N is the order of the N-gram. For example, in a **bigram**[2] model, instead of computing the probability of:

$$P(mela|\text{Luca mangia la}) \tag{1.2}$$

we calculate the probability of:

$$P(mela|\text{la}) \tag{1.3}$$

[2] A N-gram model with $N = 2$.

We are making the following approximation:

$$P(w_n|w_{1:n-1}) \approx P(w_n|w_{n-1}) \tag{1.4}$$

We can generalize the model to work with a context of arbitrary length N, thus obtaining a **N-gram**. So, for calculating the probability of a certain word, the model looks N-1 words in the past:

$$P(w_n|w_{n-N+1:n-1}) \tag{1.5}$$

The problem of estimating these N-grams probabilities remains, which can intuitively be solved using **Maximum Likelihood Estimation** (MLE). Given a certain corpus, we can count all the occurrences of all the possible N-grams in the texts, and then normalize the counts of each one so that we obtain a probability of observing any specific N-gram.
For example, for a bigram model, we'll compute the count of the bigrams $C(w_{n-1}w_n)$ and then normalize it by the sum of all the bigrams that share the same first word w_{n-1}. In this case, we can simplify the denominator, since the number of bigrams that starts with w_{n-1} is equal to the number of unigrams of w_{n-1}:

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})} \tag{1.6}$$

1.1.3 Hidden Markov Models

The **Hidden Markov Models** (HMM) were first introduced in [Baum and Petrie, 1966] as an augmentation of the Markov Chains earlier described in Section 1.1.1. The idea is to use **hidden information** in the text to compute words probabilities. As we know, language is a complex phenomenon, and using just the co-occurrence of words doesn't capture the deep relationship between words. For example, we know that some words are part of the same *grammatical classes* as others, and while they are not interchangeable with one another, their class gives to the speakers grammatical clues to what word may come next. For example we know that in English after an adjective there is probably going to be either another adjective or a noun, and in Italian after an article we are more likely to find a noun rather than an adverb.

The Hidden Markov Model tries to capture these phenomenons using discrete hidden random variables that represent some classes that are discovered and learned during the training process.

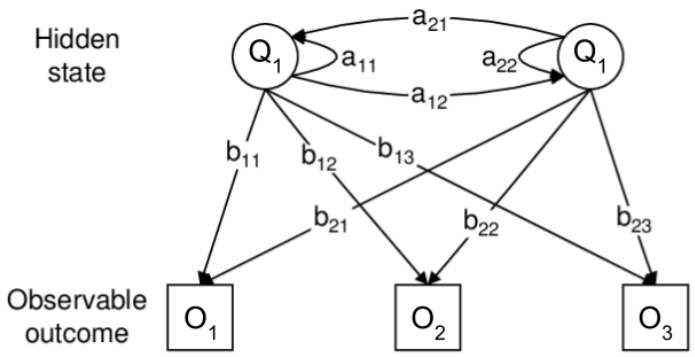

Figure 1.1: Example graph of a Hidden Markov Model.

Formalizing, an HMM is composed of:

$Q = q_1, q_2, ..., q_N$ a set of N **states**;

$A = a_{11}, ..., a_{ij}, ..., a_{NN}$ a **transition probability matrix** A where each a_{ij} represents the probability of moving from state i to state j, s.t. $\sum_{j=1}^{N} a_{ij} = 1 \ \forall i$;

$O = o_1, o_2, ..., o_T$ a sequence of T **observations**, each drawn from a vocabulary $V = v_1, v_2, ..., v_V$;

$B = b_i(o_t)$ a sequence of observations likelihoods, or **emission probabilities**, each express the probability of emitting o_t from state q_i;

$\pi = \pi_1, \pi_2, ..., \pi_N$ an **initial probability distribution** for the states. π_i is the probability that the Markov Chain will start in state i. Some states may have an initial probability of zero. Also $\sum_{i=1}^{n} \pi_i = 1$.

With a trained model we can use the **transition probabilities** and the **emission probabilities** to find the optimal state from which to emit the correct next word. The optimal state problem is solved through the use of a dynamic programming algorithm called Viterbi algorithm.

The model is trained using the **Expectation Maximization** (EM) algorithm which is an iterative two steps algorithm:

1. **E-step** (Expectation step): in this step, the algorithm estimates the expected values of the hidden variables (in the case of HMM, the states) given the current estimate of the model parameters. This is typically done using the Forward-Backward algorithm;

2. **M-step** (Maximization step): in this step, the algorithm maximizes the expected likelihood function with respect to the model parameters (transition probabilities, emission probabilities, and initial state probabilities) given the hidden variables estimated in the E-step. The aim is to adjust the parameters of the model in a way that maximizes the probability of the observed data. This step recalculates the values of A, B and π, using the probabilities computed in the E-step.

Hidden Markov Models can also be used with labeled data and can be trained to be classifiers. They have been used for a lot of different NLP applications, such as Part-of-Speech Tagging [Kupiec, 1992] and Speech recognition [Baker, 1975].

1.1.4 Conditional Random Fields

While HMM are useful and powerful models, to achieve a high accuracy they need a number of augmentations. For example: when PoS-Tagging, the model could encounter an unknown word that it never saw during training (e.g. proper names or new acronyms). To help with handling those cases it would be useful to inject some prior knowledge into the model, e.g. give a higher probability to a word with a capitalized first letter to be recognized as a proper noun, or, if a word is composed of only letters and dots, understanding that is probably an acronym.
One way to do that is to use **Conditional Random Fields** (CRF) [Lafferty et al., 2001a], that are a discriminative counterpart of the HMM. Instead of learning the joint distribution of the observations, they learn to model the conditional probability of some annotated classes given the observations.

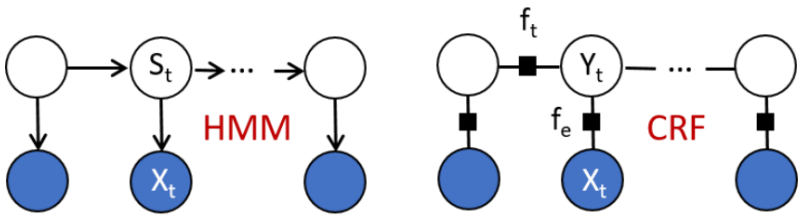

Figure 1.2: Difference between HMM and CRF. f_t replaces the transion probabilities between states, while f_e replaces the emission probabilities of the states.

The key difference to HMM is that CRFs are modeled as an undirected graph, where we have both hidden random variables and observed random variables (the sequence), then, instead of the transition probabilities and emission probabilities, we have **feature functions**. Feature functions are **hand-engineered** functions that give a score to some *preferred configurations*. By taking the previous example into consideration, we could have a function that give a high score for the proper noun class when encountering a word with a capitalized first letter in a PoS-Tagging scenario. Using these, we can model specific linguistic phenomena we think could be useful to help the model discriminate between classes. These functions are the ones that inject prior knowledge into the model. The model is trained through **Maximum Likelihood Estimation** of its parameters, which is solved by gradient descent.

CRFs have successfully been used for a variety of NLP tasks [Lafferty et al., 2001b] such as Named Entity Recognition [Settles, 2004] and Shallow Parsing [Sha and Pereira, 2003].

1.2 Neural Networks and Neural Language Models

The Neural Networks are so called because their origins lie in the **McCulloch-Pitts Neuron** [McCulloch and Pitts, 1943] which was a simplified model of a human neuron. It was used as a computing element, and was described using propositional logic. The name stuck, even if modern Neural Networks no longer share this biological inspiration.

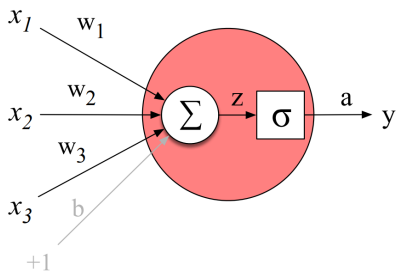

Figure 1.3: Schema of a single unit with three inputs (plus bias).

The basic unit of a network is a single computational unit, that, in the most basic terms, does a weighted sum of its inputs (which are real valued numbers) plus a **bias term**[3]. Given a set of inputs $x_1, x_2, ..., x_n$, a set of learned weights $w_1, w_2, ..., w_n$ and a bias b, the weighted sum z is represented as:

$$z = b + \sum_i w_i x_i \tag{1.7}$$

This can be represented in **vector notation** using a **dot product** ($\cdot$):

$$z = \mathbf{w} \cdot \mathbf{x} + b \tag{1.8}$$

The last important step in the basic unit of a network is that we don't use z directly, which is a linear function of x, but we apply a non-linear function f to z, obtaining a, the **activation** of the unit, also commonly indicated as y:

$$y = a = f(z) \tag{1.9}$$

1.2.1 Feed Forward Neural Networks

One of the most important feature of Neural Units is their capabilities to be combined into networks, which improve their performances and capabilities greatly. A clever demonstration of the need of having multi-layer networks was the proof of [Minsky and Papert, 1969] that a Perceptron (a unit, like described before, but without the non-linear activation function) can't compute the logic XOR operation of its inputs.

[3]The units, before the activation functions, learn a linear function. The bias is the term of that function that allows shifting, i.e. $y = ax + b$ the weight matrix models the a, while the bias models the b.

After [Goodfellow et al., 2016], the standard solution to the XOR problem is using two layers of units with ReLU[4] as the activation function.

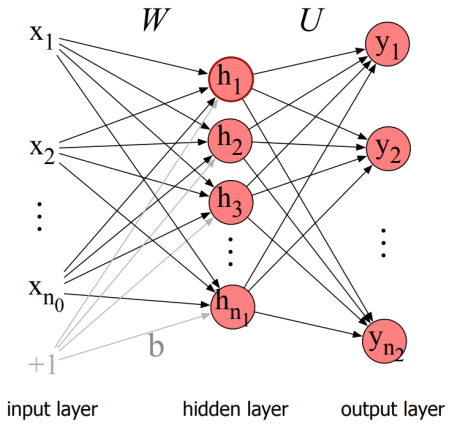

Figure 1.4: Schema of a Feed Forward Neural Network with two layer: an input layer and a hidden layer.

A Feed-Forward Neural Network (FFNN), or a Multi-Layer Perceptron (MLP), is a multi-layer network of **fully connected** units with no cycles, where each unit has in input all the outputs of the previous layer. The most important part of the Neural Network is the Hidden Layer h, where each unit h_i works as described in Section 1.2. Being fully connected, each hidden unit sums over all the inputs unit.
The parameters of a single layer are represented as a weight matrix $\mathbf{W}$ and a single bias vector $\mathbf{b}$. Each element $\mathbf{W}_{ij}$ represent the weight of the connection from unit i to unit j. Using matrix notation, and assuming we are using a sigmoid as the activation function, we can compute the activation h of the entire layer as:

$$h = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1.10}$$

From Figure 1.4 we see that the output layer has a weight matrix $\mathbf{U}$, and using the various h_i as input we can calculate its activation z. However, z is not a useful output since is a vector of real numbers. Normalization techniques such as **Softmax** are used to scale these values in a range from 0 to 1 with a sum to 1 constraint. This gives us what can be interpreted

[4]ReLU is an activation function defined as the positive part of its argument $f(x) = x^+$. It was first introduced in the context of Neural Networks by [Hahnloser et al., 2000].

as a vector of probability. For example, in a classification scenario, if we have an output layer with a number of units equal to the number of classes we need to discriminate, the softmaxed output vector can be interpreted as the probability of the inputs x_i of being part of each of those classes. In equation, the two-layer network we described, that takes an input vector $\mathbf{x}$, outputs a probability distribution $\mathbf{y}$, is parameterized by weight matrices $\mathbf{W}$ and $\mathbf{U}$, and a bias vector $\mathbf{b}$ (the output layer usually does not have a bias vector), is defined as:

$$\mathbf{h} = \sigma(\mathbf{W}\mathbf{x} + \mathbf{b}) \tag{1.11}$$

$$\mathbf{z} = \mathbf{U}\mathbf{h} \tag{1.12}$$

$$\mathbf{y} = softmax(\mathbf{z}) \tag{1.13}$$

FFNNs for Language Modeling were first proposed by [Bengio et al., 2000]; this was because learning words distribution with a neural network has many advantages compared to other techniques (such as N-grams): they can handle longer contexts, can generalize better over similar words and are more accurate at word-prediction. This, however, comes at the price of complexity, speed and interpretability. In fact, **Neural Language Models** are slower, have a more complex architecture and are considered black-boxes, i.e. their inner workings are hard to interpret and usually meaningless for a human.

A Feed Forward Neural Language Model (ML) is defined as a FFNN that takes as input, at time t, a representation of some number of previous words, and outputs a probability distribution over possible next words. Like for N-Grams (Section 1.4) we are approximating the context to N-1 words.

In Neural Language Models, words are represented through **embeddings**, which capture more contextual information than mere word identity, as seen in traditional N-gram models. Embeddings are continuous representations of words, and usually they are vector of size equal to the hidden size of the network, and their values are simply the activation of the hidden neurons when the word is seen as input. This approach enhances the model's ability to generalize to new, unseen data. For instance, if a model trained on the phrase *"the cat gets fed"* encounters a similar but unseen phrase like *"the dog gets"* the model should be able to intelligently predict the following word *"fed"* by recognizing the con-

textual similarity between *"cat"* and *"dog"* through their embeddings. This capability is not present in N-gram models, which rely strictly on previously observed word sequences.

Feed Forward Neural Networks are trained through the use of a **loss function** which is usually the **cross-entropy loss**. For **hard classification tasks**[5] such as Language Modeling, we deal with two vectors: $\mathbf{y}$ is a vector of size K, where K is the number of classes, each element is 0 except for the correct class for that example which has value equal to 1 (it's called a **one-hot vector**), while $\hat{\mathbf{y}}$ is the output of the Neural Network last layer, being of size K, and each elements contains the predicted probability for the example of being part of that class. The loss function for a single example $\mathbf{x}$ is the negative sum of the logs of the K output classes, each weighted by their probability $\mathbf{y}_k$. Being that $\mathbf{y}$ is always 0 except for the correct class, we can write the loss function as:

$$L_{CE}(\mathbf{y}, \hat{\mathbf{y}}) = -log(\hat{\mathbf{y}}_c) \tag{1.14}$$

where c is the index of the correct class. This is called **negative log likelihood loss**, as it's the negative log of the output probability corresponding to the correct class.
To actually update the weights of the network to make it learn, the **Error Back-Propagation** technique is used. It was introduced in [Rumelhart et al., 1985a], and the idea is to calculate the **gradient** of the loss function by calculating the **partial derivative** of the function with respect of each parameter. To calculate the error, and then update every parameter in the network, we need to do a **backward pass**: starting from the last layer, and by applying the chain rule of gradients, we can go back through the net to calculate how each weight contributed to the error. Then, we update each parameter using the calculated errors accordingly.

1.2.2 Word2Vec

In previous section we assumed that words were represented as either a one-hot encoding vector of size of our vocabulary V, or as the vector of the hidden layer activation of the network. Another traditional technique is using a vector with size N, where each $n \in N$

[5] Hard classification tasks are such that only one class is the correct one across the various options.

is a linguistically defined, hand-crafted feature that should help discriminate that word, or sentence, from the others.

A better approach to *encode* information is Word2Vec, which transforms a word into a **dense** vectorial representation. The idea is to represent words using real numbers in a vector space with a pre-defined number of dimensions.

The main algorithm to do so is called **skip-gram with negative sampling** (SGNS), which is one of the two algorithms available in a software package called **word2vec** [Mikolov et al., 2013a], and is commonly referred to with that name. The intuition is that, instead of counting the co-occurrence of words in sentences, a binary classifier can be trained to predict how likely are two words in a sentence to be near each other. Then, once the model is trained, we can use its learned weights as embeddings. Word embeddings can be used as inputs to a variety of different tasks, such as token-level classification or, by aggregating them, for sentence-level classification.

The key part here is that, through a process called **Self-Supervision**, raw data can be used to train the model: for each word, we can choose a neighboring word as a positive example, and a random one from the whole dataset as a negative example. With this, annotated data is not needed, and the vast amount of free, raw text can be leveraged.

The key steps are:

1. Constructing the dataset. For each word *w* in the text:
    (a) Treat *w* and a random neighboring word as *positive* examples for the classifier;
    (b) Sample random words from the text as *negative* examples for the classifier;
2. Train a Logistic regressor to distinguish beetwen the two cases;
3. Extract the learned weights of the classifier as embeddings.

The learned embeddings are *static*, since they aren't constructed using any contextual information. This means that for *polisemic* words there is just one vectorial representation, regardless of the true meaning of the word in a specific context. However, even with its shortcomings, word2vec embeddings have been showed to have interesting semantic properties. For example, some works [Mikolov et al., 2013b, Levy and Goldberg, 2014]

demonstrated that these embeddings had capabilities to solve semantic analogies problems. One of the most famous one is that the vector obtained by doing $\overrightarrow{King} + \overrightarrow{Woman}$ is similar to the one of $\overrightarrow{Queen}$. Similarly, $\overrightarrow{Paris} - \overrightarrow{France} + \overrightarrow{Italy}$ results in a vector that is similar to the one of $\overrightarrow{Rome}$. The embedding model thus seems to be capable of extracting complex semantic relations between words, such as MALE-FEMALE or THE-CAPITAL-OF.

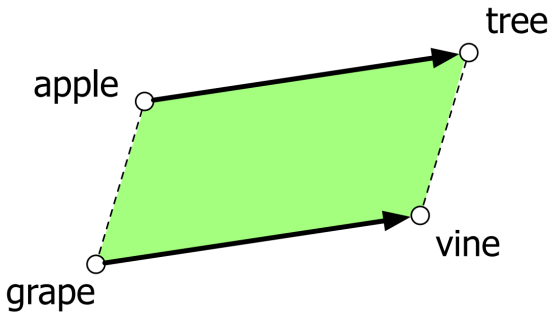

Figure 1.5: The parallelogram model for analogy problems. The location of $\overrightarrow{vine}$ can be found by subtracting $\overrightarrow{apple}$ from $\overrightarrow{tree}$ and then adding $\overrightarrow{grape}$.

These kind of operations resembles the **parallelogram model** [Rumelhart and Abrahamson, 1973] shown in Figure 1.5. However, this method doesn't work well for other kinds of relations as demonstrated in various works [Linzen, 2016, Gladkova et al., 2016, Schluter, 2018].

1.2.3 Recurrent Neural Networks for Sequences: LSTM and GRU

Language is inherently a *temporal* phenomenon; humans produce both spoken and written languages as a continuous stream that unfolds over time. This core fact about language has led many algorithms to try and include the temporal factor of language inside their inner workings, one example being the previously cited Viterbi algorithm for HMMs. Vanilla Neural Networks however can only look at a fixed-size window of words, and don't have a temporal nature, accessing all inputs simultaneously. **Recurrent Neural Networks** (RNNs), and their variants such as **Long-Short Term Memory** and **Gated Recurrent Units**, are a kind of neural networks that provide a mechanism that allows them to handle the temporal nature of language in their **recurrent connections**, making the

models capable of having no fixed-size window to the past, and keeping as context every previous words in the sentence. The idea is to have a single set of parameters for each time-step: a single word representation is passed through the network at a time, and by doing so, the network keeps building an internal representation of the sentence up until that point.

One of the main issues with Neural Networks is the so called **catastrophic forgetting** [Goodfellow et al., 2013]. This happens when, during the back-propagation, the chain rule is applied to the calculated partial derivative. Since some of the most popular activation function, such as the sigmoid, have a lower than zero cap on their derivative (the sigmoid only goes up to 0.25), this lower than one values are constantly multiplied together when chaining the gradients, resulting in ever shrinking numbers, and as a consequence, ever shrinking updates to the weights of the network. This can reach a point where learning becomes very hard or impossible. This mechanism is called **gradient vanishing**, and was first described in [Hochreiter, 1998]. This problem appears very plainly when trying to train Vanilla Recurrent Neural Networks, and the search for a solution led to the creation of **gated architectures**.

The idea of the first proposed gated architecture, the **Long-Short Term Memory** (LSTM) [Hochreiter and Schmidhuber, 1997] is to have a central memory, called **Constant Error Carousel** (CEC), to store information into. Then, the access to the CEC is regulated by two essential parameterized gates: one to write information on it; one to read out information from it.

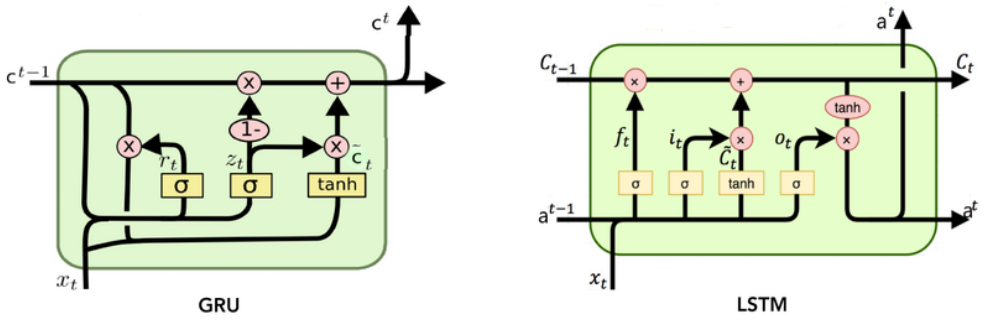

Figure 1.6: Differences between a LSTM cell and a GRU cell.

LSTM works by implementing three gates: the **Input Gate**, which controls what part of the inputs are written in the CEC, the **Forget Gate**, which controls which part of the previous activation needs to be forgotten, and the **Output Gate**, which decides which part of the CEC must be outputted. All three gates are parameterized with their own sets of weights and biases:

$$\mathbf{I}_t = \sigma(\mathbf{W}_{Ia}\mathbf{a}_{t-1} + \mathbf{W}_{Iin}\mathbf{x}_t + \mathbf{b}_I) \tag{1.15}$$

$$\mathbf{F}_t = \sigma(\mathbf{W}_{Fa}\mathbf{a}_{t-1} + \mathbf{W}_{Fin}\mathbf{x}_t + \mathbf{b}_F) \tag{1.16}$$

$$\mathbf{O}_t = \sigma(\mathbf{W}_{Oa}\mathbf{a}_{t-1} + \mathbf{W}_{Oin}\mathbf{x}_t + \mathbf{b}_O) \tag{1.17}$$

Using the input and the activation at the previous time-step, the **input potential** is calculated:

$$\mathbf{g}_t = tanh(\mathbf{W}_a\mathbf{a}_{t-1} + \mathbf{W}_{Iin}\mathbf{x}_t + \mathbf{b}_a) \tag{1.18}$$

The input potential represent *what could be written* in memory. To actually being written in the CEC, first the previous CEC state is weighted using the Forget Gate activation, which decides *what to forget*. Then, the input potential is weighted with the Input Gate activation, which decides *what to store*, and the two are added together, obtaining the new internal state:

$$\mathbf{C}_t = \mathbf{I}_t \cdot \mathbf{g}_t + \mathbf{F}_t \cdot \mathbf{C}_{t-1} \tag{1.19}$$

Finally, the cell activation is calculated by passing the CEC state through a hyperbolic tangent function, and weighting the output with the Output Gate activation, which decides *what to show* to the outside from the internal memory:

$$\mathbf{a}_t = \mathbf{O}_t \cdot tanh(\mathbf{C}_t) \tag{1.20}$$

Gated Recurrent Unit (GRU) [Chung et al., 2014] was proposed later as a cheaper alternative to the LSTM. In fact, GRU cells have fewer parameters, while their performances on most tasks are comparable to the ones of LSTM cells. GRUs have only two gates: **Reset** and **Update Gates**:

$$\mathbf{z}_t = \sigma(\mathbf{W}_{zc}\mathbf{c}_{t-1} + \mathbf{W}_{zin}\mathbf{x}_t + \mathbf{b}_z) \tag{1.21}$$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{rc}c_{t-1} + \mathbf{W}_{rin}\mathbf{x}_t + \mathbf{b}_r) \tag{1.22}$$

Differently from the LSTM, there is no CEC or any other form of internal state, but by using the Reset and the Update Gate combined, the model calculates its output and only the cell activation is propagated through the future. Basically, what in the LSTM was the input potential is calculated as:

$$\mathbf{c}_t = tanh(\mathbf{W}_{hh}\mathbf{c}_{t-1} + \mathbf{W}_{hin}\mathbf{x}_t + \mathbf{b}_h) \tag{1.23}$$

And then the activation is calculated as the Update Gate-weighted input potential, plus the previous activation weighted by the inverse of the Update Gate:

$$\mathbf{c}_t = (\mathbf{1} - \mathbf{z}_t) \cdot \mathbf{c}_{t-1} + \mathbf{z}_t \cdot \mathbf{c}_t \tag{1.24}$$

These kind of networks are trained similarly to Feed-forward Networks, however the back-propagation algorithm needs to be adjusted for their recurrent nature. In particular, the new back-propagation is a two-pass algorithm: in the first pass the network computes the hidden state and the output, accumulating the loss while doing so; in the second pass, the sequence is processed in reverse, computing the required gradients and saving the error term. This is then used to updated the hidden layer for each step backward in time. This algorithm is called **Back-Propagation Through Time** [Werbos and John, 1974, Rumelhart et al., 1985b, Werbos, 1990].

The way these RNNs cells are used is by stacking layers of them one on top of the other, with the output of a certain layer feeding the input of the layer on top of it. Then, based on the type of task the network is used to solve, a final layer can be used to do classification or regression.

LSTMs have been successfully used for a variety of NLP tasks such as Handwriting Recognition [Graves et al., 2007] and Speech Recognition [Graves et al., 2013]. One of the main variants that became successful was using a **bidirectional RNN**, where two RNNs were placed *in parallel* in the network, analyzing the same sequence, one from left to right and the other from right to left. Then the two cell activation were used together by combining them [Schuster and Paliwal, 1997]. Using pre-trained word embeddings from word2vec (see Section 1.2.2) or GloVe [Pennington et al., 2014] as input for RNNs also became quite popular, dominating performances on a variety of tasks, such as Part-of-

Speech Tagging [Ling et al., 2015], Named-Entity Recognition [Chiu and Nichols, 2016] and Semantic Role Labeling [Zhou and Xu, 2015].

1.2.4 Encoder-Decoder RNNs and Attention

One of the evolution of the RNNs architecture was the creation of the **Encoder-Decoder model** pioneered by [Kalchbrenner and Blunsom, 2013]. The model consists of three parts:

1. An **Encoder**, that accepts a sequence and generates a representation. For the Encoder, any architecture capable of handling sequences can be employed;
2. A **context vector**, that conveys the essence of the input to the Decoder;
3. A **Decoder**, which accepts the context vector and generates an arbitrary length sequence of hidden states from which a corresponding sequence of outputs can be obtained. As the Encoder, any architecture capable of handling sequences can be employed.

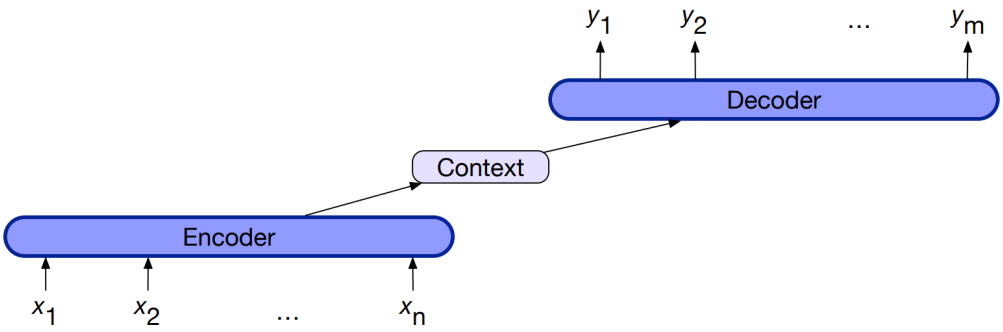

Figure 1.7: An example of an Encoder-Decoder model. The context vector is a function of the hidden activation of the Encoder and it may be used by the Decoder in a variety of ways.

The main problem of the model is that the context vector must be able to represent absolutely everything needed from the source sequence, since is the only thing that the Decoder sees about the starting text.

The solution to this bottleneck problem is the **Attention mechanism**. The Attention Module is placed between the Encoder and the Decoder; it receives each hidden state produced by the Encoder when encoding the sequence, and also receives a **context information** vector from the Decoder. With these information the module can *attend more* certain parts of the inputs that are revealed to be more important to the current context.

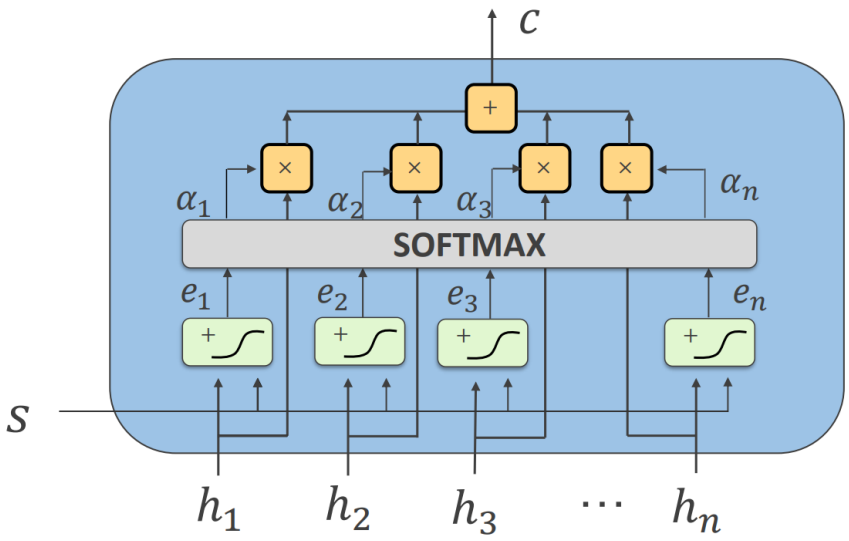

Figure 1.8: The inner workings of an Attention Module, placed between Encoder and Decoder. Where $h_1, h_2, ..., h_n$ are the hidden state of the Encoder at different time steps, S is the context information given by the Decoder.

The first step in the Attention Module is to determine the relevance of each input hidden state with respect to the contextual information S. To do so, the vector S is combined with each hidden state h_t. This can be done with something as simple as a dot product or as complex as using a Multi-Layer Perceptron. The important part is that for each hidden state $h_1, h_2, ..., h_n$ we get a scalar indicating how much the two vector align. Then, these scalars $e_1, e_2, ..., e_n$ are normalized using a Softmax. The output of a Softmax are n scalars $\alpha_1, \alpha_2, ..., \alpha_n$, ranging from 0 to 1, that indicate how important are each of the hidden state vector for the contextual information fed to the Attention Module. These are then used to compute a context vector to be fed to the Decoder module by doing an α-weighted aggregation:

$$\mathbf{C} = \sum_{i=1}^{n} \alpha_i \mathbf{h}_i \tag{1.25}$$

During the decoding part, the previous hidden state of the Decoder is fed to the Attention Module as the **contextual vector** S for each decoding step.

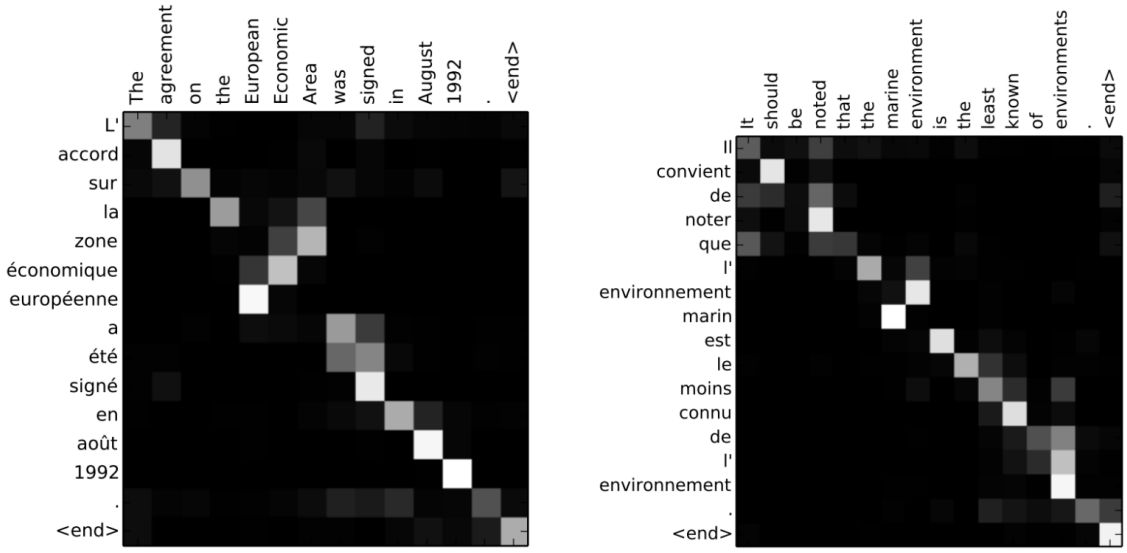

Figure 1.9: An example of Attention matrices for a translation task from French to English.

The α values can also be visualized to see which input information are used during each decoding step: these matrices are called **Attention matrices**. An example of such is visible in Figure 1.9, taken from [Bahdanau et al., 2016].

1.3 Transformers

Transformers architectures are now the standard tool to do Language Modeling and to solve most NLPs tasks. Like the RNNs of the previous section, Transformers can handle distant relations, but unlike those architecture they don't rely on recurrent connections. Transformers are based on the Encoder-Decoder architecture, where the Encoder maps an input sequence $X = (x_1, ..., x_n)$ to a sequence of continuous representation $Z = (z_1, ..., z_n)$, and the Decoder generates from Z an output sequence $(y_1, ..., y_m)$ one step at the time. Transformers are made of **blocks**, each of which is composed by a series of layers. The core innovation in Transformers is the **Self-Attention layer**, which is a specialization of the classic Attention Module, that allows the network to directly extract and use infor-

mation from arbitrarily large contexts without the need to pass it through intermediate recurrent connections, as it was the case in RNNs.

1.3.1 Self-Attention Layer

The inputs for the Self-Attention layer are three matrices, each containing a linearly transformed embedding of each element of the input sequence. The three matrices Q (Query), K (Key), V (Value), are obtained by applying three learned sets of weights to the input sequence:

$$\begin{aligned} \mathbf{Q} &= \mathbf{W}^Q \cdot \mathbf{X} \\ \mathbf{K} &= \mathbf{W}^K \cdot \mathbf{X} \\ \mathbf{V} &= \mathbf{W}^V \cdot \mathbf{X} \end{aligned} \tag{1.26}$$

The matrices are of size $[\textit{sequence length},\ d_k]$ where d_k is the size of the key vector[6]. Then, each Query vector is compared to each Key vector to calculate a similarity score. This is usually implemented by the means of a simple dot product:

$$score(Q,K) = \mathbf{Q} \cdot \mathbf{K}^T \tag{1.27}$$

Then, these scores are normalized through the use of a Softmax function:

$$attention\_weights = softmax(\frac{score(Q,K)}{\sqrt{d_k}}) \tag{1.28}$$

Then, we can use the obtained *attention weights* to calculate the output by doing a weighted sum with the Value vector:

$$attention\_weights \cdot \mathbf{V} \tag{1.29}$$

The Self-Attention calculation is usually written by compacting all the previous steps:

$$SelfAttention(Q,K,V) = softmax(\frac{\mathbf{Q} \cdot \mathbf{K}^T}{\sqrt{d_k}}) \cdot \mathbf{V} \tag{1.30}$$

[6]Technically, each vector has its own size d_q, d_k and d_v, however, these are usually set to be the same. In particular, d_k and d_q have to have the same dimensions to obtain a straightforward computation of the next step, which is the score calculation by the means of a dot product. While d_v could technically be different, it usually isn't when the models are implemented.

We can see how the formulation for the Self-Attention is basically the same as the one we previously described for the Attention Module in Encoder-Decoder RNNs (See Section 1.2.4). In fact, Self-Attention is simply a specialization of the Attention Module, where the three inputs for the Attention calculation (the context vector C, the Encoder hidden states h, and the Decoder hidden states s) are swapped for linear transformation of the input. This means that we can calculate the output of the module, without relying on hidden states provided by RNNs.

In practice, these layers in a Transformer model compute a variation of the Self-Attention called **Multi-head Attention**. The difference is that this version computes multiple Self-Attention outputs in parallel and then concatenates the outputs. This means that for each *head*, d_k, d_q and d_v are going to be set as $\frac{d_{model}}{h}$, where d_{model} is the hidden size used for all the Transformer layers and residual stream, while h is the number of heads in each Self-Attention layer.

$$\begin{aligned} MultiHeadAttention &= Concat(head_i, ..., head_h)\mathbf{W}^O \\ head_i &= SelfAttention(\mathbf{Q}\mathbf{W}_i^Q, \mathbf{K}\mathbf{W}_i^K, \mathbf{V}\mathbf{W}_i^V) \end{aligned} \tag{1.31}$$

where the weight matrices are: $\mathbf{W}_i^Q$ with dimension [d_{model}, d_q]; $\mathbf{W}_i^K$ with dimension [d_{model}, d_k]; $\mathbf{W}_i^V$ with dimension [d_{model}, d_v]; $\mathbf{W}_O$ with dimensions [hd_k, d_{model}]. The purpose of W_O is to map the output of the Multi-head Attention back to size d_{model}. Using multiple heads allows the model to project each input into h different sets of transformation, and to jointly attend to information from different representation sub-spaces at different positions, which can't be done with single Attention head.

1.3.2 The Vanilla Transformer

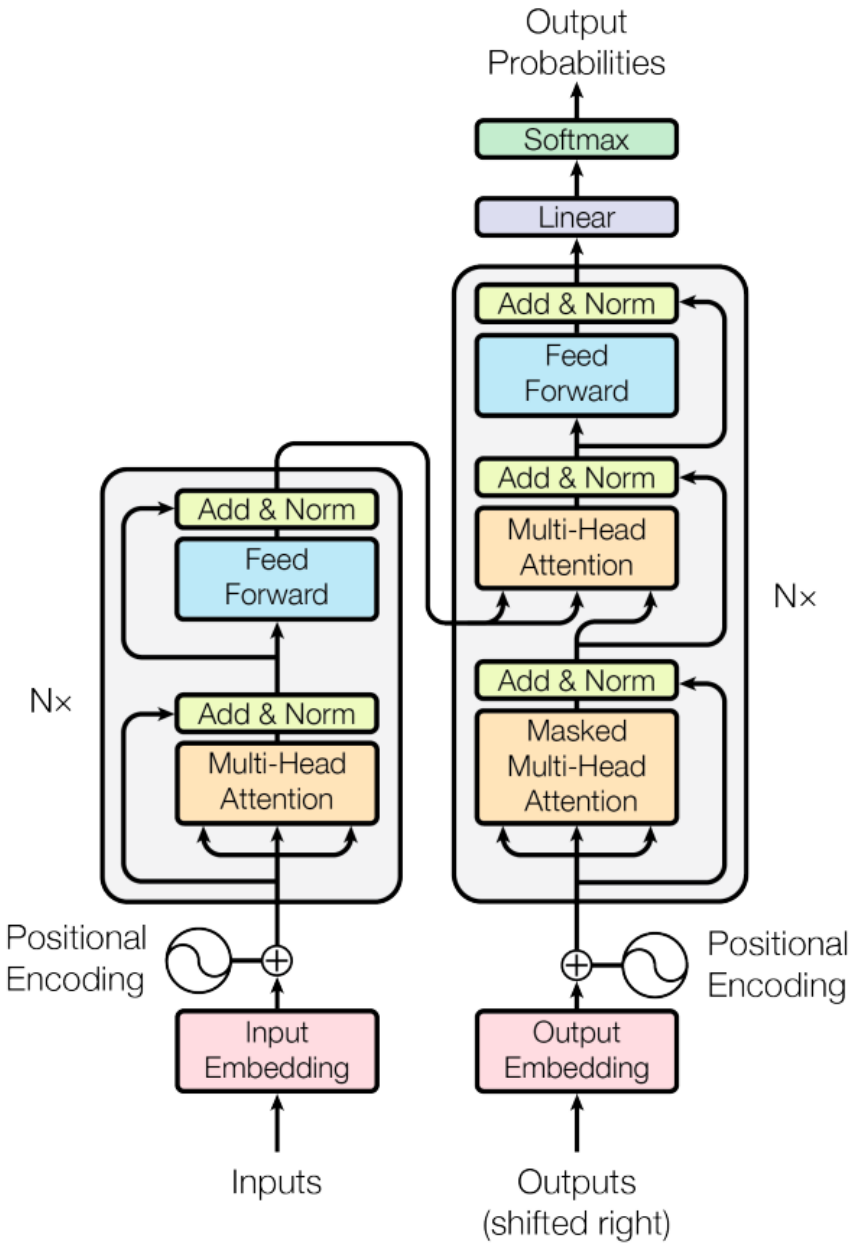

Figure 1.10: The Vanilla Transformer Architecture. The Encoder being on the left and the Decoder on the right. Image taken from [Vaswani et al., 2017].

The standard Transformer (or Vanilla) is composed by a stack of Encoders and a stack of Decoders. In the **Encoder part** of the model, the input sequence passes through an Embedding Layer that projects each token in the sequence into a vector of size d_{model}.

One of the things that the Transformers lost from the previous generation of RNNs is the concept of *time*. RNNs looked at each element in a sequence in an ordered matter and built a hidden state sequentially. This, however, is not how Transformers work: as we saw in Equation 1.30, all the tokens are processed in parallel. To give back a sense of time to the model, **Positional Encodings** are added to the input matrix X. There are a number of different ways to represent time, some techniques inject simple alternating pattern to the tokens' embedding, others, instead, learn linear maps to weight the token embeddings [Gehring et al., 2017].

Then, the inputs enter the Encoder stack. Both the Encoder and the Decoders blocks are characterized by having a skip connection before each layer. Each output of each layer is summed and then normalized[7] with the input of the layer. This stream of information, to which each layer contributes to, is called **Residual Stream**. As we can see from Figure 1.10, after the Positional Encoding the stream is split, and after the first layer (Multi-head Attention) it's added back together.

In the **Encoder block**, the inputs first go through the Self-Attention layer, and then trough Multi-Layer Perceptron. The MLP is usually made of an input layer of size d_{model}, a hidden layer of size $4d_{model}$[8] and an output size of d_{model}. The MLP also has a non-linear activation function (usually a GeLU or a ReLU) and is the main non-linear calculation point of the Transformer.

In the **Decoder Block** the first steps are the same, but instead of feeding it the input sequence, the Decoder is fed the generated sequence up until that point. The two blocks work the same, but before doing the Multi-head Attention, the target sequence passes through a Masked Multi-head Attention Layer, which functions the same as the normal one, but is only allowed to attend to *earlier positions* in the target sequence. This is done by masking future positions (setting them to -inf) before the Softmax step in the Self-Attention calculation. At the end of the Decoder we have a linear layer that maps the final

[7]The normalization is used to make the Residual Stream vector have mean equal to 0 and variance equal to 1.

[8]The hidden size of the MLP being $4d_{model}$ is really just a convention, the important idea is that the MLP projects the d_{model} Residual Stream onto a bigger space, where the model goes through non-linearity in the form of the activation function, and is then projected back to a dimension equal to d_{model}.

residual stream vector of sized d_{model}, to size d_{vocab}. By using a final Softmax layer we obtain the probabilities over the vocabulary.

1.3.3 Training a Transformer

Being a Neural Network, Transformers are trained trough gradient-descend learning by iterating over examples. However, to learn rich representation, usually their training is split in two phases.

In the **Pre-training phase** the model learns trough self-supervision (similarly to what w2v does, see Section 1.2.2) leveraging huge amount of raw textual data. To do this, the model goes through the effort of completing pre-training objectives (some of these objectives that are used to train BERT and T5 will be presented in the models' respective sections). This is the most costly phase of the training, and is where the model learns contextual representation for tokens that contains both semantic and morpho-syntactic information.

In the **Fine-tuning phase** the model is trained using a relatively small amount of labeled data used to teach it the target task.

1.3.4 BERT

Bidirectional Encoder Representations from Transformers (BERT) [Devlin et al., 2019] was the first breakthrough implementation of the Transformer architecture, reaching new state-of-the-art performances and growing rapidly in popularity. BERT uses only the Encoder part of the Transformer and learns bidirectional representation using both the left and right context of a word in the input sequence. BERT is pre-trained using two pre-training tasks:

1. **Masked Language Modeling** (MLM): is a pre-training objective where some percentage of the input is masked, and the model has to predict which token has been masked (See Figure 1.11 for an example);

2. **Next Sentence Prediction** (NSP) is a pre-training objective where the model is presented a single input that contains two sentences separated by a special token.

The model has to output whether the two inputs sentences are consequent or not.

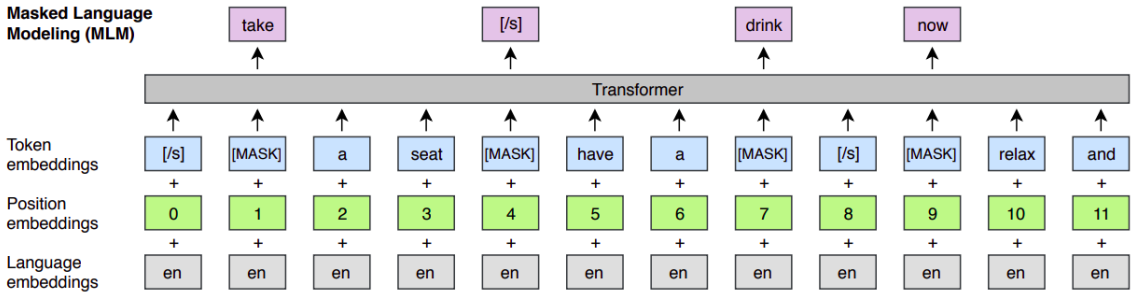

Figure 1.11: The Masked Language Modeling prediction task. Taken from [Lample and Conneau, 2019].

The original BERT was pre-trained on two datasets, the *English Wikipedia* and the *BookCorpus* [Zhu et al., 2015]. During the fine-tuning phase, BERT learns to solve classification tasks leveraging the information acquired during the pre-training phase of its training. BERT reached new state-of-the-art performances of famous and highly competitive benchmarks such as GLUE [Wang et al., 2019] and SQuAD [Rajpurkar et al., 2016].

From BERT then spawned a family of BERT-like model that improved upon the original concept, like RoBERTa [Liu et al., 2019], that improved the original BERT by training the architecture with more data, for more time, removing the NSP pre-training objective and changing the masking system to a dynamic one, or ALBERT [Lan et al., 2020], that is a lighter version of BERT, that uses parameter reduction technique to make the model computationally cheaper to train and infer with. Most of these models than received either multilingual or localized trained version for a variety of languages.

1.3.5 T5

Text-To-Text Transfer Transformer (T5) [Raffel et al., 2019] is a vanilla Transformer pre-trained using the Colossal Cleaned Crawled Corpus (C4), a collection of text took from the Common Crawl Project[9], which was presented in the same article. While the original T5 was trained only on English text, a series of multilingual or localized version of T5

[9]Common Crawl Project website: https://commoncrawl.org/

have then been published such as ByT5 [Xue et al., 2022], IT5 [Sarti and Nissim, 2022], AraT5 [Nagoudi et al., 2022], etc.

Text-to-Text Framework

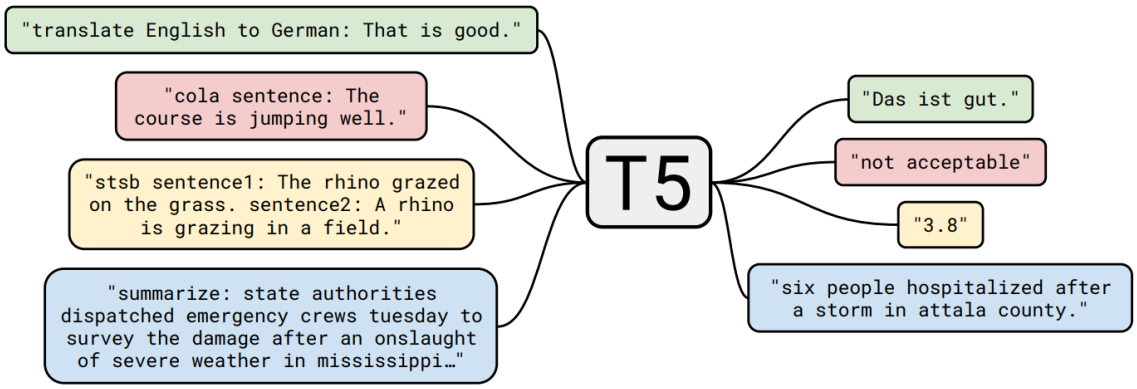

Figure 1.12: The T5 text-to-text framework, every task is cast to feed the model textual inputs and training it to generate some target text. In the image we see translation, classification, regression and generation tasks. Taken from [Raffel et al., 2019].

The idea behind T5 is to create a text-to-text framework that can be used for any kind of task. As we see in Figure 1.12, the objective is to have a single model that generates text, and use it to solve various kind of tasks, like classification, regression, generation etc. To do so, each task has to be cast to have both textual input and output. This means that for a classification task, each possible output class needs to be *verbalized*, i.e. each class needs to have a specific output sequence associated to it. For example, in a Sentiment Polarity classification task with two classes, POSITIVE and NEGATIVE, each need to be associated with some output sequence that the model will output when it want to predict that class (e.g. for POSITIVE the word *good* and for NEGATIVE the word *evil*).

To facilitate this kind of generalization capabilities in the pre-trained model, T5 is usually pre-trained in two phases, using different pre-training objectives:

1. **Masked Language Model** task: this self-supervised phase consists in corrupting parts of the text by masking some tokens. An example of the procedure is shown in Figure 1.13;

2. **Supervised** Tasks: the model is then trained on a series of supervised tasks, in particular: Sentence Acceptability Judgment [Warstadt et al., 2019], Sentiment Analysis [Socher et al., 2013], Sentence Similarity, or Paraphrasing [Dolan and Brockett, 2005, Cer et al., 2017, Iyer et al., 2017], Natural Language Inference [Williams et al., 2018, Rajpurkar et al., 2016, Dagan et al., 2006, Marie-Catherine de Marneffe, 2019], Sentence completion [Roemmele et al., 2011], Word Sense Disambiguation [Pilehvar and Camacho-Collados, 2019] and Question Answering [Khashabi et al., 2018, Zhang et al., 2018, Clark et al., 2019a].

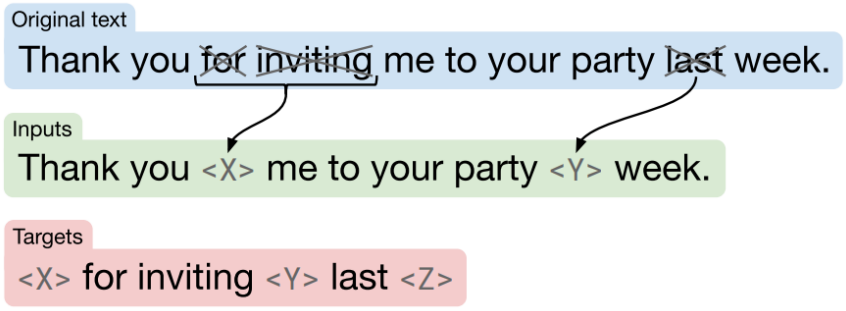

Figure 1.13: Scheme of the self-supervised Masked Language Model objective. In this example, in the sentence *"Thank you for inviting me to your party last week."* the tokens *"for inviting"* and *"last"* are randomly chosen to be masked. Each consecutive span of masked text is replaced by a *sentinel token* (here <X> and <Y>) that is unique to each input. Since *"for"* and *"inviting"* occur consecutively they are replaced by a single sentinel token. In the output sequence we expect the model to predict the masked tokens delimited by the sentinel tokens used to replace them plus a final sentinel token (here <Z>). Taken from [Raffel et al., 2019].

The use of supervised task in pre-training is then been expanded in one of the latest published version of T5, called Flan-T5 [Chung et al., 2022], where the objective was creating a text-to-text model that was trained in a few-shot setting to solve 1.843 different supervised tasks. This makes the model able to obtain more general capabilities so that it could then be used in inference without any further fine-tuning (zero-shot setting) to solve a variety of tasks successfully.

1.4 Attention Attribution Methods and Explainability

One of the main problems with Neural Networks is the so called *black box* problem. This problem refers to the fact that once the model starts to have hidden layers, and the inputs passes through multiple activation function, we don't know *how* the model is making its prediction, and we can't know in what way neurons are working together to respond to certain patterns. While this is already troublesome in tabular or continuous settings, is especially worse when dealing with text, since the model doesn't work *directly* with the words themselves, but with learned continuous representation that are usually built either by other deep-learning models, or by the model itself. To try and solve this problem, one of the most popular technique to understand what the model may be picking up from the text is to look at the Attention matrices. By looking at them we can see which words are attended the most, and we can also try and find syntactic patterns or other kind of relationship between words. While the technique seem promising and can provide a lot of insights on the inner workings of the model [Clark et al., 2019b, Hao et al., 2021], whether or not we can treat Attention patterns as an explanation for how the model behave is still debated in the scientific literature [Jain and Wallace, 2019, Wiegreffe and Pinter, 2019].

1.4.1 Value-Zeroing

Value-Zeroing is an explainability technique first introduced in [Mohebbi et al., 2023]. The method appears to draw inspiration from traditional techniques, where the influence of a feature (in this case, a token representation) on the model's output is extracted by removing that feature from the input. Since deleting a word from a sentence, without changing the semantics of it, is either challenging or impossible, the method opts to *eliminate it* during the Attention computation of the considered layer, by *zeroing* its Value vector, i.e. setting each element in the vector to 0. As we saw in Section 1.3.1, for each Attention head h, the input vector $\mathbf{x}_i$, for the i^{th} token in the sequence is transformed in three distinct vector through the use of different sets of weight: the Query vector $\mathbf{q}_i^h$, the key vector $\mathbf{k}_i^h$ and the Value vector $\mathbf{v}_i^h$. The context vector $\mathbf{z}_i^h$ for the i^{th} token of each

Attention head is generated as a weighted sum over the Value vector:

$$\mathbf{z}_i^h = \sum_{j=1}^{n} \boldsymbol{\alpha}_{ij}^h \mathbf{v}_j^h \tag{1.32}$$

where $\boldsymbol{\alpha}_{ij}^h$ is the raw Attention weight assigned to the j^{th} token and computed as a Softmax-normalized dot product between the corresponding Query and Key vectors. In Value-zeroing Equation 1.32 is changed by replacing the Value vector associated to j with a zero vector $\mathbf{v}_j^h \leftarrow \mathbf{0}, \forall h \in H$, where the context vector for the i^{th} token is being computed. This provide a new representation $\mathbf{x}_i^{\neg j}$ that has excluded j. By comparing the original representation $\mathbf{x}_i$ with this new one, usually by the means of a pairwise distance metric such as the cosine distance, we obtain a measure of how much the output representation is affected by the exclusion of j:

$$\mathbf{C}_{ij} = cs(\mathbf{x}_i^{\neg j}, \mathbf{x}_i) \tag{1.33}$$

Computing Equation 1.33 for each tokens i, j generates a **Value-Zeroing Matrix C** where the value of cell $\mathbf{C}_{ij}$ in the map indicates the degree to which the i^{th} token is dependent on the j^{th} to form its contextualized vectorial representation. Visualizing the matrix **C** we obtain an Attention map, where the importance of the Value vector is preserved.

2. Dataset and Models Description

In this chapter the dataset that has been used for the experiments will be introduced. The necessary cleaning and pre-processing steps will also be presented. Finally, the models used for the experiments we'll be discussed, along with how they have been used and modified.

2.1 TAG-it Dataset

TAG-it [Cimino et al., 2020] is a dataset for a profiling task presented at EVALITA 2020 [Basile et al., 2020]. The dataset is based on the corpus defined in [Maslennikova et al., 2019], and consists in 2.5 million posts written in Italian. These posts have been collected from different Italian forums. The main resource is the *ForumFree* platform, but other resources were used and aggregate to have more data (e.g. data from the *500x* forum and the *audiclub* were aggregated and inserted in the dataset by classifying those posts as AUTO-MOTO).

Attribute	**Description**	**Value**
Age	Age of the writer	0-19, 20-29, 30-39, 40-49, 50-100
Gender	Gender of the writer	M, F
Topic	Topic of the post	ANIME, AUTO-MOTO, BIKES, CELEBRITIES, ENTERTAINMENT, NATURE, MEDICINE-AESTHETIC, METAL-DETECTING, SMOKE, SPORTS, TECHNOLOGY

Table 2.1: TAG-it dataset target variables description.

The dataset is structured in collections of posts, where each training instance is a collection of posts written by the same author. The task is to predict three variables associated with the author and the posts. In fact, each collection is annotated with the Age and Gender

of the author and the Topic of discussion. In Table 2.1 are reported the semantics of the target variables.

2.1.1 Pre-processing

The division of the dataset in collections from the same author diminishes greatly the number of training instance available. To be able to have enough data for the fine-tuning of our models, we decided to shuffle all the posts in all the collections, and tag each individual post with the corresponding collection target variables. This, however, created a number of posts where the content, in the absence of the greater context provided by the other collection posts, wasn't representative of the target variables. This happened mostly to posts that contained too little text. We empirically decided to use a cutoff of 10 tokens and removed all the sentences from the new dataset that had less tokens than that. We kept the original train/test split as it was (70% training, 30% test). At the end of this process, we obtained a dataset consisting of 13.553 posts for the training set and 5055 posts for the test set.

2.1.2 Dataset Analysis

To prepare for potential problems during the experiments the distribution of the dataset has been studied (Figure 2.1) by plotting the number of available posts for each value of each task for the training set. As it can be noticed from the figures, the Age variable presents a quite balanced distribution among the five classes, especially for the three intervals between *30* and *100*. For what concerns the Gender task, we can observe that the majority of posts were written by male users, thus determining a strongly unbalanced distribution of the two classes. The last variable, Topic, presents 11 labels, with 3 of them (ANIME, SPORTS and AUTO-MOTO) having more than 2.500 posts each. The rest of the classes are somewhat equally distributed, except for TECHNOLOGY that has less than 100 posts. This is going to be an issue in the final part of our experiments, where the low number of sentences available for the class made the application of our Label Representation Selection method less effective.

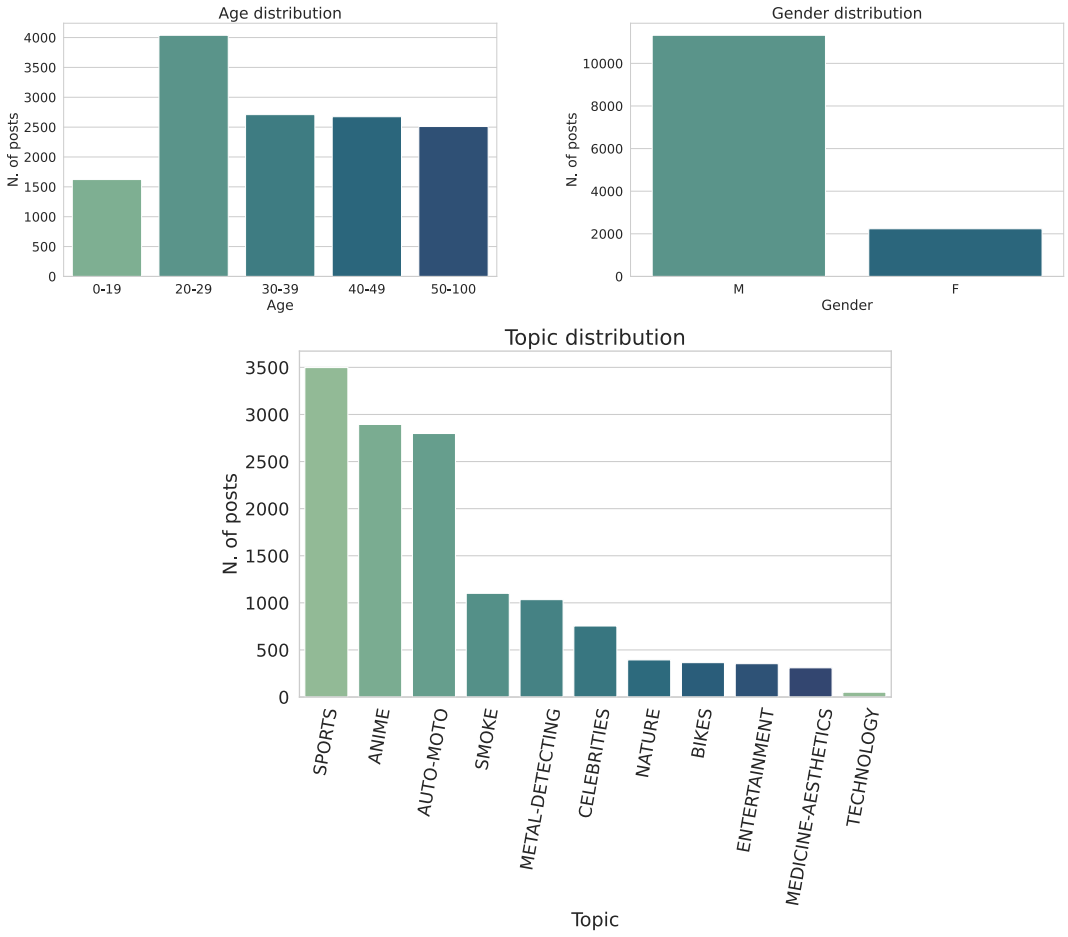

Figure 2.1: Distributions for the target variables of the dataset after the pre-processing steps.

2.2 Models: BERT and T5

As the main objective of our experiments being the use of the text-to-text framework for classification, we mainly used T5 during the experiments. In particular we used IT5 [Sarti and Nissim, 2022], an Italian version of T5 pre-trained on the cleaned Italian section of the mC4 Corpus [Xue et al., 2021]. In the first part of our experiments we've also compared its performances against the ones of an Italian BERT, available through the Transformers Python library[1], that was pre-trained on the Italian sections of the OPUS Corpora [Tiedemann, 2016]. In order to compare different architectures (e.g. T5 and BERT), it would

[1]Model available at: https://huggingface.co/dbmdz/bert-base-italian-cased

be ideal to analyze models with meaningful similarities, e.g. having a similar number of parameters or amount of computation to process an input-output sequence. Since T5 is an Encoder-Decoder model while BERT only has an Encoder stack, a T5 with n layers has approximately the same number of parameters as a BERT with $2n$ layers, but also the same amount of computational cost of an n-layers BERT. To try and achieve the fairest comparison of the two Transformers, we decided to use the base version of IT5 (220M parameters) against the base version of the Italian Bert (110M parameters), since the main reason to use smaller Transformer model isn't the number of parameter by itself, but the computational cost associated with the use of bigger models. All the code has been written in Python 3, and all the Transformers-related experiments have been done using the Transformers Library in its PyTorch version.

2.2.1 Multi-task BERT

In the first set of experiments, we compared the performances of BERT and IT5 in a multi-task setting, where they're trained to resolve all three tasks available in the dataset. While this is simple enough in a text-to-text setting (see Section 3.1.1), for BERT we had to modify the architecture to allow the model to make three prediction simultaneously.

We created a new class for the model, called `BertMultiTaskForSequenceClassification` that inherit from the class `BertPreTrainedModel` available through the Transformers Library. Then we changed the single Output Classifier, which is the final layer of BERT, by substituting it with three different Linear Layers on top of the Encoder stack:

```
class BertMultiTaskForSequenceClassification(BertPreTrainedModel):
    def __init__(self, config):
        super().__init__(config)
        self.num_labels=config.num_labels
        self.config=config
        self.bert=BertModel(config)
        classifier_dropout=(
            config.classifier_dropout if config.classifier_dropout
    is not None else config.hidden_dropout_prob
        )
```

```
        self.dropout = nn.Dropout(classifier_dropout)

        # The three new classifiers
        self.gender_classifier=nn.Linear(config.hidden_size, 2)
        self.topic_classifier=nn.Linear(config.hidden_size, 11)
        self.age_classifier=nn.Linear(config.hidden_size, 5)
        self.post_init()
```

Then, the `forward` method of the model needed to change accordingly. The `forward` method is the one called to pass inputs trough the architecture, both during training and inference. We needed to pass the Encoder output to the three new classifier, and once we obtained the logits, we had to change the calculation of the loss to account for the three different logits, each associated with one of the heads:

```
def forward(self, input_ids: Optional[torch.Tensor] = None,
    attention_mask:    Optional[torch.Tensor] = None, token_type_ids
    : Optional[torch.Tensor] = None, position_ids: Optional[torch.
    Tensor] = None, head_mask: Optional[torch.Tensor] = None,
    inputs_embeds: Optional[torch.Tensor] = None, gender_labels:
    Optional[torch.Tensor] = None, topic_labels: Optional[torch.
    Tensor] = None, age_labels: Optional[torch.Tensor] = None,
    output_attentions: Optional[bool] = None, output_hidden_states:
    Optional[bool] = None, return_dict: Optional[bool] = None):

    # Obtaining the output of the Encoder
    outputs=self.bert(
        input_ids,
        attention_mask=attention_mask,
        token_type_ids=token_type_ids,
        position_ids=position_ids,
        head_mask=head_mask,
        inputs_embeds=inputs_embeds,
        output_attentions=output_attentions,
        output_hidden_states=output_hidden_states,
        return_dict=return_dict,
```

```
    )
    pooled_output=outputs[1]
    pooled_output=self.dropout(pooled_output)

    # Obtaining the logits from the three classifiers
    gender_logits=self.gender_classifier(pooled_output)
    topic_logits=self.topic_classifier(pooled_output)
    age_logits=self.age_classifier(pooled_output)

    # New Loss Calculation
    loss=0.0
    loss_fct=CrossEntropyLoss()
    loss+=loss_fct(gender_logits, gender_labels)
    loss+=loss_fct(topic_logits, topic_labels)
    loss+=loss_fct(age_logits, age_labels)

    return MultiTaskSequenceClassifierOutput(
        loss=loss,
        gender_logits=gender_logits,
        topic_logits=topic_logits,
        age_logits=age_logits,
        hidden_states=outputs.hidden_states,
        attentions=outputs.attentions,
    )
```

As we can see from the code, the new loss is simply calculated as the sum of the `CrossEntropyLoss` obtained from the outputs of each head. This is then back propagated through the network during training.

2.2.2 T5 Enconder for Value-Zeroing

In Section 3.3 a Label Representation Selection method will be presented that, to function, needs to use an explainability technique called Value-Zeroing (See Section 1.4.1). To be able to do, we need to have a T5 model that can *zero* specific indexes of the Value vector

during the calculation of the Self-Attention layer output in the Encoder. To do so, in the `forward` method of the `T5AttentionLayer` of the Python Transformers library we added a `zero_value_index` between the parameters:

```
def forward(self, hidden_states, mask=None, key_value_states=None,
    position_bias=None, past_key_value=None, layer_head_mask=None,
    query_length=None, use_cache=False, output_attentions=False,
    zero_value_index=None):
```

Then, before weighting the Value vector with the calculated Attention scores, we zero the provided index:

```
if zero_value_index is not None:
    value_states[:, :, zero_value_index] = torch.zeros(value_states
    [:, :,zero_value_index].size(),device=scores.device)
```

And then we use that `value_state` tensor to calculate the Attention output:

```
attn_output = unshape(torch.matmul(attn_weights, value_states))
attn_output = self.o(attn_output)
```

Then we used the modified model to extract the Value-Zeroing score Matrix for each sentence in the dataset:

```
# The Score Matrix is initialized
score_matrix = np.zeros((config.num_hidden_layers, seq_length,
    seq_length))

for layer_index, layer_module in enumerate(model.encoder.block):
    if layer_index in layers:
        for t in range(seq_length):
            extended_blanking_attention_mask: torch.Tensor =
            model.get_extended_attention_mask(
                inputs['attention_mask'], input_shape
                )
```

```
            with torch.no_grad():
                layer_outputs = layer_module(
                    org_hidden_states[layer_index].unsqueeze(0),
                    attention_mask=extended_blanking_attention_mask,
                    output_attentions=False, zero_value_index=t
                    )


            # compute similarity between original and new outputs
            x = layer_outputs[0].squeeze()
            y = org_hidden_states[layer_index + 1]

            distances = cosine_distances(x, y).diagonal()
            score_matrix[layer_index, :, t] = distances
```

As we can see, the algorithm is pretty simple: for each sentence we iterate through each layer and for each token t we calculate the Value-Zeroing score of every other token, to assess how the token t contributes to the representation of the other part of the sentence. We obtain a Score Tensor with dimensions [*number of layer*, *max sentence length*, *max sentence length*].

Finally, we apply to the Score Tensor a technique called Attention Rollout [Abnar and Zuidema, 2020], where we aggregate the Tensor across the *number of layer* dimension by doing a point wise multiplication of the slices. We then use as scores for the tokens the value taken from the aggregated matrix.

3. Text-to-Text models for classification tasks

In this chapter the main experiments of the thesis are going to be discerned. In particular, the first section will focus on how evaluating the performance of text-to-text model for classification tasks, and how do they perform against models specifically built for classification. Then, it will also touch on the problem of how important are label representation for text-to-text models in classification scenarios.
In the second section, the focus will be on a particular classification task where label representations are highly important. A thorough testing of different label representations and their effect on the performance of the model will be presented. There will also be qualitative analysis of the relationship between certain characteristics of the chosen representations and the score the models trained on them achieved.
Finally, in the third section, a novel Label Representation Selection technique, based on Attention attribution methods, will be presented and evaluated, both from a performance point of view and from a qualitative perspective by a thorough analysis on the representations obtained with this method.

3.1 Evaluation of Text-to-Text model for Classification

The purpose of these first experiments was to evaluate how well we could use text-to-text models for classification tasks. In particular, being the task in the Italian language, we decided to evaluate the first text-to-text Transformer model developed for the Italian language, IT5 [Sarti and Nissim, 2022], against a series of baselines. The experiments were performed in two different classification scenarios: single-task and multi-task, and we compared the performances of IT5 against those obtained with an Italian version of BERT, a model specifically built for classification. Finally, following the findings in [Chen et al., 2020], we performed a more in-depth analysis to test the impact of label representations in the chosen tasks.

3.1.1 Experimental Settings

The experiments were performed on two different classification scenarios: i) single-task and ii) multi-task classification. For what concerns the single-task scenario, we fine-tuned both BERT and IT5 three times in order to create three different single-task sequence classification models, one for each task. To perform fine-tuning with the BERT model, we converted the three target variables into numeric label, associating a unique index to each value of the target variable. For T5, the target variables were verbalized empirically, by manually choosing a label representation for each possible value:

- Gender: values have been transformed in *uomo* and *donna*;
- Topic: values have been translated in Italian, written in lowercase and truncated into a single word (e.g. MEDICINE-AESTHETIC into *medicina*), thus resulting in the following list: *anime*, *automobilismo*, *bici*, *sport*, *natura*, *metalli*, *medicina*, *celebrità*, *fumo*, *intrattenimento*, *tecnologia*;
- Age: we didn't change the representation and we kept the original classes as their own representations.

Moreover, following the *Fixed-prompt LM tuning* approach (see [Liu et al., 2021] for an overview), we added a prefix to each input when fine-tuning the IT5 model. This approach implies providing a textual template that is then applied to every training and test example. Fixed-prompt LM tuning has been already successfully explored for text classification, allowing more efficient learning [Schick and Schütze, 2020, Schick and Schütze, 2021, Gao et al., 2021]. In our experiments, we tested three different prefixes, one for each classification task: *"Classifica argomento"*, *"Classifica età"* and *"Classifica genere"*. Concerning instead the multi-task classification, each sentence has been presented three times during the training phase of the two models, each one with the appropriate label and, in the case of IT5, with the appropriate prefix. This further shows the importance of having a task-related prompt, since the multi-task model can learn which of the three task to do based on the presented prefix.

We relied on two different typology of models as baseline. The first one is based on two *dummy* classifiers: i) *most frequent classifier*, Dummy (MF), which always predicts the most frequent label for each input sequence, and ii) *stratified dummy classifier*, Dummy(S), that generates predictions by respecting the class distribution of the training data. Moreover, in order to assess the impact of the pre-training phase of the two Transformer models, we also used an Italian BERT and an IT5 model with randomly initialized weights that we called *BERT Random* and *IT5 Random*. We used F-Score (macro and weighted) as evaluation metric for all the experiments, and all the models have been parameterized the same and trained for the same amount of training steps.

3.1.2 Results

Model	**Topic**		**Age**		**Gender**	
	Macro	**Weighted**	**Macro**	**Weighted**	**Macro**	**Weighted**
Dummy (S)	0.09	0.17	0.20	0.22	0.50	0.68
Dummy (MF)	0.04	0.10	0.09	0.14	0.44	0.69
BERT Random	0.14	0.34	0.26	0.27	0.56	0.74
IT5 Random	0.14	0.34	0.20	0.26	0.36	0.74
BERT	**0.50**	**0.64**	**0.32**	**0.33**	**0.76**	**0.84**
IT5	0.19	0.41	0.16	0.22	0.31	0.70
Multi-task						
MT BERT	**0.56**	**0.67**	**0.32**	**0.33**	**0.75**	**0.84**
MT IT5	0.31	0.52	0.16	0.23	0.33	0.71
Best-Performing Model Evalita 2020						
MT UmBERTo	—	0.64	—	0.39	—	0.86

Table 3.1: Macro and Weighted average F-Score for all the models and according to the tree classification variables. In the last line part of the table, it has also been reported the best performing model from the original EVALITA shared task, for which the dataset was created. Their Macro F-Scores weren't available. In **bold** the best performing model for each metric and task.

Classification results are reported in Table 3.1. We've also reported the best performing model at the EVALITA 2020 TAG-it shared task, for which the dataset we used was created. It's important to note that we've used the data in a different way with respect to how was supposed to be used from the task (see Section 2.1.1), with the main difference being that for the original task less training instances were available, but each had longer context. The best performing model at the shared task was an UmBERTo model [Parisi et al., 2020], which is a BERT-like model with a different pre-training objective, trained in a multi-task setting (similar to what we did for our multi-task Bert, Section 2.2.1). The winning system was described in [Occhipinti et al., 2020] and we reported here their weighted F-Scores for all three tasks (*MT UmBERTo*). It's interesting to observe that both our single-task BERTs reached the same results as them in the Topic classification task, while our multi-task BERT surpassed it. For Age and Gender, however, their performances were slightly better.

For our systems we can observe, instead, that Transformer models outperformed the dummy baselines in almost all the classification tasks. The only exception concerns the performance of IT5 on the Age prediction task, for which the stratified dummy classifier obtained the same scores. It should be considered that the Age classification task appears to be the most complex task, regardless of the model taken into account. In fact, the best performing model (BERT) obtained only 0.11 points more than the baseline. The complexity in predicting the age ranges could be due to the fact that the task requires more sophisticated information than those that the model can extract from linguistic clues and writing styles. These don't seem to be enough for these models to infer the age range of the writer.

On the other hand, on the other two tasks, Gender and Topic, the classifier achieved better results. This is in line with [Cimino et al., 2020], where the authors suggested that textual clues seem to be more indicative of these dimensions rather than Age. Moreover, the higher scores obtained for the Gender classification task could also be indicative of the fact that, differently from the other two, gender prediction was cast as a binary task and thus is easier to predict.

When we look at the performances obtained by the randomly initialized BERT and

IT5, we note that the latter achieved results close to those of the pre-trained models. Indeed, in some cases, like IT5 on the Age and Gender prediction tasks, the Random model gets better results. This seems to suggest that the pre-training phase of IT5 did not allow the model to encode enough useful information in order to improve its performance on the selected tasks. On the other hand, the pre-training phase had a strong impact on BERT performances, since the pre-trained model outperformed the Random one in all classification tasks.

If we focus, instead, on the differences between the two models, we can clearly notice that BERT performed best in all configurations. In particular, IT5 achieved fairly reasonable results in comparison with BERT for simpler tasks, such as Gender and Topic classification. For what concerns the Age prediction task instead, we observed a performance drop, with a difference in terms of weighted F-Score of 0.17 points. A possible explanation for this behavior could be due to the fact that, differently from BERT, T5 has to produce the label by generating open text, thus making the prediction more complex from a computational point of view. In this regard, it is important to notice that for our experiments we relied on the base version of IT5, which, despite being bigger in terms of parameters than BERT base, is still quite smaller than the best-performing model (T5-11B) presented in [Raffel et al., 2019]. Another possible explanation is that the representations for the Age class were semantically too similar to one another, and also complex from a number of sub-tokens point-of-view.

It should be pointed out that in some cases IT5 generated labels that did not belong to those we defined, but which actually turned out to be more accurate than the original ones. This is the case, for instance, of a few posts labelled with *fumo* (English: *smoke*) that were predicted by IT5 with the label *tabacco* (English: *tobacco*). We also found that sometimes IT5 was not able to generate meaningful labels, but rather produced only punctuation marks or single letters. Nevertheless, we only identified a few isolated cases of them (less than 5 for what concerns Topic classification), which had no real impact on the overall performances of the model.

It is also interesting to point out that the IT5 Random model did not generate unexpected labels like the pre-trained one did. This could be another motivation for its better perfor-

mances in the two cases of Age and Gender classification.

Observing the results obtained in the multi-task setting (MT BERT and MT IT5), we notice a significant increase in the performances of IT5. In fact, while BERT achieved a consistent boost only in the Topic prediction scenario, IT5 performances improved significantly in all classification tasks, with an average improvement of around 0.06 points more (in terms of weighted F-Score) than during single-task classification. This is particularly evident with regard to Topic and Age classification, while the scores obtained for the Gender prediction task remained roughly the same. This result could suggest that, besides having more data for the fine-tuning phase, the IT5 model particularly benefits from learning multiple tasks at a time, thus improving its generalization abilities.

3.1.3 Label Representation Analysis

As described in the original T5 paper, one of the issues of using text-to-text models for classification tasks is that the model could output text that does not correspond to any of the possible pre-defined label representations for a certain task. While this happened in our experiments, in some cases it seems that IT5 was able to generate more appropriate labels than the one that were originally assigned to the post during the construction of the dataset, thus suggesting some kind of generalization ability of the model. For instance, as we can observe from the examples in Table 3.2, the labels predicted for the three input posts are not among those expected for the Topic prediction task (See Section 3.1.1). Nevertheless, by looking at the posts, the labels predicted by IT5 might be considered more appropriate choices as Topics tag for each text.

Inspired by such behavior, we decided to further investigate the generalization abilities of the IT5 model by measuring the impact of different label representations on model performance. More specifically, we wanted to understand if changing the representation to the model could affect its capabilities of correctly predicting the right classes. To test this, we produced a shuffled version of each dataset by randomly shuffling all the labels of a certain category with another. For Topic we randomly shuffled the categories representation (e.g. we used *medicina* to represent the AUTO-MOTO class), for the binary Gender prediction task we used *maschio* to represent the F class and *femmina* to represent the M

Sentence	Predicted Label	Correct Label
“Che bell’acqua e che bei vitellini! Grande Pres.!”	animali	celebrità
“Perchè non l’alcool alimentare essendo neutro? E costa pure meno”	alcool	fumo
“terza miscela svizzera champagne eccellente! non vedo l’ora di tornare da two lions per altre miscele”	bevande	fumo

Table 3.2: Examples of IT5 predictions.

class. For the Age classification task, with the target variable being ordinal, we shuffled them by trying to maximize the distance between the original class and the new one, i.e. we used *30-49* for 0-19, *40-49* for 20-29, *50-100* for 30-39, *0-19* for 40-49 and *20-29* for 50-100. The results of this experiment is reported in Table 3.3.

Model	Topic		Age		Gender	
	Macro	**Weighted**	**Macro**	**Weighted**	**Macro**	**Weighted**
IT5	**0.19**	**0.41**	**0.16**	**0.22**	**0.31**	**0.70**
IT5 shuffled	0.07	0.17	0.11	0.17	0.29	0.69

Table 3.3: Macro and Weighted F-Scores for the classification tasks obtained with IT5 using correct and shuffled labels (*IT5 shuffled*). In **bold** the best performing model for each metric and task.

As we can see, the most significant variations in model performance concern the Topic and Age classification tasks. In particular, we can observe a drastic performance drop for the Topic task, with a difference between the predictions on the original and shuffled datasets of more than 0.24 points in terms of weighted F-Score. Moreover, it is interesting to note that the scores obtained with the shuffled labels are also lower than those obtained by the randomly initialized IT5 (0.17 vs. 0.34). This result seems to suggest that the IT5 model is indeed able to learn some specific lexical correlations between the encodings of the input tokens and the encodings of the labels during the fine-tuning phase, and that these correla-

tions are no longer observable after the shuffling process. This is also corroborated by the fact that, when presented with shuffled data, the model stopped generating new and more specific labels for the input sequences as it was doing in the previous experiments. From the results we can also understand that these lexical connections between the sentence and the chosen label are essential to maximize the model performances, and for certain tasks, like Topic, the absence of these connections make the model unable to reach satisfactory performances altogether. These findings are what motivated the next set of experiments that will be presented in Section 3.2.

If we look, instead, at the results obtained with the Gender dataset, we can notice that shuffling the labels does not have a significant effect on the performance of the model. This is a clear evidence that, unlike Topic, the Gender prediction task does not present a direct lexical connection between the input sequence and the label. As a result, the model tends to memorize the information available in the fine-tuning data rather than derive generalities exploiting the knowledge learned during the pre-training phase, and as such, any label works for the model. To further prove this, we conducted another experiments on the Gender classification task by trying a different set of label representations: we changed from *uomo* and *donna* to *m* and *f*. As shown in Table 3.4, modifying the label representation did not affect the performance of IT5, which obtained basically the same results in both configurations. This seems to confirm once again that for tasks that do not show an explicit relationship between input samples and labels, the choice of the label largely does not affect model performances.

Labels	**Macro**	**Weighted**
m/f	**0.32**	**0.70**
uomo/donna	0.31	**0.70**

Table 3.4: Macro and Weighted F-Score on the Gender prediction task using *m/f* and *uomo*/*donna* as target variables. In **bold** the best performing model for each metric.

This is in line with the findings of [Chen et al., 2020], where they found that the importance of the label representations is **task-dependent**. In fact, we similarly find that changing the representation to the Topic classification task has a direct impact on its per-

formances, while for the Gender classification task the string used to represent the gender doesn't have a meaningful impact on the capabilities of the model.

3.2 Impact of Label Representation on Model Perfomances

With the previous set of experiments we established that comparably sized text-to-text models can achieve performances on par with classification model like BERT, especially in a multi-task setting, where more context is provided. We've also assessed that how we represent the label is especially important for some tasks where the model seems to be able to draw clues from lexical connection between the input and the text it should generate. In our experiments, we saw that the Topic classification task seemed to be deeply affected by this, since when the representations were shuffled the model wasn't able to learn how to solve the task.
In this section, we tried to see *how much* different label representations impacted the model prediction capabilities, and while doing so, we tried to find a way to determine beforehand which representation could work best. The idea was to try and find some kind of metric or heuristic that could be used as a Label Representation Selection method. To achieve this we fine-tuned a IT5 again, just on the Topic classification task, using a number of different label representations. Then, we evaluated the results gaging the impact of different label representation and with those results we then tried to find some correlations between the model performances and some metrics.

3.2.1 Experimental Settings

As introduced in Section 3.2, to investigate the influence of label selection on the model performance, we fine-tuned the IT5 model using different combinations of strings to represent the original classification categories. We will refer to the set of the original categories,

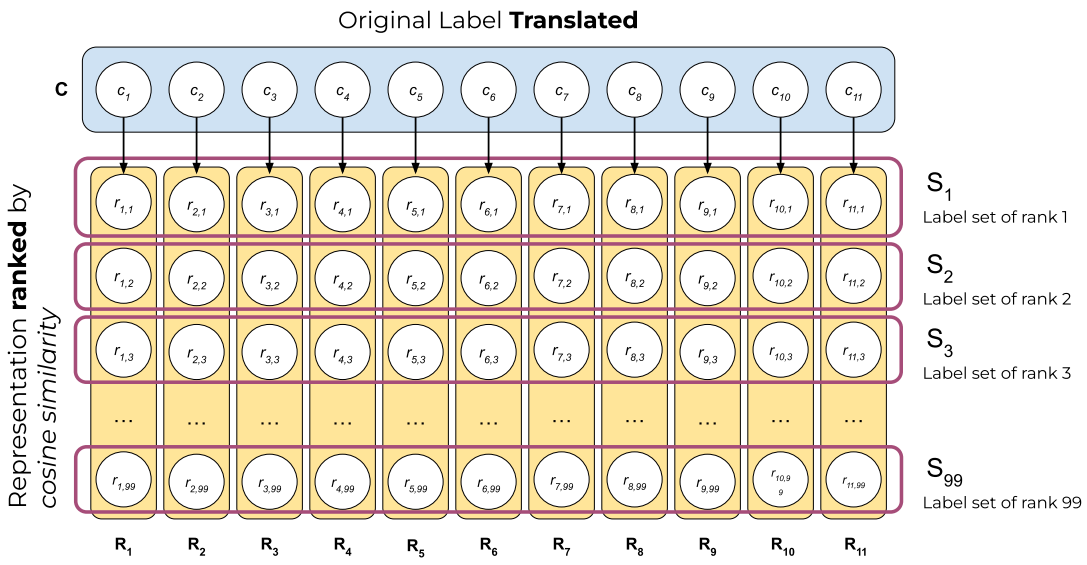

Figure 3.1: The framework for the creation of the different sets of labels S_j ranked by cosine similarity.

translated in Italian, with C[1]. For each category c_i in C we created a set R_i, composed by 100 string representations: 10 were selected from synonyms and related words to the original categories (including aforementioned translated ones), while the remaining 90 were randomly chosen from the most frequent nouns in the ItWac corpus [Baroni et al., 2009]. Let $R_i = \{r_{i0}, r_{i1}, ..., r_{i99}\}$ be the set of labels for the category c_i, and r_{ij} be the j^{th} label in the set. Then, for each category c_i, we ranked its corresponding set of labels R_i in descending order of similarity:

$$cs(c_i, r_{i0}) \geq cs(c_i, r_{i1}) \geq ... \geq cs(c_i, r_{i99}) \tag{3.1}$$

where $cs(c_i, r_{ij})$ is the cosine similarity between the average embedding[2] of the sub-tokens of c_i and r_{ij}, extracted from the last Encoding layer of the IT5 model.

Given the previously defined sets R_i, which contains the elements ranked by similarity, we created 100 sets of labels S_j (where j ranges from 0 to 99). Each set is defined as

[1]List of translated labels: *anime*, *automobilismo*, *bicicletta*, *sport*, *natura*, *metal detector*, *medicina*, *celebrità*, *fumo*, *intrattenimento* and *tecnologia*.

[2]For embedding we used the activation of the last layer of the Encoder stack of IT5 after seeing in inference the tokens of the chosen representation.

$S_j = \{r_{0j}, r_{1j}, ..., r_{10j}\}$, where e.g. r_{0j} is the j^{th} ranked label for category c_0. As a consequence, S_0 contains the labels that achieved the highest cosine similarity with the original categories, while S_{99} is the set containing the lowest cosine similarities. A graphical overview of the experimental setting is shown in Figure 3.1.

We then fine-tuned IT5 for each ranked set of representations S_j. Each model was trained for 10 epochs using F-Score as the evaluation metric, resulting in 100 models, where the model with rank 0 is the model trained with the original set of labels C, the model with rank 1 is the model trained with S_1 (the set of representations with the highest cosine similarity to C) and the model of rank 99 is the model trained with S_{99} (the set of representations with the lowest cosine similarity to C).

3.2.2 Results

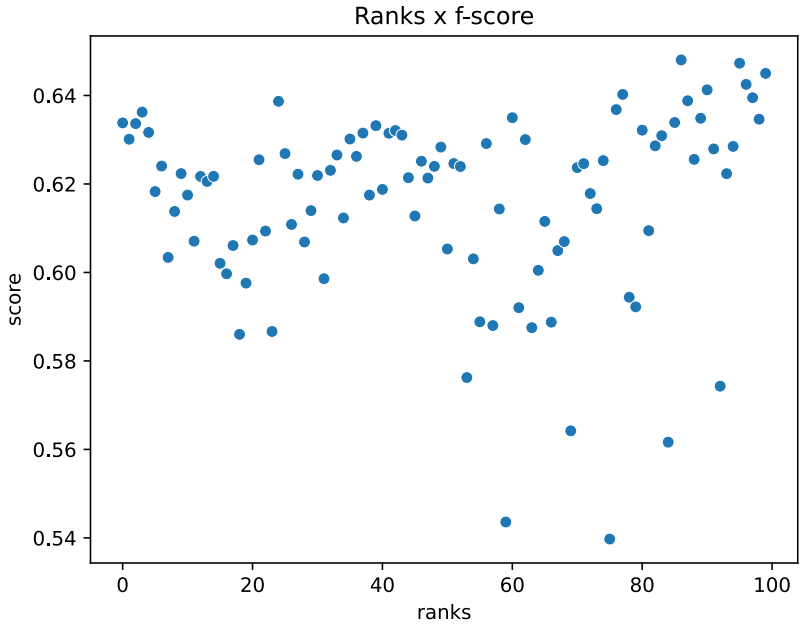

Figure 3.2: Scatter-plot between the rank of the different sets of labels S_j against the IT5 weighted F-scores obtained by the model trained on that set.

In Figure 3.2 we have a summary of the results obtained by the T5 models fine-tuned on the Topic classification tasks according to the 100 different sets of labels (S_j). At first

glance, we can readily observe that the choice of words used to represent the classification categories has a considerable impact on the model's average performance. Indeed, we can see that the classification scores vary significantly, ranging from a minimum of 0.54 (rank 75) to a maximum of 0.65 (rank 86). Additionally, it is worth noting that the model trained with S_0, which contains the original translated labels, achieved an F-score of 0.63. This result indicates that simply using the original translated labels directly still provides a competitive performance. However, the significant fluctuations in the classification scores among the different sets S_j suggest that certain labels may still offer better performances than the original ones, while others may introduce noise or ambiguity, resulting in sub-optimal outcomes.

Interestingly, these findings appear to diverge from previous studies, e.g. [Chen et al., 2020], where the role of label representation was underestimated. While being a task-dependent issue, the role of label representation seems to have a large impact on model performance, especially for lower frequency labels, going as far as making certain labels range from being completely unpredictable to reaching satisfactory performances. We will discuss more on this in the next paragraphs.

That being said, despite the differences in terms of weighted F-scores, there does not seem to be a clear correlation between the model's performance and the degree of *semantic* distance between the chosen labels and the original ones (represented by the rank j of the Representation Set). In fact, as the cosine similarity decreases between the selected representations and the original ones (from rank 0 to rank 99), there is no apparent trend in F-score values.

Per-label results

In order to gain a more precise insight into the impact of the tested labels, Figure 3.3 illustrates the variation of F-scores obtained with the 100 different sets of labels (S_i) for each individual category. Firstly, we can observe that the average results can vary significantly depending on the category under consideration. For instance, IT5 shows promising average performances in classifying posts related to ANIME, SPORT or AUTO-MOTO, while encountering difficulties in identifying posts annotated with the topics BIKES and

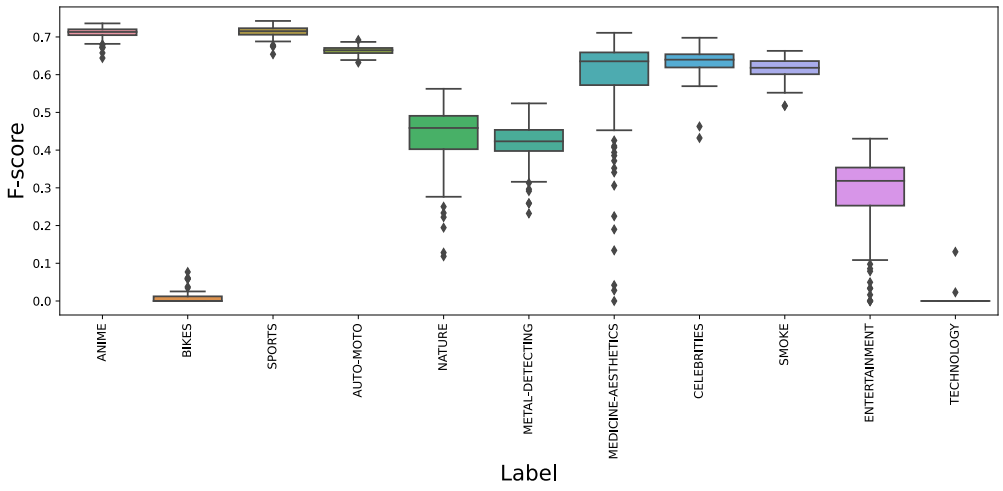

Figure 3.3: Boxplot showing the variation of the F-scores using different labels according to each classification category.

TECHNOLOGY. This is possibly due to the fact that the posts belonging to the former categories are the most frequent in the entire dataset. Particularly noteworthy is the fact that, across almost all tested ranks, the model failed to correctly identify any posts related to TECHNOLOGY. This issue is likely attributed to the limited representation of this category within the dataset, further compounded by the original dataset configuration having more examples in the test set than in the training set (51 and 85 samples in the training and test sets respectively).

Analyzing the variation of results based on the labels used for representing the categories, we observe that the choice of the label often has a significant impact on the model's performances. While some labels exhibit relatively stable results with minor variations across different representations, such as ANIME, BIKES, SPORT and AUTO-MOTO, there are other instances where the selected labels lead to remarkable fluctuations in the model's performances. Notably, this behaviour emerges especially in the identification of posts related to NATURE, METAL-DETECTING, MEDICINE-AESTHETICS and ENTERTAINMENT. For these categories, IT5's classification performances can change drastically depending on the specific label. In some cases, the model manages to achieve quite good results, accurately classifying posts with a high F-Score. However, in other instances it struggles significantly, making erroneous classifications for the majority of cases. For

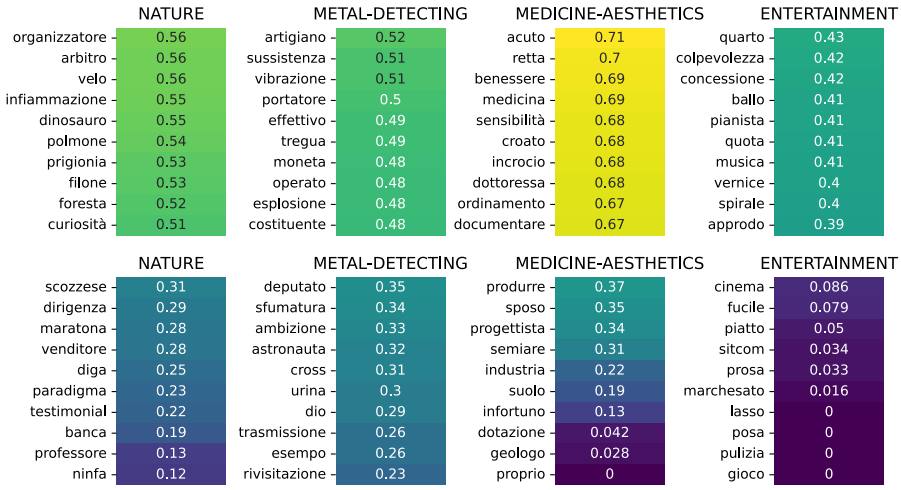

Figure 3.4: Top and bottom 10 labels that maximize/minimize the results for the most varying categories (Nature, Metal-Detecting, Medicine-Aesthetics and Entertainment).

instance, in the case of MEDICINE-AESTHETICS, the F-Score reaches a maximum of 0.71 when the label is represented by the term *acuto* but it fails to correctly classify any instance (F-Score = 0) when the label is represented as *proprio*. This highlights the importance of exploring optimized selection strategies to maximize the model performance.

To obtain a more comprehensive qualitative perspective of these findings, we include, in Figure 3.4, the top and bottom 10 representations that maximized/minimized the F-Score values for the four aforementioned categories. As we can observe, among the four considered categories, only one (MEDICINE-AESTHETICS) contains the original label, i.e. the one with cosine similarity equal to 1 (*medicina*), in the top 10 representations. For the other categories, the absence of the original label seems to suggest that the chosen word for the label, which should be the closest one to the reference topic, may not be the one that can maximize the results. When analyzing individual words, it becomes evident that not all words contributing to the model's best performance belong exclusively to the domain of the considered category. Surprisingly, words such as *cinema* and *sitcom*, seemingly related to the ENTERTAINMENT domain, are among those that most negatively impact the model's F-Scores. Nevertheless, MEDICINE-AESTHETICS shows an exception, with several words aligned with the category's domain, e.g. *benessere*, *medicina*, *dottoressa* e *sensibilità*. Lastly, it is worth noticing that the performance drop

is mostly label-dependent, and there is a significant difference between the most- and least-performing representations for the four categories. In fact, while NATURE and METAL-DETECTING exhibit a relatively modest decrease (around 0.20 F-Score points), MEDICINE-AESTHETICS and ENTERTAINMENT display a far more pronounced difference in performances.

3.2.3 Relationship between Representations and Performances

Categories	**Spearman**	**p-value**
Entertainment	0.29	0.003 *
Auto-Moto	0.05	0.62
Medicine-Aesthetics	-0.02	0.85
Bikes	-0.05	0.61
Anime	-0.10	0.37
Technology	-0.12	0.21
Smoke	-0.20	0.04 *
Sports	-0.22	0.03 *
Nature	-0.25	0.01 *
Metal-Detecting	-0.35	0.00 *
Celebrities	-0.45	0.00 *

Table 3.5: Spearman correlations between F-Scores and label similarities (cosine similarity) for each category. Statistically significant correlations are marked with *.

Having analyzed the model's performances and assessed the impact of words used to represent the categories on the classification results, we decided to explore the existence of any relationship between the model's performances and the employed words.

Semantic Similarity

Initially, we aimed to ascertain whether there is a correlation between the words that are more/less semantically similar to the original categories and the performance of IT5. To

achieve this, we computed the Spearman correlation between the T5 model's performance and the cosine similarity values calculated to construct the 100 sets for each label S_j.

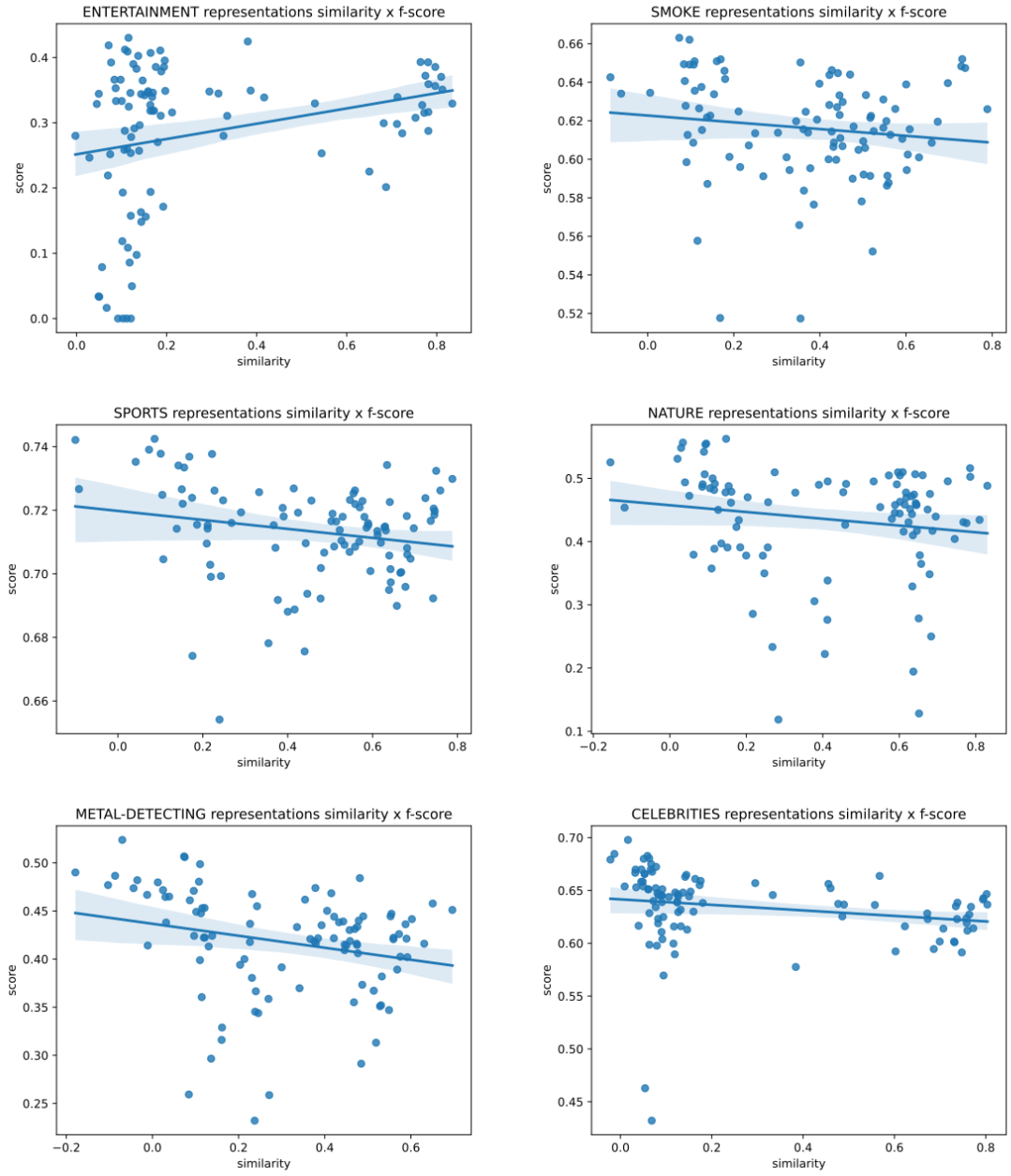

Figure 3.5: Scatter-plot showing the relationship between F-Scores and cosine similarity values for the 6 categories that exhibited a statistically significant correlation.

The results of these correlations are presented in Table 3.5 and their scatter-plot can be seen in Figure 3.5. As observed, 6 out of the 11 classification categories exhibit sta-

tistically significant correlations. Among these, only one correlation is positive (*Entertainment*), while the others show negative correlation values. This outcome is quite unexpected as it seemingly implies that the improvement in the model's performance is linked to a decrease in semantic similarity. However, it is crucial to emphasize that the correlation values are not particularly high, and thus we cannot draw any conclusion about these results. Moreover, it is important to consider that, while cosine similarity can serve as a useful measure of similarity between embeddings, it may not encompass the entire semantic space.

Internal Similarity

Since the similarity between selected labels representations within each set could potentially impact the model's performances, we conducted an additional test to investigate whether higher semantic similarity among representations within a set could negatively affect the performance of IT5. To achieve this, we computed the *inner similarity* of each set, defined as the average cosine similarity of all possible distinct label combinations[3]. Subsequently, we computed the Spearman correlation between each set's inner similarity and the F-Scores obtained by the model fine-tuned with it. Although the values of inner similarities vary considerably across the sets (ranging from a similarity of 0.69 for rank 0 to 0.38 for rank 100), we did not find a statistically significant correlation with the model's performances (Spearman = 0.01, p-value = 0.90). These results suggest that, despite the sets exhibited considerable variation in terms of inner similarity, the similarity between the representations didn't plainly affect the model's performances.

Representations Frequencies

Finally, since the aforementioned results have demonstrated that different labels have an impact on the model's performances, we decided to investigate whether this impact could be somehow related to the frequency of these representations within the model's training dataset. To this end, we computed the absolute frequency of each label used in our experi-

[3]As defined in Section 3.2.1, a label is represented as the average embedding of each subtoken in the string.

ments (11 labels per 100 sets, totalling 1.100 words) within the Italian version of the mC4 Corpus, i.e. the corpus on which IT5 was trained.

Categories	**Spearman**	**p-value**
Medicine-Aesthetics	0.13	0.20
Nature	0.06	0.54
Sports	0.04	0.66
Bikes	0.01	0.94
Technology	-0.02	0.88
Anime	-0.02	0.84
Entertainment	-0.03	0.75
Auto-Moto	-0.05	0.62
Metal-Detecting	-0.06	0.57
Celebrities	-0.06	0.54
Smoke	-0.25	0.01 *

Table 3.6: Spearman correlations between f-scores and labels absolute frequencies (computed in the Italian mC4 Corpus) for each category. Statistically significant correlations are marked with *.

Subsequently, we calculated the correlation between the scores obtained by IT5 for each label of each set R_i and the corresponding frequencies of each label found in the mC4 corpus. Among the eleven categories present in the dataset, only one showed a statistically significant correlation, *Smoke*, with a Spearman correlation value of -0.25. This result suggests that, at least for this particular category, a decrease in the label's frequency in the training corpus corresponds to an increase in the model's performance. However, the fact that only one category exhibits a significant correlation, and that this correlation is not particularly high, once again prevents us from drawing any conclusive findings.

3.3 Attention-based Label Representation Selection method

Having assessed with the previous experiments that varying the representation can heavily impact the performances of the model on the Topic classification task, we decided to test a new possible Label Representation Selection method. As we did for the previous experiment, we tested the new method by selecting a number of different possible representations for the categories of the Topic classification task, and then trained IT5 on those representation. This was done to see if the representation score, given by this new method, could correlate with the performances obtained by the model using those representations.

In particular, we tried three different Attention-based methods to select possible label representations from the available training set. The idea was that we could look at which tokens were the most salient in the construction of specially placed tokens in the sentence, and then use the most salient tokens as representations. Doing this, we could find suitable representations for our labels directly from the words used in the training set, and the words we end up choosing would also be words that, in that context, seem to be preferred by the model to construct representation of other important tokens. The hope is that the chosen representation should help the model to create the lexical connection between input and output in a easier manner, thus resulting in a easier *learning environment* and higher F-Scores.

We chose as the specially placed tokens in the sentence, the translated class names, or a variation of those (see Section 3.3.1), and we appended that at the end of the sequence, to study which tokens are the most salient to construct their continuous representation. To do this, instead of looking directly at Attention weights, we used a technique presented in Section 1.4.1 called Value-Zeroing.

We chose to use Value-Zeroing because, as it's already attested in the scientific literature, using the Attention weight to directly gauge the saliency of a certain token is based on the assumption that the larger is an Attention weight of a vector, the larger will be its contribution. While this assumption is not wrong, it also completely disregards the impact

of the transformed input vector (the Value vector, in particular). A big Value vector with a small Attention weight could contribute more to a representation than a very small vector with a high Attention weight [Kobayashi et al., 2020]. Value-Zeroing takes into consideration both the Attention weights and the Value vector of each tokens, and as such we believed to be better suited to indicate which part of the sentence are more salient to build the representation of our specially placed tokens.
After finding which tokens is the most salient according to Value-Zeroing, we use it as a representation for the particular class associated to the sentence it was extracted from.

3.3.1 Experimental Settings

We define $S \in D$ as one of the training sentences for our model, in the dataset D. The sentences are tokenized using the provided IT5 trained tokenizer T. For each sentence S, we injected a series of tokens p tokenized with T. The objective is to study which tokens from the original sentence S are more salient for the model to build the representation of the tokens in p. The difference between the three methods we've tested is how p is defined:

- In the **Appended Label** method, we define p as the translation of the original class names[4], e.g. the sentence *"Che giornata indimenticabile... è passato proprio tanto tempo!</s>"* from category SPORTS, becomes: *"Che giornata indimenticabile... è passato proprio tanto tempo! Sport</s>"*;
- In the **Appended Label with Prompt** method, we provide the model additional context, by defining p as: *La frase precedente appartiene alla categoria* (English translation: *The previous sentence belongs to the category of*) followed by the original class name translated, e.g. the sentence *"Che giornata indimenticabile... è passato proprio tanto tempo!</s>"* from category SPORTS, becomes: *"Che giornata indimenticabile... è passato proprio tanto tempo! La frase precedente appartiene alla categoria Sport</s>"*

[4]List of translated labels: *anime*, *automobilismo*, *bicicletta*, *sport*, *natura*, *metal detector*, *medicina*, *celebrità*, *fumo*, *intrattenimento* and *tecnologia*.

- In the **End Of Sentence** (EOS) method, we don't append any new tokens to the sentence, and we define p as the special EOS (*</s>*) token, e.g. for the sentence *"Che giornata indimenticabile... è passato proprio tanto tempo!</s>"* no new token is appended;

After injecting p in each $S \in D$ we pass each sentence in inference through a modified IT5, whose Encoder is able to calculate the Value-Zeroing matrix (see Section 2.2.2). Then, we focus on which token $s \in S$ has the highest Value-Zeroing score, with respect to the tokens in p.

$$s = max(value\_zeroing\_score(S, p))$$

By doing so we obtain, for each sentence, the most important token whose embedding vector is used to construct the representation of p[5]. After doing it for the whole dataset, we obtain, for each category $c \in C$, a list of representations R_c. R_c contains a number of representations r_i equal to the number of sentences in the dataset tagged with c. Each list is composed by tokens s that obtained the highest Value-Zeroing score vz in their sentence, with respect to p, $R_c = [(s_1, vz_1), ..., (s_n, vz_n)]$. Since some of these representations may be duplicate, in the sense that the same token appeared in multiple sentences, and had the highest Value-Zeroing score, we decided to aggregate those representations, in a way that rewards their higher frequency count. We define b as a subset of R_c where all the representations s are the same:

$$b = \{(s_i, vz_i), (s_j, vz_j), ..., (s_k, vz_k) | s_i = s_j = ... = s_k, (s_i, vz_i), (s_j, vz_j), ..., (s_k, vz_k) \in R_c\}$$

We aggregate all the representations in b by creating a single entry in the list R_c, (s^*, vz^*) where the score is defined as the sum of the scores:

$$vz^* = \sum_{(s,vz) \in b} vz \tag{3.2}$$

[5]In reality, we obtain a token which could not be a full word, but just a subpart of it. This is an issue that arise from how the words are tokenized in Transformers models. To obtain the full word, we reconstruct it by reconnecting all the tokens that are part of the word that the token with the highest Value-Zeroing score is from. The Value-Zeroing score we consider for the full word is the one of the token that was selected. We decided to avoid to aggregate the score of the full word in any way, because that could reward or punish multi-token words unjustly.

After doing this aggregation steps for all possible subsets b in R_c, and for every category $\forall R_c : c \in C$, we have, for each category c, a set of representations R_c that we sort based of vz in descending order, obtaining a ranked Representation Set.
Finally, we define a set of representations E_i, called the Representation Set of rank i where, for each category c, we have the i ranked representation r_i in R_c. E.g. in the set E_0, for each category we have the best ranked representation, while in the set E_{10}, for each category we have the representation that ranked 11th.

The method has been applied to the training dataset using all three methods of defining p. We then evaluated their performances by testing the first ten best performing Representation Sets $E_0, E_1, ..., E_9$ for all three methods, training ten models for each, using those set of representations as the target labels, for a total of 30 trained models.

Then, after establishing that the End Of Sentence Method worked best, we decided to evaluate it by using all the Representation Set we had extracted, which totalled to 23 models. Since we were limited to only 23 Representation Sets because of the TECHNOLOGY category (which has a low representation in the training set, and only 23 representations were extracted), we decided to exclude the TECHNOLOGY category and all its sentences from D, and we evaluated the method with the rest of the categories by using the first 100 Representation Sets, $E_0, E_1, ..., E_{99}$ training another 100 models. All the models were IT5, with the same hyper-parameters as before.

3.3.2 Results

First, we evaluated the first ten Representation Sets $E_0, ..., E_9$ from each method to try and see if one of them looked promising for finding a way to predict which representations are going to work best. As we can see from Figure 3.6, the first two methods, Appended Label and Appended Label with Prompt, don't show any particularly interesting trend. The first one has a slightly negative coefficient, and a Spearman Correlation of 0.03 with a p-value of 0.934. With such a low correlation value and high p-value we can't reject the null hypothesis and the obtained trend is probably random. The same can be said for the second method too, where we have a slightly positive trend, with a Spearman correlation of 0.151 with a p-value of 0.67.

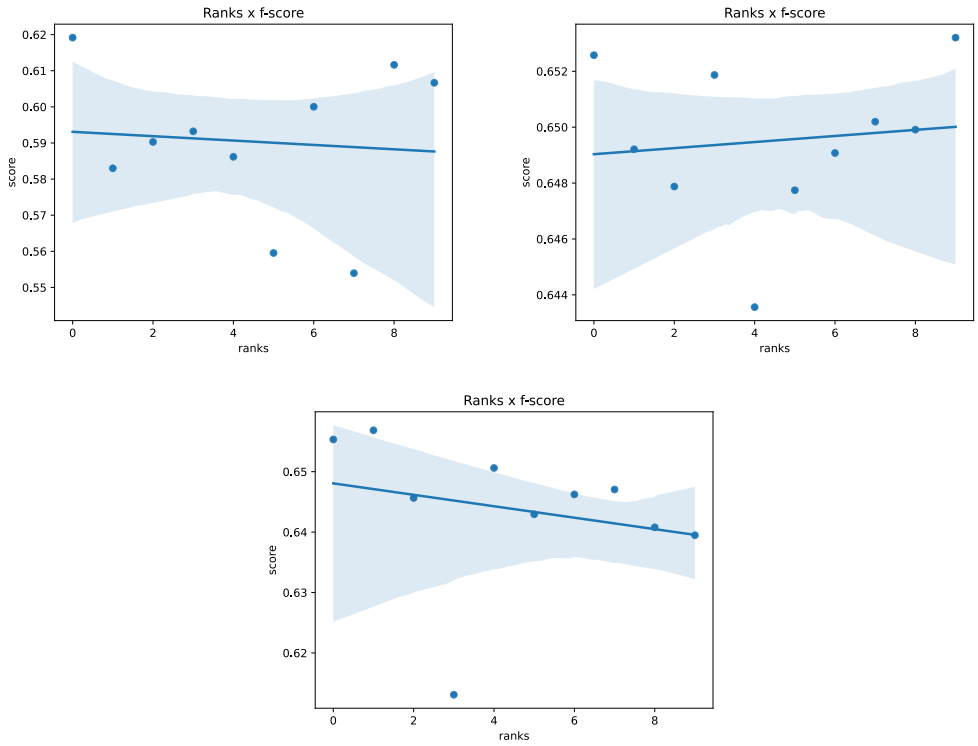

Figure 3.6: Scatter-plots with regression lines where each point is a model. On the y-axis we have the weighted F-Score on the test set and on the x-axis we have the rank of the Representation Set used to train it. On the top-left we have the Label Appended method, then on the top-right the Appended Label with Prompt method and on the bottom the EOS method.

The third method, however, is a bit more interesting: there is a much more pronounced negative trend, so, as the rank increases, the representations that have ranked lower also have a negative effect on the F-Score, lowering it as well. The correlation is also promising, with a Spearman rank of -0.552 and a p-value of 0.098. While the p-value doesn't reach the standard cut-off value of 0.05, and thus we can't reject the null hypothesis, with the correlation being higher, and the trend much more apparent, we decided to keep testing the EOS method with more representations.

First, we used all 23 representations that we had extracted from the training set. We could extract only those representations because the TECHNOLOGY class has a very low frequency in the training set, and thus we only had 23 sets that contained a label

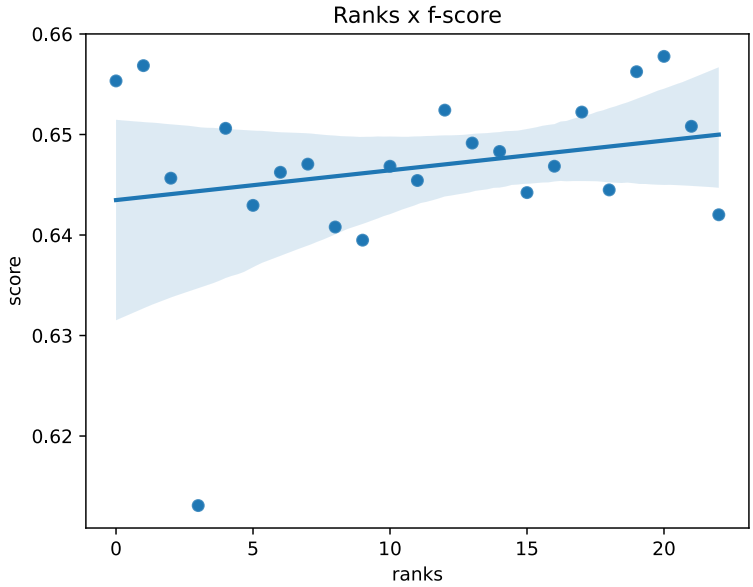

Figure 3.7: Scatter plot for the first 23 Representation Set extracted with the EOS method.

representation for that class. To both try the method on the complete dataset with all the classes, and with a more significant number of Representation Sets, we decided to test it first on all the 23 Representation Sets where TECHNOLOGY is included, and then remove all TECHNOLOGY sentences from the dataset to extract a 100 Representation Sets for the remaining classes. Then, 100 models were trained with those.

As we can see from Figure 3.7, the trend seems to be different from before and became slightly positive, with a Spearman rank of 0.123 and a p-value of 0.578. While these results seem to overturn the previous ones, we need to keep in mind that they could be a spurious outcome due to the low number of points used. In fact, from this experiment and the ones in Figure 3.6, we can't really draw conclusions on the goodness of the proposed methods, and these experiments should just be seen as preliminary tests done to decide which next steps to take. Being that the EOS method was the only one showing interesting results, we decided to continue the experiments and testing it on a higher number of Representation Sets.

Testing the method on 100 sets revealed a correlation between the Representation Sets rank we've extracted and the performance of the model trained on it. In particular, we have

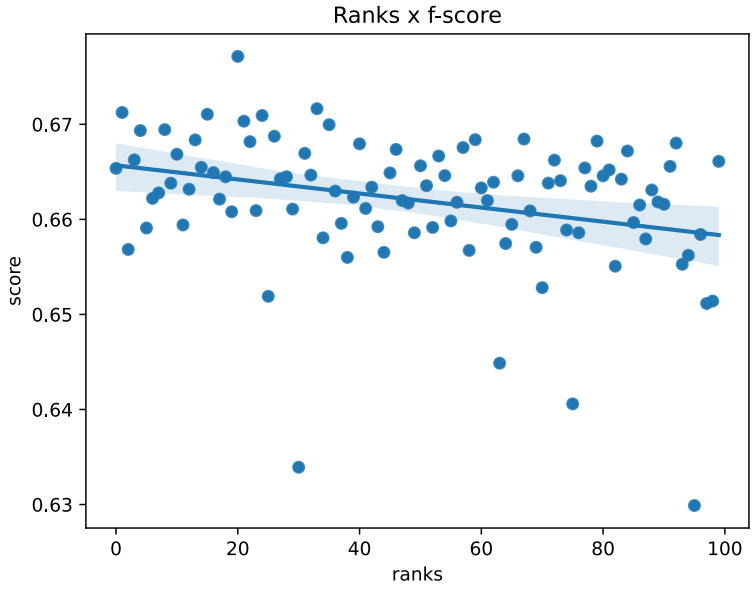

Figure 3.8: Scatter plot for the first 100 Representation Set extracted with the EOS method from the training set where the low frequency TECHNOLOGY class was removed.

a Spearman Rank of -0.314 with a statistically significant p-value of 0.001. While we can't draw general conclusions about the method, it appears that, in this setting, selecting labels from the training set using Attention attribution techniques like Value-Zeroing, is a helpful way to identify keywords in the data that have meaningful semantic connections that IT5 can exploit to achieve higher performances. Using the EOS method we obtained a difference of 0.05 in terms of F-Score between the highest F-Score value (0.68) obtained by rank 20, and the lowest F-score value (0.63) obtained by rank 95. It's interesting to note that the worst performing model using the EOS Method reached the same results as using the simple class names translated, which obtained an F-Score of 0.63. On this, is also interesting to see that the Label Appended model, in the first ten representation ranks we tested, never achieved above 0.62 of F-Score, while the Label Appended with Prompt, while not showing any apparent correlation with the F-Score, in the first ten representation ranks was consistently above 0.64 of F-Score with all its models, with the highest being rank 9, reaching 0.653. The EOS method is superior to just using the class names translated: in fact, if we focus on the Representation Set of rank 0, which is the one the method

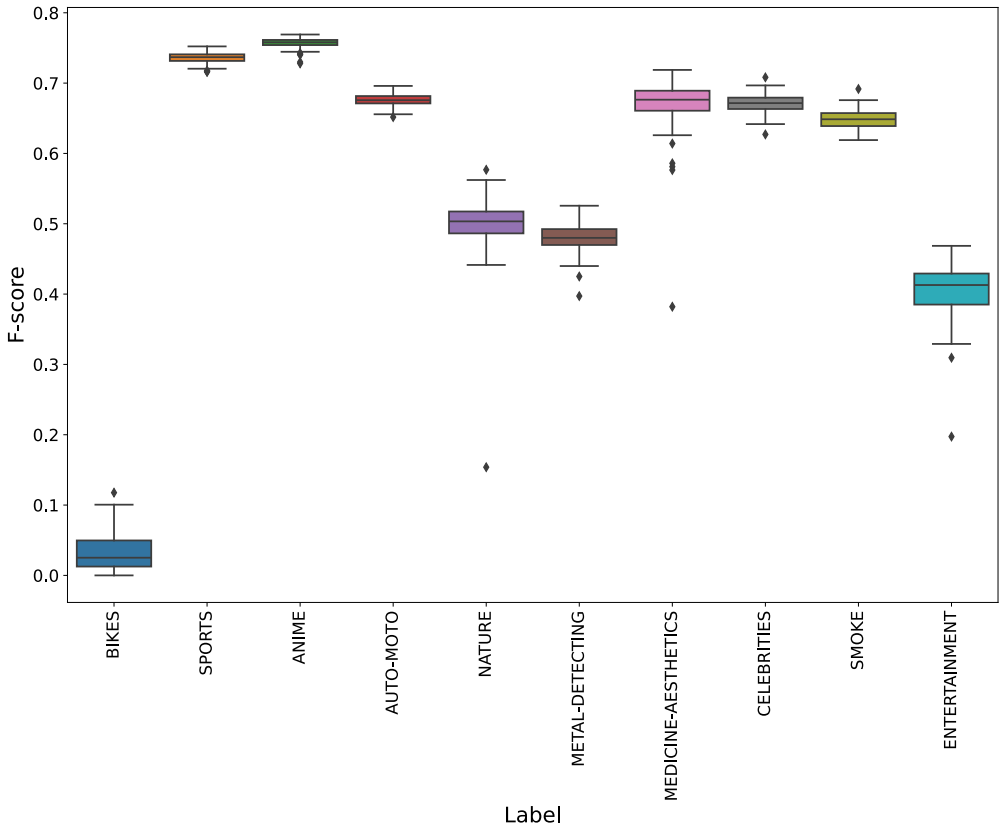

Figure 3.9: Boxplot for each Category to show the variations in F-Score obtained using the various Representation Sets.

indicates as the best, the model trained on that obtained a score of 0.656. While not being the best Representation Set the method has produced, still surpassed the standard approach of simply using the class names (weighted F-Score of 0.63).

In Figure 3.9 we can see the variation in F-Scores obtained with each class. As is the case for the overall weighted F-scores, it appears that the average F-Scores for each class are higher than when we did the previous experiments with Representation Sets composed by 10 synonyms and 90 random words (See Section 3.2.2 and Figure 3.3). The effect is much more pronounced in the lower frequency categories, where we have both a higher average F-Score and a lower variance. For example, in the first experiment the MEDICINE-AESTHETICS class had a a large number of outliers, some of which resulted in very low F-scores values (down to 0). With this technique this does not happen, and even considering the outliers, the class performances remain decent even in the worst

scenarios. The same behaviour applies to the ENTERTAINMENT class.

On why the EOS methods outperforms the other two, it seems that, for building the </s> character, the Encoder of the model uses particularly informative words that we can leverage if used as label representation. The role of the EOS character and other similar characters that are used for modeling purposes, like the *[CLS]* character in BERT-like models, is to be used as input for the final Language Modeling classifier. That pushes the model during the pre-training phase to learn to construct a representation of such token that summarize all the relevant informations in the sentence that are needed to complete the language modeling task [Clark et al., 2019b]. So, by taking the highest Value-Zeroing score for constructing </s>, we find tokens that are usually very contextually informative to the language modeling task and contains a lot of useful information. It's likely, then, that this information is also useful during the fine-tuning phase, to construct that lexical connection between input sentences and output classes. It must also be said that, when using the other two techniques, we focus on injected tokens that are often appended without sufficient context to justify their presence in the end of the sentence. It may be that appending tokens to the end of the sentence may change the semantics of the sequence too much. The first method, that simply appends the label to the end of the sentence, often creates scenarios where the word appears to be out of place. The same applies for the second method, but thanks to the prompt, this effect is less noticeable. This effect may also be the reason why the first method is the worst performing one, while the Appended Label with Prompt method achieve F-Scores almost as high as the EOS one but whithout showing any useful correlation between the chosen representation and the model F-Score, thus not being usable as a Label Representation Selection method.

3.3.3 Qualitative Representations Analysis

In Table 3.7 are reported the representations for each class for the best and worst performing models. Similarly to the experiments in Section 3.2, there doesn't seem to be any apparent pattern about which words work best, both from a morpho-syntactically and a semantically point of view.

In the best performing set, the only two words that are somehow related to their class

Class	Best Performing Representation Set (Rank 20) F-Score: 0.68	Worst Performing Representation Set Rank (95) F-Score: 0.63
BIKES	risolvo	temperatura
SPORTS	schedina	decidesse
ANIME	troverai	principiante
AUTO-MOTO	premuto	abbassato
NATURE	gippi	causarne
METAL-DETECTING	cosa	pistolina
MEDICINE-AESTHETICS	capelluto	soffermarmi
CELEBRITIES	ilaria	scherzo
SMOKE	eccola	piaciuto,proviamo
ENTERTAINMENT	origini	dragonette

Table 3.7: Table showing the representations for the best performing and worst performing set in the first 100 Representation Sets extracted from the training set where the sentences of the TECHNOLOGY class were removed.

seems to be *schedina* for SPORTS, referring to the betting ticket used to bet for sports games, and *ilaria* for CELEBRITIES, referring to the proper noun that could be the name of some famous Italian celebrity. For the worst representation, the only in-domain word we find is, again, a proper noun: *dragonette*, which is the name of a Canadian band. Another interesting word is *piaciuto,provato*, which is a typo where a space was missing after the comma. While the tokenizer divided the word into multiple sub-tokens, no special space character was found in the tokens that formed the word, and our system correctly re-built it to be a single word. Our heuristic also rewarded frequency, nonetheless, of the best performing representations only four of these representations had been chosen as the most salient token in the text multiple times: *schedina* (3 times), *troverai* (2 times), *premuto* (2 times), *gippi* (2 times). This could mean that we should re-evaluate how important frequency is, and maybe change the heuristic of aggregation (See Equation 3.2) to something that doesn't reward the frequency as much.

To better understand the role of the representations frequencies in the training set we've calculated both the raw frequency of each representation in the whole dataset, and the TF-IDF of the representations. We've then calculated the Spearman Rank between the frequencies, the TF-IDFs, the obtained F-Score and the Representation Set rank the representations are in. The TF-IDF has been calculated by treating all the documents of a single category as a single document, and the documents' length has been calculated as the number of tokens inside it. We've obtained ten documents with the following lengths:

- ANIME: 154.347;
- BIKES: 14.078;
- SPORTS: 163.142;
- AUTO-MOTO: 118.859;
- NATURE: 21.595;
- METAL-DETECTING: 40.722;
- MEDICINE-AESTHETICS: 16.070;
- CELEBRITIES: 25.555;
- SMOKE: 41.040;
- ENTERTAINMENT: 18.216;

By looking at Table 3.8, we can see that Frequency doesn't correlate to any class with the obtained F-Score. This, again, confirms that the role of the absolute frequency of a certain term inside the training set doesn't seem to have any positive nor negative effect on the ability of the model to use such representation for its classes. However, by using an aggregation formula that rewarded frequency, we can see the effect that the absolute frequency had on the placement of the representation. In particular, for two classes (SPORTS and AUTO-MOTO) more frequent representations had a higher chances to be placed in the best ranks. This could mean that in these particular categories, the most informative words are frequently the same.

Class	F-score x Frequency	Rank x Frequency	F-Score x TF-IDF	Rank x TF-IDF
BIKES	-0.080	0.080	-0.118	0.007
SPORTS	0.138	-0.307*	0.213*	-0.499*
ANIME	0.125	-0.101	0.283*	-0.408*
AUTO-MOTO	0.147	-0.346*	0.081	-0.290*
NATURE	0.049	-0.026	0.148	-0.159
METAL-DETECTING	-0.053	-0.128	0.146	-0.204*
MEDICINE-AESTHETICS	0.152	-0.149	0.025	-0.067
CELEBRITIES	-0.182	-0.014	0.031	-0.329*
SMOKE	0.030	-0.034	0.080	-0.388*
ENTERTAINMENT	-0.099	-0.005	-0.103	0.004

Table 3.8: Spearman ranks between the F-Score and Representation rank against the raw frequency of the representations in the dataset, and the TF-IDF of the representations calculated as all the sentence of a certain category are part of a document. Statistically significant correlation are marked with *.

For the TF-IDFs, instead, we've got two minor correlations with the F-Score. It seems that for SPORTS and ANIME, using in-domain words, that don't appear as often in the other categories, had a slightly positive impact on performances. We've also empirically observed that SPORTS and ANIME are also the two categories where our model chose a lot of domain-specific words. In fact, the first ten ranked representations for the two categories are all domain-specific words:

- for SPORTS: *campionato*, *gol*, *pareggio*, *centrocampo*, *milan*, *juventus*, *atalanta*, *tifosi*, *trequartista* and *derby*;
- for ANIME: *streaming/download*, *graffio*, *manga*, *pokémon*, *pokemon*, *ko*, *morso*, *pokèmon*, *cmq*, *drago*[6].

[6] *morso*, *graffio* and *ko* are all domain-specific words in the settings of the popular anime, cartoon and video-game Pokémon, with the first two being moves and the latter being a specific status.

Again, the correlation between the TF-IDF and the representation rank was to be expected, based on the heuristic we've used to aggregate the representations. We've also empirically noticed that the method we've used to extract the representations was keen on choosing domain-specific, low frequency words. Which is why often it chose typos and similar words with errors in them. This is probably due to the fact that low frequency words are usually *full words* with much more semantic value in them, and by being domain-specific they carry high contextual information, useful for constructing the other tokens' representations. This could explain why the TF-IDF, which is a metric that is specifically built to find such words, correlates so highly and significantly with the extracted words' ranks.

Class	Subtoken length x F-Score	Subtoken Lenght x Ranking
BIKES	-0.031	0.029
SPORTS	-0.166	0.303*
ANIME	0.059	-0.071
AUTO-MOTO	-0.408*	0.126
NATURE	0.113	0.132
METAL-DETECTING	-0.130	0.181
MEDICINE-AESTHETICS	-0.116	0.074
CELEBRITIES	-0.071	0.210*
SMOKE	-0.090	0.127
ENTERTEINMENT	0.116	0.048

Table 3.9: Spearman ranks between the F-Score and representation ranks against the number of sub-tokens each representation is built with. Statistically significant correlations are marked with *.

In Table 3.9 we've also tried to look if there is any correlation between both the F-Score and the Representation rank with the number of sub-tokens of the representations. In fact, Transformers models tokenize their text by splitting them into sub-word parts; this is done to optimize the Vocabulary size, while, at the same time, keeping some meaningful

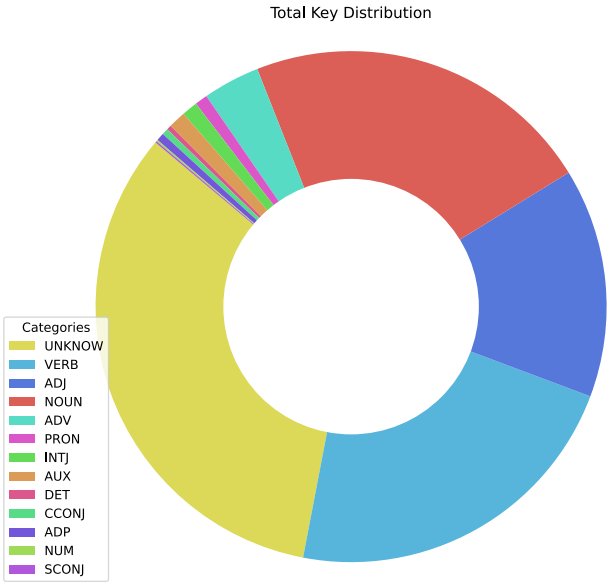

Figure 3.10: Distribution of the Parts-of-Speech across all the representation extracted using the EOS method.

morpho-syntactic structure of the words. From our results, we can see that subtoken length doesn't seem to affect neither the model's performances nor our selection technique seem to be have any kind of preference towards words that are split into more or less quantity of sub-tokens. The only two exception are AUTO-MOTO, where a higher number of sub-tokens lead to a decrease in performances, and SPORTS, where our model seemed to place words with a higher subtoken number to lower places in the ranking system.

We've also looked at the Part-of-Speech for the representations, both globally (See Figure 3.10) and on a per-class basis (See Figure 3.11), to see if we could find any interesting preferences in our method representation choice. The PoS are extracted from an Italian Word Form Vocabulary developed by the Institute for Computational Linguistics (ILC) of the National Research Council of Italy (CNR), that contains all the word forms and their possible Parts-of-Speech from the Italian language. We can see that, globally, the most frequent PoS are UNKNOWN, VERB, NOUN and ADJ. The class UNKNOWN contains the words that are not found in the Word Form Vocabulary, and these usually consists of

typos, English words, proper noun, etc. and are going to be seen more in detail for each category. The category with the highest number of UNKNOWNs are ENTERTAINMENT, CELEBERITIES, ANIME and SPORTS:

- in ENTERTAINMENT, the majority of the UNKNOWNs are typos (e.g. *cioe* instead of *cioè*), abbreviations (e.g. *nnt* instead of *niente*), words with an increased vocal length in the last character (e.g. *iniziaaaaaa* instead of *inizia*) or english words (e.g. *wish*);

- in CELEBRITIES, the majority are proper nouns (e.g. *alessia*, *mirco*, *federica*, etc.) and typos;

- in ANIME, the majority are proper nouns of video-games or tv shows characters (e.g. *pokémon* or *charmender*) or Japanese words (e.g. *manga*);

- in SPORTS, the majority are proper nouns of soccer teams or players (e.g. *milan*, *juventus*, *higuain*, etc.) or match names composed by multiple teams or nation names (e.g. *italia-uruguay* or *brasile-olanda*) that our system didn't split since they didn't contain any spaces.

We can also see that for BIKES, NATURE and AUTO-MOTO more VERBs are chosen instead of NOUNs, while for METAL-DETECTING, SPORTS and SMOKE the opposite is true. That being said, all the Parts-of-Speech seem reasonably distributed and it seems that no particular one is preferred by the method when choosing representations from the training set.

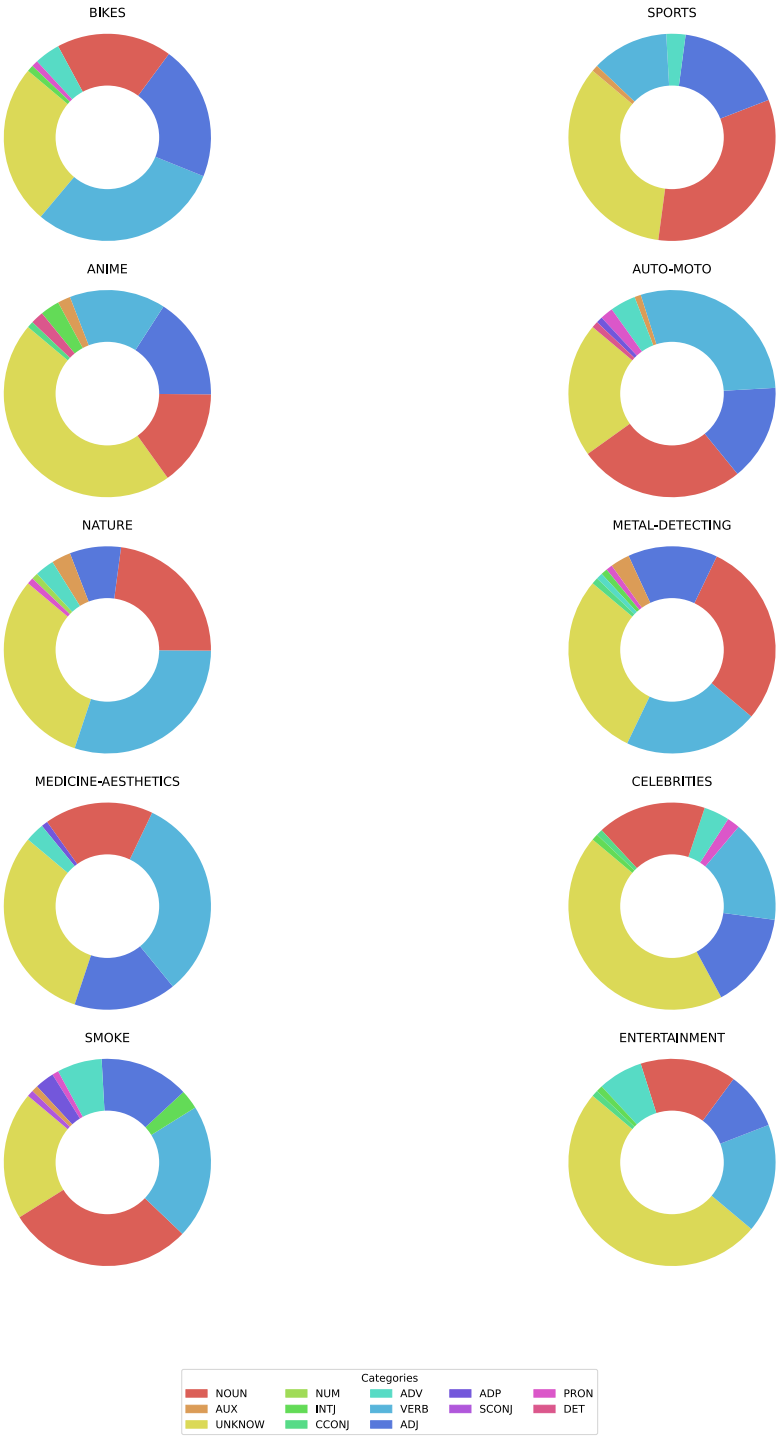

Figure 3.11: Distribution of Parts-of-Speech per Category of the representations extracted using the EOS method.

Conclusions

In recent years, with the advent of the Transformers architecture, a revolution has begun in the field of Natural Language Processing (NLP). These new Neural Language Models aim to produce probability distribution over the natural human language, which has a lot of intrinsic properties, like the fact that is a dynamic system of communication, with loose rules and endless possibilities for creativity, that makes it very hard to model. These models succeed in this challenging tasks by creating dense representations of words that also take into consideration the context those words are found in. With the huge success of this architecture, text generation models, or text-to-text models, became better than ever before, and gained a lot of attention.

With this new framework reaching new state-of-the-art performances, [Raffel et al., 2019] proposed to use it as a unifying method to solve all kinds of NLP tasks. By taking any task, and casting it to have both textual input and textual output, this model could be trained to solve it, with the advantage of having just one framework to work with. That could enable the comparison of different tasks, different pre-training objectives, different training datasets, etc.

In this thesis, we challenged this idea by asking if reasonably sized text-to-text models could be used to do classification tasks, which are a type of problems that aren't usually solved by generation models. Then, we also tested how important the way we cast this problems into textual form is, by focusing on how to *verbalize* the class values into text sequences that the model can produce. Finally, we presented a method that finds label representation from the training set of the task by using Attention attribution methods.

Can Text-to-Text Models classify successfully?

We answered this first question during the first of the presented experiments. We evaluated the performance of IT5, a text-to-text model pre-trained on the Italian language, against a series of baselines in various classification tasks taken from TAG-it, an EVALITA 2020 shared task. The objective was to classify the gender and age of the author and the topic

of discussion of a single forum post. We tested IT5 and an Italian version of BERT in single- and multi-task scenarios, and also against an IT5 and a BERT without pre-training. The results showed that while BERT outperformed IT5 in both scenarios, the text-to-text model still achieved fairly reasonable results, especially in multi-task. Also, we found out that BERT pre-training had a huge impact on its performance, while IT5 pre-training didn't seem to affect the model as much, with the non pre-trained model actually scoring better results in the Gender and Age classification tasks. IT5 also showed some interesting behaviour in the Topic classification task, sometimes generating labels that weren't in the set of pre-determined ones, that empirically resulted more appropriated for the text they were generated for. Finally, some preliminary tests using a shuffled dataset where all the labels for a certain class were associated to another one (e.g. *medicina* was associated to AUTO-MOTO, instead of MEDICINE-AESTHETICS, for the Topic classification task) showed that, for the Topic classification task, having a meaningful lexical connection between input and generated output is of huge importance for performances.

How important are label representation for performances?

Having assessed that text-to-text models are reasonable to use for classification, we focused on the importance of having different representations for the class names.
We experimented on that by creating 100 sets of label representation for the Topic classification task. We first chose 10 synonyms of the class names and 90 random words from the ItWac corpus for each class, and then ranked these 100 words in descending order using the cosine distance between these representation and the class name they should represent. Then we trained 100 models using these representation sets, with the set of rank 0 containing the most similar representations to each class, and rank 99 containing the least similar representations to each class. A hundred IT5 models were trained using those representation and we observed a remarkable variation in performances depending on the representation used. We couldn't find any correlation between the cosine similarity of the representations to their class name and the performance of the model trained on them. Nonetheless, we can say that for this task there is an important variation of performances based solely upon the way we represent the labels of the target variable, and to be able to

use this relatively small text-to-text models for classification is of high importance to find a way to choose this representation without having to train a bunch of different models.

Can we find a way to choose the optimal label representations?

Finally, we tackled the problem of finding a way to select the label representation that optimizes the model's performances. To do so, we built an heuristic based on Attention attribution methods.

We experimented by injecting tokens to the training set sentences and then looking at which words from those sentences are used to build the representation of the tokens we injected. To do so, we relied on an Attention attribution method called Value-Zeroing.

After some tests, it seems that the best method for choosing the tokens was to look at which part of the sentences were used to build the representation of the End Of Sentence special token. By doing so, we were able to extract 100 representation sets. These were then ranked in descending order by their Value-Zeroing score, aggregating the score of representations that were selected multiple times in different sentences (to reward their higher frequency of selection).

We trained another hundred IT5 models using these representations and calculated their F-scores. We found a statistically significant negative correlation between the representation set rank and the model F-score, meaning that our method seems to correctly predict which representations are going to work best (Spearman Rank: -0.314).

We did some qualitative analysis on the words we used as representation to see if any interesting trend appeared, and one worth noting is that the model's class-specific F-score seemed correlated with the TF-IDF of the chosen words for two categories. This is in line with empiric observation of the fact that using Value-Zeroing for choosing representation seemed to yield mostly domain specific and words with a low global frequency, which often have a high TF-IDF.

Considerations and Future Work

The work described in this thesis exemplifies an interesting approach to deal with classification when using text-to-text models. In particular, we focused on one area of research, being Label Representation Selection, that has very few works published about. We believe that having text-to-text as a unifying framework to work with is ultimately beneficial, but comes with a price. Most of the generation models that work really well are billion parameter networks that are costly to train and to use, and aren't accessible to most of the population. To scale down this giant architectures in a meaningful way, we need to solve problems that those huge models can ignore thanks to their over-parametrization. There is, indeed, the need to identify and solve a number of different problems, with the aim of creating a truly unifying framework that is accessible both from a difficulty of use point-of-view, but also from a computational cost point-of-view. Right now, even the open source Large Language Models are hardly used directly in production, due to their hefty costs.

To continue this work, it would be enlightening to test this Label Representation Selection technique across different model architectures (e.g. auto-regressive ones, such as GPT-like models), across different classification tasks and on multiple languages. While we worked on classification and specifically Label Representation Selection, there is also the need to find techniques that solve problem linked to the other uses of the text-to-text framework, such as regression, generation, translation, etc.

In conclusion, this thesis provides the foundations to work towards smaller but better text-to-text language models capable of solving any kind of task they're trained for, providing simultaneously both the more intuitive text-to-text framework and the ability to be put to use in production, without the excessive computational costs required by the Largest Language Models popular today.

Bibliography

[Abnar and Zuidema, 2020] Abnar, S. and Zuidema, W. (2020). Quantifying attention flow in transformers. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4190–4197. Association for Computational Linguistics.

[Bahdanau et al., 2016] Bahdanau, D., Cho, K., and Bengio, Y. (2016). Neural machine translation by jointly learning to align and translate.

[Baker, 1975] Baker, J. (1975). The dragon system–an overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):24–29.

[Bansal et al., 2023] Bansal, H., Gopalakrishnan, K., Dingliwal, S., Bodapati, S., Kirchhoff, K., and Roth, D. (2023). Rethinking the role of scale for in-context learning: An interpretability-based case study at 66 billion scale. In *ACL 2023*.

[Baroni et al., 2009] Baroni, M., Bernardini, S., Ferraresi, A., and Zanchetta, E. (2009). The wacky wide web: a collection of very large linguistically processed web-crawled corpora. *Language resources and evaluation*, 43:209–226.

[Basile et al., 2020] Basile, V., Di Maro, M., Croce, D., and Passaro, L. (2020). Evalita 2020: Overview of the 7th evaluation campaign of natural language processing and speech tools for italian. In *7th Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop, EVALITA 2020*, volume 2765. CEUR-ws.

[Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554 – 1563.

[Bengio et al., 2000] Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

[Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., and Lin, H., editors, *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

[Cer et al., 2017] Cer, D., Diab, M., Agirre, E., Lopez-Gazpio, I., and Specia, L. (2017). Semeval-2017 task 1: Semantic textual similarity multilingual and crosslingual focused evaluation. In *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*. Association for Computational Linguistics.

[Chen et al., 2020] Chen, X., Xu, J., and Wang, A. (2020). Label representations in modeling classification as text generation. In *Proceedings of the 1st Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics and the 10th International Joint Conference on Natural Language Processing: Student Research Workshop*, pages 160–164.

[Chiu and Nichols, 2016] Chiu, J. P. and Nichols, E. (2016). Named Entity Recognition with Bidirectional LSTM-CNNs. *Transactions of the Association for Computational Linguistics*, 4:357–370.

[Chung et al., 2022] Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X., Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi,

E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. (2022). Scaling instruction-finetuned language models.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. In *NIPS 2014 Workshop on Deep Learning, December 2014*.

[Cimino et al., 2020] Cimino, Dell'Orletta, and Nissim (2020). Tag-it – topic, age and gender prediction. In *7th Evaluation Campaign of Natural Language Processing and Speech Tools for Italian. Final Workshop, EVALITA 2020*.

[Clark et al., 2019a] Clark, C., Lee, K., Chang, M.-W., Kwiatkowski, T., Collins, M., and Toutanova, K. (2019a). Boolq: Exploring the surprising difficulty of natural yes/no questions.

[Clark et al., 2019b] Clark, K., Khandelwal, U., Levy, O., and Manning, C. D. (2019b). What does BERT look at? an analysis of BERT's attention. In Linzen, T., Chrupała, G., Belinkov, Y., and Hupkes, D., editors, *Proceedings of the 2019 ACL Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 276–286, Florence, Italy. Association for Computational Linguistics.

[Dagan et al., 2006] Dagan, I., Glickman, O., and Magnini, B. (2006). The pascal recognising textual entailment challenge. In Quiñonero-Candela, J., Dagan, I., Magnini, B., and d'Alché Buc, F., editors, *Machine Learning Challenges. Evaluating Predictive Uncertainty, Visual Object Classification, and Recognising Tectual Entailment*, pages 177–190, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Devlin et al., 2019] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of deep bidirectional transformers for language understanding. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

[Dolan and Brockett, 2005] Dolan, W. B. and Brockett, C. (2005). Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the Third International Workshop on Paraphrasing (IWP2005)*.

[Gao et al., 2021] Gao, T., Fisch, A., and Chen, D. (2021). Making pre-trained language models better few-shot learners. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 3816–3830, Online. Association for Computational Linguistics.

[Gehring et al., 2017] Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. N. (2017). Convolutional sequence to sequence learning. In Precup, D. and Teh, Y. W., editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1243–1252. PMLR.

[Gladkova et al., 2016] Gladkova, A., Drozd, A., and Matsuoka, S. (2016). Analogy-based detection of morphological and semantic relations with word embeddings: what works and what doesn't. In Andreas, J., Choi, E., and Lazaridou, A., editors, *Proceedings of the NAACL Student Research Workshop*, pages 8–15, San Diego, California. Association for Computational Linguistics.

[Goodfellow et al., 2016] Goodfellow, I., Bengio, Y., and Courville, A. (2016). *Deep Learning*. MIT Press. http://www.deeplearningbook.org.

[Goodfellow et al., 2013] Goodfellow, I. J., Mirza, M., Da, X., Courville, A. C., and Bengio, Y. (2013). An empirical investigation of catastrophic forgeting in gradient-based neural networks. *CoRR*, abs/1312.6211.

[Graves et al., 2007] Graves, A., Liwicki, M., Bunke, H., Schmidhuber, J., and Fernández, S. (2007). Unconstrained on-line handwriting recognition with recurrent neural networks. In Platt, J., Koller, D., Singer, Y., and Roweis, S., editors, *Advances in Neural Information Processing Systems*, volume 20. Curran Associates, Inc.

[Graves et al., 2013] Graves, A., Mohamed, A.-r., and Hinton, G. (2013). Speech recognition with deep recurrent neural networks. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 6645–6649.

[Hahnloser et al., 2000] Hahnloser, R. H., Sarpeshkar, R., Mahowald, M. A., Douglas, R. J., and Seung, H. S. (2000). Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit. *Nature*, 405(6789):947–951.

[Hao et al., 2021] Hao, Y., Dong, L., Wei, F., and Xu, K. (2021). Self-attention attribution: Interpreting information interactions inside transformer.

[Hochreiter, 1998] Hochreiter, S. (1998). The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6:107–116.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9:1735–80.

[Iyer et al., 2017] Iyer, S., Dandekar, N., and Csernai, K. (2017). First quora dataset release: Question pairs.

[Jain and Wallace, 2019] Jain, S. and Wallace, B. C. (2019). Attention is not Explanation. In Burstein, J., Doran, C., and Solorio, T., editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 3543–3556, Minneapolis, Minnesota. Association for Computational Linguistics.

[Kalchbrenner and Blunsom, 2013] Kalchbrenner, N. and Blunsom, P. (2013). Recurrent continuous translation models. In Yarowsky, D., Baldwin, T., Korhonen, A., Livescu, K., and Bethard, S., editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1700–1709, Seattle, Washington, USA. Association for Computational Linguistics.

[Khashabi et al., 2018] Khashabi, D., Chaturvedi, S., Roth, M., Upadhyay, S., and Roth, D. (2018). Looking beyond the surface: A challenge set for reading comprehension

over multiple sentences. In Walker, M., Ji, H., and Stent, A., editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 252–262, New Orleans, Louisiana. Association for Computational Linguistics.

[Kobayashi et al., 2020] Kobayashi, G., Kuribayashi, T., Yokoi, S., and Inui, K. (2020). Attention is not only a weight: Analyzing transformers with vector norms. In *Conference on Empirical Methods in Natural Language Processing*.

[Kupiec, 1992] Kupiec, J. (1992). Robust part-of-speech tagging using a hidden markov model. *Computer Speech & Language*, 6(3):225–242.

[Lafferty et al., 2001a] Lafferty, J. D., McCallum, A., and Pereira, F. (2001a). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *International Conference on Machine Learning*.

[Lafferty et al., 2001b] Lafferty, J. D., McCallum, A., and Pereira, F. C. N. (2001b). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Lample and Conneau, 2019] Lample, G. and Conneau, A. (2019). Cross-lingual language model pretraining.

[Lan et al., 2020] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations.

[Levy and Goldberg, 2014] Levy, O. and Goldberg, Y. (2014). Linguistic regularities in sparse and explicit word representations. In Morante, R. and Yih, S. W.-t., editors, *Proceedings of the Eighteenth Conference on Computational Natural Language Learning*, pages 171–180, Ann Arbor, Michigan. Association for Computational Linguistics.

[Lewis et al., 2019] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

[Ling et al., 2015] Ling, W., Dyer, C., Black, A. W., Trancoso, I., Fermandez, R., Amir, S., Marujo, L., and Luís, T. (2015). Finding function in form: Compositional character models for open vocabulary word representation. In Màrquez, L., Callison-Burch, C., and Su, J., editors, *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 1520–1530, Lisbon, Portugal. Association for Computational Linguistics.

[Linzen, 2016] Linzen, T. (2016). Issues in evaluating semantic spaces using word analogies. In *Proceedings of the 1st Workshop on Evaluating Vector-Space Representations for NLP*, pages 13–18, Berlin, Germany. Association for Computational Linguistics.

[Liu et al., 2021] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., and Neubig, G. (2021). Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *arXiv preprint arXiv:2107.13586*.

[Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach.

[Marie-Catherine de Marneffe, 2019] Marie-Catherine de Marneffe, Mandy Simons, J. T. (2019). The commitmentbank: Investigating projection in naturally occurring discourse.

[Markov, 1913] Markov (1913). Essai d'une recherche statistique sur le texte du roman "Eugene Onegin" illustrant la liaison des epreuve en chain ('Example of a statistical investigation of the text of "Eugene Onegin" illustrating the dependence between samples in chain'). *Izvistia Imperatorskoi Akademii Nauk (Bulletin de l'Académie Impériale des Sciences de St.-Pétersbourg)*, 7:153–162. English translation by Morris Halle, 1956.

[Maslennikova et al., 2019] Maslennikova, A., Labruna, P., Cimino, A., and Dell'Orletta, F. (2019). Quanti anni hai? age identification for italian. In *CLiC-it*.

[Maximilian, 2023] Maximilian, S. (2023). Gpt-4 architecture, datasets, costs and more leaked. *THE DECODER*.

[McCulloch and Pitts, 1943] McCulloch, W. S. and Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, 5(4):115–133.

[Mikolov et al., 2013a] Mikolov, T., Chen, K., Corrado, G. S., and Dean, J. (2013a). Efficient estimation of word representations in vector space. In *International Conference on Learning Representations*.

[Mikolov et al., 2013b] Mikolov, T., Yih, W.-t., and Zweig, G. (2013b). Linguistic regularities in continuous space word representations. In Vanderwende, L., Daumé III, H., and Kirchhoff, K., editors, *Proceedings of the 2013 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 746–751, Atlanta, Georgia. Association for Computational Linguistics.

[Minsky and Papert, 1969] Minsky, M. and Papert, S. (1969). Perceptrons.

[Mohebbi et al., 2023] Mohebbi, H., Zuidema, W., Chrupała, G., and Alishahi, A. (2023). Quantifying context mixing in transformers. In *EACL*.

[Nagoudi et al., 2022] Nagoudi, E. M. B., Elmadany, A., and Abdul-Mageed, M. (2022). Arat5: Text-to-text transformers for arabic language generation.

[Occhipinti et al., 2020] Occhipinti, D., Tesei, A., Iacono, M., Aliprandi, C., De Mattei, L., and AI, A. (2020). Italianlp@ tag-it: Umberto for author profiling at tag-it 2020.

[OpenAI et al., 2023] OpenAI, Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O.,

Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H. W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S. P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross, J., Gu, S. S., Guo, Y., Hallacy, C., Han, J., Harris, J., He, Y., Heaton, M., Heidecke, J., Hesse, C., Hickey, A., Hickey, W., Hoeschele, P., Houghton, B., Hsu, K., Hu, S., Hu, X., Huizinga, J., Jain, S., Jain, S., Jang, J., Jiang, A., Jiang, R., Jin, H., Jin, D., Jomoto, S., Jonn, B., Jun, H., Kaftan, T., Łukasz Kaiser, Kamali, A., Kanitscheider, I., Keskar, N. S., Khan, T., Kilpatrick, L., Kim, J. W., Kim, C., Kim, Y., Kirchner, H., Kiros, J., Knight, M., Kokotajlo, D., Łukasz Kondraciuk, Kondrich, A., Konstantinidis, A., Kosic, K., Krueger, G., Kuo, V., Lampe, M., Lan, I., Lee, T., Leike, J., Leung, J., Levy, D., Li, C. M., Lim, R., Lin, M., Lin, S., Litwin, M., Lopez, T., Lowe, R., Lue, P., Makanju, A., Malfacini, K., Manning, S., Markov, T., Markovski, Y., Martin, B., Mayer, K., Mayne, A., McGrew, B., McKinney, S. M., McLeavey, C., McMillan, P., McNeil, J., Medina, D., Mehta, A., Menick, J., Metz, L., Mishchenko, A., Mishkin, P., Monaco, V., Morikawa, E., Mossing, D., Mu, T., Murati, M., Murk, O., Mély, D., Nair, A., Nakano, R., Nayak, R., Neelakantan, A., Ngo, R., Noh, H., Ouyang, L., O'Keefe, C., Pachocki, J., Paino, A., Palermo, J., Pantuliano, A., Parascandolo, G., Parish, J., Parparita, E., Passos, A., Pavlov, M., Peng, A., Perelman, A., de Avila Belbute Peres, F., Petrov, M., de Oliveira Pinto, H. P., Michael, Pokorny, Pokrass, M., Pong, V., Powell, T., Power, A., Power, B., Proehl, E., Puri, R., Radford, A., Rae, J., Ramesh, A., Raymond, C., Real, F., Rimbach, K., Ross, C., Rotsted, B., Roussez, H., Ryder, N., Saltarelli, M., Sanders, T., Santurkar, S., Sastry, G., Schmidt, H., Schnurr, D., Schulman, J., Selsam, D., Sheppard, K., Sherbakov, T., Shieh, J., Shoker, S., Shyam, P., Sidor, S., Sigler, E., Simens, M., Sitkin, J., Slama, K., Sohl, I., Sokolowsky, B., Song, Y., Staudacher, N., Such, F. P., Summers, N., Sutskever, I., Tang, J., Tezak, N., Thompson, M., Tillet, P., Tootoonchian, A., Tseng, E., Tuggle,

P., Turley, N., Tworek, J., Uribe, J. F. C., Vallone, A., Vijayvergiya, A., Voss, C., Wainwright, C., Wang, J. J., Wang, A., Wang, B., Ward, J., Wei, J., Weinmann, C., Welihinda, A., Welinder, P., Weng, J., Weng, L., Wiethoff, M., Willner, D., Winter, C., Wolrich, S., Wong, H., Workman, L., Wu, S., Wu, J., Wu, M., Xiao, K., Xu, T., Yoo, S., Yu, K., Yuan, Q., Zaremba, W., Zellers, R., Zhang, C., Zhang, M., Zhao, S., Zheng, T., Zhuang, J., Zhuk, W., and Zoph, B. (2023). Gpt-4 technical report.

[Papucci et al., 2023] Papucci, M., Miaschi, A., and Dell'Orletta, F. (2023). Lost in labels: An ongoing quest to optimize text-to-text label selection for classification. In *Proceedings of the 9th Italian Conference on Computational Linguistics (CLiC-it 2023)*.

[Papucci et al., 2022] Papucci, M., Nigris, C. D., Miaschi, A., and Dell'Orletta, F. (2022). Evaluating text-to-text framework for topic and style classification of italian texts. In *NL4AI@AI*IA*.

[Parisi et al., 2020] Parisi, L., Francia, S., and Magnani, P. (2020). Umberto: an italian language model trained with whole word masking. https://github.com/musixmatchresearch/umberto.

[Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In Moschitti, A., Pang, B., and Daelemans, W., editors, *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[Pilehvar and Camacho-Collados, 2019] Pilehvar, M. T. and Camacho-Collados, J. (2019). Wic: the word-in-context dataset for evaluating context-sensitive meaning representations.

[Radford et al., 2018] Radford, A., Narasimhan, K., Salimans, T., Sutskever, I., et al. (2018). Improving language understanding by generative pre-training. *OpenAI blog*.

[Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., Sutskever, I., et al. (2019). Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9.

[Raffel et al., 2019] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research (JMLR)*, 21(140):1–67.

[Rajpurkar et al., 2016] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). Squad: 100,000+ questions for machine comprehension of text.

[Roemmele et al., 2011] Roemmele, M., Bejan, C., and Gordon, A. (2011). Choice of plausible alternatives: An evaluation of commonsense causal reasoning. In *AAAI Spring Symposium - Technical Report*.

[Rumelhart and Abrahamson, 1973] Rumelhart, D. E. and Abrahamson, A. A. (1973). A model for analogical reasoning. *Cognitive Psychology*, 5(1):1–28.

[Rumelhart et al., 1985a] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1985a). Learning internal representations by error propagation.

[Rumelhart et al., 1985b] Rumelhart, D. E., Hinton, G. E., Williams, R. J., et al. (1985b). Learning internal representations by error propagation.

[Sarti and Nissim, 2022] Sarti, G. and Nissim, M. (2022). It5: Large-scale text-to-text pretraining for italian language understanding and generation. *ArXiv preprint 2203.03759*.

[Schick and Schütze, 2020] Schick, T. and Schütze, H. (2020). Few-shot text generation with pattern-exploiting training. *arXiv preprint arXiv:2012.11926*.

[Schick and Schütze, 2021] Schick, T. and Schütze, H. (2021). Exploiting cloze-questions for few-shot text classification and natural language inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Compu-*

tational Linguistics: Main Volume, pages 255–269, Online. Association for Computational Linguistics.

[Schluter, 2018] Schluter, N. (2018). The word analogy testing caveat. In Walker, M., Ji, H., and Stent, A., editors, *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*, pages 242–246, New Orleans, Louisiana. Association for Computational Linguistics.

[Schuster and Paliwal, 1997] Schuster, M. and Paliwal, K. (1997). Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681.

[Settles, 2004] Settles, B. (2004). Biomedical named entity recognition using conditional random fields and rich feature sets. In Collier, N., Ruch, P., and Nazarenko, A., editors, *Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications (NLPBA/BioNLP)*, pages 107–110, Geneva, Switzerland. COLING.

[Sha and Pereira, 2003] Sha, F. and Pereira, F. C. (2003). Shallow parsing with conditional random fields. In *North American Chapter of the Association for Computational Linguistics*.

[Socher et al., 2013] Socher, R., Perelygin, A., Wu, J., Chuang, J., Manning, C. D., Ng, A., and Potts, C. (2013). Recursive deep models for semantic compositionality over a sentiment treebank. In Yarowsky, D., Baldwin, T., Korhonen, A., Livescu, K., and Bethard, S., editors, *Proceedings of the 2013 Conference on Empirical Methods in Natural Language Processing*, pages 1631–1642, Seattle, Washington, USA. Association for Computational Linguistics.

[Tiedemann, 2016] Tiedemann, J. (2016). OPUS – parallel corpora for everyone. In *Proceedings of the 19th Annual Conference of the European Association for Machine Translation: Projects/Products*, Riga, Latvia. Baltic Journal of Modern Computing.

[Touvron et al., 2023] Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). Llama: Open and efficient foundation language models.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. *Advances in neural information processing systems*, 30.

[Wang et al., 2019] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2019). Glue: A multi-task benchmark and analysis platform for natural language understanding.

[Warstadt et al., 2019] Warstadt, A., Singh, A., and Bowman, S. R. (2019). Neural network acceptability judgments.

[Werbos, 1990] Werbos, P. (1990). Backpropagation through time: what it does and how to do it. *Proceedings of the IEEE*, 78(10):1550–1560.

[Werbos and John, 1974] Werbos, P. and John, P. (1974). *Beyond regression: new tools for prediction and analysis in the behavioral sciences*. Harvard University.

[Wiegreffe and Pinter, 2019] Wiegreffe, S. and Pinter, Y. (2019). Attention is not not explanation. In Inui, K., Jiang, J., Ng, V., and Wan, X., 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 11–20, Hong Kong, China. Association for Computational Linguistics.

[Williams et al., 2018] Williams, A., Nangia, N., and Bowman, S. R. (2018). A broad-coverage challenge corpus for sentence understanding through inference.

[Xue et al., 2022] Xue, L., Barua, A., Constant, N., Al-Rfou, R., Narang, S., Kale, M., Roberts, A., and Raffel, C. (2022). Byt5: Towards a token-free future with pre-trained byte-to-byte models.

[Xue et al., 2021] Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., and Raffel, C. (2021). mt5: A massively multilingual pre-trained text-to-text transformer.

[Zhang et al., 2018] Zhang, S., Liu, X., Liu, J., Gao, J., Duh, K., and Durme, B. V. (2018). Record: Bridging the gap between human and machine commonsense reading comprehension.

[Zhou and Xu, 2015] Zhou, J. and Xu, W. (2015). End-to-end learning of semantic role labeling using recurrent neural networks. In Zong, C. and Strube, M., editors, *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)*, pages 1127–1137, Beijing, China. Association for Computational Linguistics.

[Zhu et al., 2015] Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books.

Acknowledgments

First of all, I would like to thank my supervisor and mentor Felice Dell'Orletta that supported me during all my Master's degree, always giving me great opportunities and the tools I needed to succeed. I will never forget that and I hope to continue to work with him in the future.

I would also like to thank all the ItaliaNLP lab from the Institute of Computational Linguistics of the CNR, that two years ago welcomed me between them: Alessio, Andrea A., Chiara A., Chiara F., Dominique, Felice, Giulia, Luca and Marta. Thanks for being more friends than colleagues.

My thanks also goes to GruppoMeta, and in particular Daniele and Paolo, for giving me the chance to finish my Master's and for never saying no whenever I needed something to do so.

I need to thank my family for supporting me in my academic journey and in the choices I've made so far.

I want to thank Sofia, which has always been by my side, and was always available to support and comfort me when I needed it. I want to thank her for believing in what I do, and for supporting the choices I've made and the ones I am making. I hope she'll be able to say the same about me.

I need to thank Gabriele D.Z., for being the best friend one could've asked for, and I also want to thank the rest of my group of friends: Alessandro, Federico, Francesco, Gabriele G., Lorenzo, Marco and Matteo.

I also want to thank my Master's colleagues, and now friends, Chiara D.N., Chiara G., Emanuele and Simona, for all the time we spent together studying, following lectures, and preparing exams: you've made reaching this achievement much easier. I feel obliged to thank again Chiara D.N.: she was the only person I knew when I started my Master's and, without her notes and her help, I would've never reached this goal this fast.

Thanks again to Sofia, Chiara D.N., Chiara F. and Felice for reading and proofreading my thesis.