University of Padua
School of Engineering
Department of Information Engineering

Master's Degree in Computer Engineering
curriculum Artificial Intelligence and Robotics
Master Thesis

Leveraging Recursive Neural Networks on Dependency Trees for Online-Toxicity Detection on Twitter

Candidate:
Nicolò Penzo
University of Padua

Supervisors:
prof. Giorgio Satta
prof. Fabio Vandin
University of Padua

Co-supervisors:
prof. Dirk Hovy
Federico Bianchi, PhD
Bocconi University

Academic Year 2021-2022
July 14^{th}, 2022

Contents

Abstract

Current social dynamics are strongly linked to what happens on Social Media. Opinions, emotions, and how people perceive the world around them are strongly influenced by what they see or read on Social Platforms. We can insert in this field Social Media phenomena like Fake News, Hate Speech, Propaganda, Race and Gender biases. All these events are considered to be among the most significant problems for social stability and one of the most effective means of influencing people. Much work has been done by researchers from different areas of Computer Science, in particular from Natural Language Processing and Network Analysis, focusing on textual information in the first case (articles, posts, comments, etc.) or graph structures and node activities in the second (detection of malicious spreaders, polarization, etc.). In this thesis, we will clarify what are the main problems in this area of research, known by most as Computational Social Science, providing the theoretical basis of the most used tools. Then, we will go into specifics dealing with the topic of the detection of toxic messages on Twitter at the level of the single tweet, comparing different Deep Learning models, among which some innovative solutions proposed by us, trying to answer the following question: can Natural Language syntax be useful in such task? Unlike, for instance, Sentiment Analysis, we have not yet achieved high performance, especially because the models typically used, given a sentence, turn out to focus a lot on the occurring words rather than on the meaning of the sentence itself. Our idea starts from the assumption that exploiting syntactic information can be effective to overcome this obstacle. In the end, we will provide the results of our experiments and possible related interpretations, proposing scientific and ethical reflections, and finally try to convince the reader on why research should invest efforts on this topic, and what future scenarios we should focus on.

Keywords: Dependency trees, toxic language, Recursive Neural Networks.

Sommario

Le attuali dinamiche sociali sono strettamente legate a ciò che accade nei Social Media. Opinioni, emozioni, ed il modo in cui le persone percepiscono il mondo attorno a loro sono fortemente influenzati da ciò che vedono o leggono nelle piattaforme Social. Possiamo inserire in questo campo fenomeni dei Social Media come: Fake News, contenuti d'odio, propaganda, bias di razza e genere. Tutti questi eventi sono considerati tra i maggiori problemi per la stabilità sociale e uno dei mezzi più efficaci per influenzare le persone. Molto lavoro è stato svolto dai ricercatori di diverse aree delle Scienze Informatiche, in particolare dal Natural Language Processing e Network Analysis, focalizzandosi sull'informazione testuale nel primo caso (articoli, post, commenti ecc.) o su strutture a grafo e attività dei nodi nel secondo (rilevamento di propagatori nocivi, polarizzazione ecc.). In questa tesi, andremo a chiarire quali sono i problemi principali in quest'area di ricerca, conosciuta dai più come Scienze Sociali Computazionali, fornendo le basi teoriche degli strumenti più utilizzati. Dopodiché andremo nello specifico trattando il tema del rilevamento di messaggi tossici in Twitter a livello del singolo tweet, confrontando diversi modelli di Deep Learning, tra i quali alcuni totalmente innovativi proposti da noi, tentando di rispondere alla seguente domanda: la sintassi del linguaggio naturale può essere utile in questo task? A differenza, per esempio, del sentiment analysis, non abbiamo ancora raggiunto performance elevate, soprattutto perché i modelli utilizzati solitamente, data una frase, risultano focalizzarsi molto sulle parole presenti più che al senso della frase stessa. La nostra idea parte dal presupposto che per superare questo scoglio, sfruttare l'informazione sintattica può risultare efficace. Alla fine forniremo i risultati dei nostri esperimenti e le possibili relative interpretazioni, proponendo riflessioni di carattere scientifico e etico, provando infine a convincere il lettore su come mai la ricerca dovrebbe investire sforzi in questo tema, e quali potrebbero essere gli scenari futuri su cui puntare.

Parole chiave: Dependency trees, toxic language, Recursive Neural Networks.

Acknowledgments

Una tesi non convenzionale merita ringraziamenti non convenzionali. In questa avventura ho avuto modo di fare un vero e proprio viaggio all'interno della comunità scientifica e delle sue dinamiche. Sono molte le persone che mi sento di dover ringraziare, del mondo accademico e non, che sono state fondamentali per una moltitudine di motivi. Sono profondamente convinto che ciò che si esprime con la propria testa è in larga parte frutto di chi ci sta attorno e delle esperienze vissute con gli altri.

Un gigantesco ringraziamento va a Giorgio Satta. È stata la prima persona a cui ho esposto le mie idee e passioni, e in lui ho sempre trovato aiuto, supporto, e consigli pronti per il lavoro di tesi, per la carriera futura, e un mondo di aneddoti ed esempi. Non lo potrò mai ringraziare abbastanza per quanto abbia fatto in quest'ultimo anno, per avermi fatto appassionare al campo del NLP, stimolato a cercare sempre di meglio e messo continuamente in crisi con dubbi e domande inaspettate che hanno sicuramente influenzato il mio modo di pensare e procedere.

Un ringraziamento speciale va anche a Fabio Vandin. Seppur la direzione della tesi abbia subito un'importante deviazione rispetto a quanto pensato all'inizio, allontanandosi magari dai suoi campi di ricerca, si è sempre interessato allo svolgimento ed è sempre stato di aiuto anche in ottica di carriera futura, dandomi l'opportunità di conoscere il suo gruppo di ricerca. Grazie per tutto questo.

Anche a Dirk Hovy va un sentitissimo ringraziamento. Da una semplice mail di uno studente sconosciuto e con esperienza quasi nulla ne è nata una collaborazione inaspettata ed estremamente fruttuosa. Lo ringrazio per la fiducia, per avermi permesso di conoscere la sua realtà MilaNLP, per le chiacchierate sempre stimolanti e per essere stato di aiuto fondamentale in un momento di stallo con la tesi.

Dulcis in fundo, un ringraziamento di cuore va a Federico Bianchi. Lo ringrazio per essere sempre stato disponibile per qualsiasi dubbio, per i consigli e la sua esperienza, e soprattutto per tutto il tempo perso a sentire le mie perplessità e le mie insicurezze su codici, esperimenti e interpretazioni, ed aver alleggerito la pressione. E, sopra ogni altra cosa, per avermi trasmesso l'amore e l'ammirazione verso i Capibara! Buona fortuna per la nuova avventura.

Mi sento in dovere di ringraziare per i loro feedback e le chiacchierate interessanti Giovanni Da San Martino (Università degli Studi di Padova), Fabio Massimo Zanzotto (Università di Roma-Tor Vergata) e Tommaso Green (Università di Mannheim). A quest'ultimo un ringraziamento speciale per avermi guidato all'interno della comunità NLP, con consigli sempre utili per il futuro, e senza il quale probabilmente la collaborazione con Dirk Hovy e Federico Bianchi non sarebbe venuta alla luce.

Colgo l'occasione per ringraziare anche Andrea Pietracaprina, mio relatore per la Tesi di Laurea Triennale, e prima persona in assoluto a correggere un testo scientifico scritto da me. I suoi suggerimenti anche dopo la tesi sono stati fondamentali, grazie mille di tutto.

Adesso arriva il momento di ringraziare chi c'è stato sempre, nell'ambiente universitario e non. Ognuno meriterebbe di essere nominato per nome e cognome, ma la lista diventerebbe estremamente lunga. Mi limito a descrivervi, e credo che saprete tranquillamente dove andare a collocarvi.

Un grande abbraccio va ai ragazzi del cosiddetto "Unigroup" (più chiaro di così). Ci siamo conosciuti tutti quanti per circostanze estremamente casuali, ma ne è nato un gruppo coeso di amici, un appoggio per l'università ma anche per lo svago. I successi che abbiamo raggiunto tutti quanti sono sintomo di quanto questo gruppo sia stato una forza fondamentale. Vi auguro il meglio per il futuro e i vostri progetti.

Un calorosissimo abbraccio va ai ragazzi che conobbi 20 anni fa e con cui tutt'oggi trascorro giornate e serate. Grazie per il divertimento, le pazzie, le esperienze vissute, gli aperitivi, i pranzi, le cene e le improvvisazioni. Non tutti hanno la fortuna di fare affidamento letteralmente sugli "amici di una vita".

Piccola nota (solo perché siete in pochi) anche a quei "soggetti" che ho conosciuto al liceo in quel di Schio e che tutt'oggi sono rimasti preziosi. Grazie per essere sempre delle spalle su cui contare.

Un ringraziamento va anche agli amici più vecchi (in senso di età), a tutta la mia (larghissima) famiglia per le esperienze vissute soprattutto quand'ero più giovane, a quelli che mi hanno tirato pugni in faccia (ovviamente per sport) e a quanti mi hanno strappato almeno una risata in vita loro. Un ringraziamento speciale va anche a te che, in fin dei conti e nonostante tutto, ci sei stata nei momenti più duri.

Ultimo ringraziamento va alla mia famiglia più stretta, gli unici che nominerò per nome. Ringrazio i miei genitori, Antonella ed Elvis, per avermi permesso di seguire le mie passioni e i miei sogni, senza chiedere nulla in cambio e fidandosi delle mie idee e dei miei progetti (a volte un po' troppo), per cui spero di aver soddisfatto almeno parte di quella fiducia, e anche Daniele (che altrimenti si offende), sperando che il mio percorso possa essere un po' da guida per il suo futuro.

Grazie di tutto, a tutti.

Nicolò

DISCLAIMER

This Thesis contains slur, offensive and abusive language, due to the topics discussed and the experiments performed. To present the results, this cannot be avoided. However, we have to assure the reader that every possible offensive sentence is just a matter of research, and there is no malicious intent.

Introduction

Social Media platforms have moved from being simple and intuitive means of communication to being real "social squares", where people can chat with friends and strangers, share pieces of personal life, sell products and even their own image, look for a job, keep up to date with what is happening in the world, watch and participate to a kind of impromptu talk show, publicize political campaigns and many more activities, all in the same virtual environment.

Nowadays, it is difficult to find someone who does not use at least one Social Network like Facebook, Twitter, Instagram, TikTok, LinkedIn, Reddit, Youtube, etc. Not using them has become almost impossible, since even institutions use their official social pages to publish opportunities (for work or other reasons) and send updates or clarifications (for instance during the COVID19 pandemic, with instant lockdowns and health measures). Because of all these possible uses, people exploit their own social personal page as a window (accurately managed) to make new friends, spam projects or attract possible hiring companies in the work context. Also finding the complete list of social contacts in a CV is standard nowadays[1].

Now that the reader has seen the centrality of Social Networks in everyday life, it is time to show the side effects. As we have reported, anyone can post anything for any reason. This includes insults, slurs, harassment, disinformation, hate speech, etc., with racist, misogynist, homophobic goals, and many others. All these phenomena have no clear bounds (sometimes they overlap, sometimes not), so in recent years it is typical to summarize all these terms under the name of **online-toxicity behavior**. Only for what regards Fake News sometimes we have a clear distinction, where given a claim we want to state if it is true or not, not looking at the style, etc. However, typically disinformative content is delivered in a "provocative" style and therefore is included in the toxic set. It is also common to write the term "hate speech" meaning the toxic definition. The community of Artificial Intelligence (or more generally Computer Science) today finds itself constantly facing these problems, since a purely handcrafted treatment is impossible, due to the enormous amount of data to work on and also for the subjectivity of the human operators. From Natural Language Processing (NLP) techniques to detect toxic posts to Computer Vision (CV) tools to find violent images/video, passing through Network Analysis to retrieve and analyze communities and/or content spreading. The

[1] https://www.kickresume.com/en/help-center/how-include-your-social-media-resume/

relevance of this topic and its importance are proven by the exponential growth of publications in this field (Tontodimamma et al. [2021]), which follows the increase of hate crimes and online hate speech (see Chaudhary et al. [2021]).

This dissertation is the result of an extremely broad exploration of current research regarding Fake News and Toxic Messages, from the perspective of NLP and Network Analysis. One major difficulty of working in this field from scratch is the speed with which new articles are continually published, presenting new results each time and proposing possible changes in the scenario. After a careful and extensive review of the latest publications, we have decided to focus on a basic (but fundamental) problem: **toxicity detection** on tweets.

This is due to two fundamental reasons: (i) the detection of toxic or hate speech messages on individual posts/tweets is still a difficult problem (unlike Sentiment Analysis) and people take it for granted that Transformer-based models or Foundation Models, in general, are the only possible solution, with a small preprocessing step and "letting the model learn everything"; we propose a set of experiments to see if smaller models but more structure-aware and explainable can have similar results, in order to propose a new research direction; (ii) we are convinced that focusing on the single message is weak in a practical scenario, and we believe that an effective task can be the forecasting of conversational derailment towards toxic contents, but to do this we need good models capable to capture useful textual information. This is why we have focused on this basic but fundamental step for future research.

In Chapter 1 we will present the context in which the work develops, the research field that deals with this environment, and a general introduction to online-toxicity, providing also the formal definition for Toxicity Detection task. We then move on to Chapter 2 where we report most of the literature on which our work is based and also some other solutions proposed in the same context, but without a direct connection with the main topic of the thesis, just to show the reader the great variety of possible tasks in these problems. In Chapter 3 we will provide definitions and terminologies used on Twitter, also explaining some social mechanisms, and we will extensively present the datasets used in our experiments. Subsequently, in Chapter 4 we present the historical "sequential" models (BiLSTM and BERT) that we use as a baseline or for other comparisons, while in Chapter 5 we present our new model that exploits dependency trees to perform toxicity classification, called "Syntactic Encoder". Chapter 6 is dedicated to describing our experimental setup

and results, discussed later in Chapter 7, with some ethical considerations. Finally, we summarize the thesis and share our conclusions in Chapter 8.

Contributions

We can summarize our research contributions in three main points:

- We propose a novel Recursive Neural Network (RecNN) that processes trees both top-down and bottom-up, in a combined way. This can truly be seen as a generalization to trees of traditional BiLSTM which process strings bidirectionally. In addition, RecNN processes trees with unbounded node degrees and in a way that accounts for children ordering at each node. This is in contrast to standard approaches in the literature, where children are processed as bags, thereby dropping order information.

- We apply RecNN to so-called dependency trees, a popular syntactic representation for natural language. This allows us to experimentally investigate the effectiveness of syntax in toxicity detection on tweets. To the best of our knowledge, this is the first time that dependency trees are exploited in this task. We believe this draws an important direction for future research since in online-toxicity we cannot rely on "trigger" words only. Taking into account syntactic structure allows us to extract complex linguistic aspects, such as rhetoric, style of writing, scope of negation, writer demographics, etc.

- Finally, we propose a novel test typology to verify to which extent a syntactic-aware model makes effective use of syntactic features. This is based on the idea of training the model on gold or silver syntactic trees, and then using randomly constructed trees at test time. To the best of our knowledge, this has not been considered so far in the specialized literature, even for tasks other than toxicity detection that use syntactic representations, for instance in sentiment analysis. We believe this is a useful methodology to spot cases in which the model relies on trigger words only, even though it has been exposed at training time to syntactic features. This can in turn happen for specific datasets in which syntactic features are not of main relevance, such as texts with very short and straight sentences.

Chapter 1

Research Landscape

In this first Chapter, we will provide some ideas about the subject of the study (Social Media and Toxic Language) and a brief introduction to a research field that is becoming prominent in recent years: Computational Social Science. First of all, we give the reader an idea of how Social Networks are pervasive in everyday life. Then, we will report an overview of how computational methods can be exploited to analyze Social Media. Finally, we will present the main topic of this thesis: the detection of toxic tweets.

1.1 Social Networks: statistics

The term "Social Networks" usually refers to "Social Media Platforms", online applications typically used to share content (e.g., texts, images, videos, etc.) with other people, from real-life friends to mere followers. This allows users to keep in touch with each other easily and eventually to form a sort of personal social circle, where discussions or content reflect their passions and preferences. Nowadays, such platforms have become almost indispensable for many peoples, and just to name a few: Facebook, Instagram, Twitter, Reddit, LinkedIn, more recently TikTok, etc.

According to www.datareportal.com[1], in January 2022 there were 4.62 billion users on Social Media, equal to 58.4% of the total world population. Out of all the statistics reported in this article, we need to focus on three main points:

[1] https://datareportal.com/social-media-users

- taking into account possible duplicates or restrictions for the subscription to the platform, the users around are equivalent to almost 75% of the eligible global population;
- users actively use or visit an average of 7.5 different social platforms each month;
- among the main reasons for using social media we have: staying in touch with friends; filling free time; reading news stories; finding content; seeing what is being talked about; finding inspiration for things to do and buy; finding products to buy; sharing and discussion of opinions; establish new contacts; watch live streams; work-related networking and research; find like-minded communities and interest groups; follow celebrities or influencers; post about your life; avoiding losing things.

The large number of users and the actions carried out on the Social Platforms make every possible "moderation" of the contents really challenging, both from a technical and ethical point of view. Another observation is that all the activities reported in the third point are carried out in the same platform/environment. This causes a lot of confusion and chaos among users: news "sold" as stories to read, opinions shared at first thought, unverified claims, and debates where the stronger rant wins, not the one who proves that its positions are the better.

1.2 Computational Social Science and Social Media

Before talking about Toxic Language, we need to talk about the intersection between Computer Science (for quantitative analyses) and Social Science (to qualitatively describe social phenomena): Computational Social Science (CSS for short). This is a huge field in which researchers develop computational methods for working on complex human behavioral data (from Lazer et al. [2020]), from spatial data to social networks and human coding of text and images. Especially with the advent of Big Data and Deep Learning, this field has gained a lot of interest, and many general "computational fields" nowadays are exploited extensively in Social contexts, especially Natural Language Processing, Network Analysis, and Complex Systems.

Still from Lazer et al. [2020], "*whereas traditional quantitative social science has focused on rows of cases and columns of variables, typically with assumptions of independence among observations, CSS encompasses language, location and*

movement, networks, images, and video, with the application of statistical models that capture multifarious dependencies within data".

In Social Media, there are some "topical" activities where CSS researchers are involved. We can look at a single post/message, with different possible types of classification, from sentiment analysis to detection of Fake News, Hate Speech, or Toxic posts in general, but also topic detection (what the post is about), and message generation (more advanced and ethically problematic).

Looking at the networks, there are other possible analyses, such as measuring community polarization, bot detection, and echo chamber detection[2].

All of these perspectives have found a challenging in combining messages and network information into more advanced tasks. An example is conversation analysis, where we do not only analyze the single message but a combination of them, also trying to grasp the "social context" in which a text appears and how these phenomena evolve over time.

This variety of tasks (and their combination) requires advanced models and tools to analyze large amounts of data, stimulating researchers to find new computational solutions applicable to large-scale data, which are sometimes also useful for other types of analysis in other research fields.

1.3 Toxicity Detection on Tweets

Finding an official definition of online-toxicity is not trivial. The literature uses the adjective "toxic" sometimes interchangeably with "hate speech" (like in Zhou et al. [2021]) or as "abusive" language (like in Pavlopoulos et al. [2017]). There is no absolute alignment of researchers around the meaning of the term "toxic language".

In our work we consider "toxic" all comments that can lead to intolerance, incivility, or antisocial behaviors, similar to Chang and Danescu-Niculescu-Mizil [2019], such as hate speech, disinformation, harassment, insults towards target protected groups, etc. However, there are also several problems in creating datasets for these events, which prove to be *biased* depending on the personal characteristics of the annotators (as mentioned in Zhou et al. [2021]). Another problematic aspect of this task is that models easily focus on "toxic" keywords instead of generalizing into

[2]We call echo chambers particular communities where opinions, political leaning, or beliefs of users on a topic get reinforced due to repeated interactions with peers or sources having similar tendencies and attitudes, from Cinelli et al. [2021]

something like sentence understanding, and this backfires on minorities (as reported by Sap et al. [2021]).

The idea behind using this super-definition to group a set of social phenomena is not counterproductive. The goal of toxicity detection is to find content that can lead to polarization of communities, make people hate each other because of race or gender, and prevent political figures from creating a sort of "virtual army" with attacks sometimes even coordinated by supporters towards enemies. All of these phenomena have common ground and lead to similar effects on users, so we believe finding a common solution is an effective path.

The only distinction that we need to point out is between toxicity detection and Fake News detection. The first, in fact, is based only on linguistics and on the single tweet, while Fake News detection wants to verify the veracity of a sentence, so there is also work on Information Retrieval. The sentence can only report incorrect information without using abusive expressions. For our part, when we talk about disinformation in the context of toxicity, we mean all the messages regarding false news but also with an attempted attack (against individuals or groups). This is similar to the Early Fake News Detection framework, where there is not enough data to directly verify the veracity of a claim and we can exploit only textual/network information, without retrieval work.

Given this brief introduction to the problem, we can provide a more formal definition of the task. We will perform a "simple" (but not trivial) binary classification task. Given a tweet t from a set T, our goal is to classify it with a label between 0 (non-toxic) and 1 (toxic). More formally:

Definition 1. ***Toxicity detection****: assuming we have the universe $\mathcal{T}$ of all possible tweets and a set of tweets T, toxicity detection is a binary classification task in which, given a tweet $t \in \mathcal{T}$, we have to predict if t is toxic (label 1) or non-toxic (label 0) through a learnable function $P_T : \mathcal{T} \to \{0, 1\}$, where P_T is learned from the set of tweets T.*

Chapter 2

Related Work

Research into online-toxicity detection and similar phenomena, such as Fake News detection, community polarization analysis, and echo chamber detection, has been extensively explored in recent years. As we said in Section 1.2, several methods can be applied to solve these problems. In this Chapter, we provide the reader with a broad overview of significant works that inspired our research, to give an idea of which computational methodologies are exploited in this field, what is the current state of the art, and where the main problems are located.

2.1 Detection and forecasting of online-toxicity

Online toxicity detection and forecasting are closely related, even though the second is much more challenging (and has higher expectations of effectiveness in a real application). By detection, we mean finding the target event *after* it has occurred. Instead, forecasting means predicting its occurrence *before* it happens.

The detection of a single toxic message is critical to leverage the same tweet modeling primarily to detect toxic conversation and, finally, forecast the emergence of toxic content in conversations. As we will see in Section 2.1.1, current research is focusing on models that tend to be biased on "toxic" keywords, but if we want to shift our focus to forecasting, we need more general models, which capture something more than "trigger" words.

2.1.1 Tweet level

By detecting toxicity at the tweet level, we usually mean models mainly focused on the style and the content of a single message, possibly enriched with user meta-data (username, provenience, profile description, age, etc.) and network information (we will see some methodologies in Section 2.1.3).

Toxicity detection can be identified as a case of text (or sentence) classification. Therefore, successful models in NLP for this task have been based on LSTM or BiLSTM, although these are now considered baselines. With the birth of Large Language Models, as expected, researchers have profoundly exploited models such as BERT (from Devlin et al. [2019], we will present it in detail in Section 4.2) and its variants such as RoBERTa (from Liu et al. [2019]) and BERTweet (from Nguyen et al. [2020a]), which we will perform experiments on in Chapter 6. These models appear to work well, but as reported by Yin and Zubiaga [2021], the model performance had been grossly overestimated. When cross-dataset experiments are performed, they all show a significant drop in performance, highlighting that the test set of the same dataset does not realistically represent the distribution of unseen data.

The same survey provides a history of models for detecting Hate Speech (or Toxicity). Before 2019, the SoTA usually concerned Recurrent Neural Networks (see Gröndahl et al. [2018]). But, with the introduction of BERT, this and its variants have established new states-of-the-art, with a pipeline that is always similar: take a model pre-trained on domain-general data, and fine-tune it on a target-classification dataset. However, they maintain a lack of generalization (but less than LSTM and BiLSTM). Some recent experiments have also tried to add hate-specific knowledge from outside the fine-tuning dataset, retraining BERT on abusive online communities corpus (as in HateBERT from Caselli et al. [2021]) or providing lexical features extracted from hate speech lexicon (as in HurtBERT, from Koufakou et al. [2020], which exploit the lexicon HurtLex presented in Bassignana et al. [2018]). Another idea by Vidgen et al. [2021] is to improve Hate Speech Detection by dynamically generating datasets, through perturbations and fine-grained labels for the type and target of hate.

Recently, models that make use of explicit syntactic information have been tested for hate speech recognition. The idea behind this is that leveraging the syntactic structure can help to overcome this word-bias, by providing topological features

that can give information on the meaning of the sentence, the rhetoric, etc. In this direction, Mastromattei et al. [2022] present a version of the KERMIT model (from Zanzotto et al. [2020], which exploit constituency trees), called KERM-Hate. This model shows to reduce the bias on words, providing explainable decisions without decreasing the performances (indeed, it obtains better results than Transformer-based models across multiple datasets). Our work follows this direction, and we will give more details in Section 5.1.

Once we have our (trained) model, we can perform further analysis over the test set of our dataset. We can exploit other tools suitable for functional tests, which test the behavior of our models over particular linguistic structures and situations where models struggle. An example is Röttger et al. [2021], who provide a collection of hateful and non-hateful examples carefully created to test the performance of Hate Speech Detection systems over particular patterns (it can be seen as a "checklist"). For example, we go from clearly hateful samples such as "[IDENTITY] are scum.", to non-hateful samples with "trigger" hate words like "Statements like '[IDENTITY] are scum' are deeply hurtful.", or statements with particular syntactic structure as "I have met many [IDENTITY] and I hate every single one of them". We will use this tool for our experiments, and we will provide further information about it in Section 3.2.3. Another similar test is the one proposed by Ribeiro et al. [2020] (but regarding different fields like Sentiment Analysis), just to show how this type of analysis is important not only in Hate Speech Detection but in general on a broad range of NLP tasks.

2.1.2 Conversation level

When authors talk about online conversations in literature, there are two possible meanings: (i) a single thread, where there is a "chain of messages" in temporal order, assuming that the writer of a certain message has read all the messages before; (ii) the conversation tree, where we consider the original post and all comment threads, so whoever writes a message is aware of the messages in the same chain, but we do not know if it reads the others.

The forecasting of toxic conversations (or their derailment) is possible because we can see them as events that evolve over time. Instead, the detection can be viewed as an "extension" of the single message classification.

When it comes to single threads, the usual pipeline is to embed each tweet in

a kind of "tweet embedding", then process those embeddings (as with a simple LSTM) and predict whether the conversation derails or not. An example is presented by Chang and Danescu-Niculescu-Mizil [2019], which also makes use of a "generative pre-training" objective to improve performance. Or more recently Kementchedjhieva and Søgaard [2021] try to confront the model presented by Chang and Danescu-Niculescu-Mizil [2019] with other models that embed tweets thanks to BERT. However, this specific topic has not been explored too much and further research is needed.

More generally, researchers have tried to model these conversations from a topological point of view (focusing on the tree), to predict dynamics (e.g., the size or where the next node will appear in chronological order) as in Bollenbacher et al. [2021], or to describe which people act as "catalyzers" in a conversation (as in Saveski et al. [2021a]), or by proposing interesting ways to model the interactions of the participants (in Zhang et al. [2018]).

In particular on toxicity analysis, researchers have presented both analytical results (as in Majó-Vázquez et al. [2020]) or they exploited only topological features for the forecasting work (Saveski et al. [2021b]) or in combination with linguistic features (Hessel and Lee [2019]).

All the publications cited in this section propose different possibilities for modeling the interactions between participants, supports, responses, etc., without necessarily resorting to Deep Learning techniques such as Graph Neural Networks (in the next section we will report some examples of GNN solutions) but also handcrafted features such as motifs, structural features of the graph, etc.

We believe that in the future a good (and generalizable) linguistic representation in combination with these topological features, that provide the "social dynamics" in which a conversation takes place, is the effective answer to forecast online-toxicity and respond with innovative solutions (see Section 2.3 for some examples).

2.1.3 Network level

Some problems related to online-toxicity are community polarization and echo chambers. Indeed, there are relationships between toxicity (especially disinformation) and these "network" problems: toxic behaviors tend to increase the contrast between opposing factions, users tend to group with like-minded people, creating echo chambers, and here harmful (false, hateful, etc.) opinions and statements tend

to spread without any opposition, increasing again the polarization of people, in a continuous "positive feedback".

One of the milestones in this field was proposed by Garimella et al. [2018], who seek to quantify the controversy over a topic in Social Media, primarily by building retweet graphs, following graphs, contents graphs, and hybrid retweets/contents graphs, and then testing different measures (from Random Walks to betweenness scores) to see which controversial patterns the networks show. Instead, Kumar et al. [2018] try to show how users from different opposing communities tend to interact (with attackers against defenders). Other works also try to reduce polarization by adding edges to the social graph and measuring the reduction (as in Haddadan et al. [2021], who quantify polarization in terms of "bubble radius").

The study of polarized communities and echo chambers can be effective because it allows researchers to design recommended systems adapted to depolarize these networks. An example is to offer the user (non-harmful) content with different positions from those of the user itself or to avoid exasperating the polarization of users by recommending like-minded material (as unfortunately often happens nowadays). However, due to company policies and ethical discussions, this direction has not yet been explored.

2.1.4 Work on different anti-social behaviors

Even though the adjective "toxic" can indicate a wide range of antisocial behaviors (including disinformation), there are some special cases that should be treated differently. Here we report a simple case study example, the solutions of which can also be useful also in detecting online-toxicity: Fake News detection.

Fake News detection can be treated as a toxicity detection subtask if we just consider the posts of a conversation and its users. However, sometimes we want to go "outside" the Social Platform, mainly for three elements: (i) the original article, (ii) the website of the newspaper, and (iii) a knowledge graph from which verify the veracity of a news. More recent works effectively extract all these aspects of the phenomenon, using Deep Learning models capable of embedding information and mixing them together effectively, something that in online-toxicity detection still missing (but we will definitely deal with it in future). We will report in this section only this type of works. Shu et al. [2017] reported a thorough presentation of all these aspects, also giving a definition for different approaches. We refer to it for further information.

In Shu et al. [2019] the reader can find one of the first solutions in which someone combines source information, news/post information, and user information, including also embedding of the relations among these entities (e.g., publishing, spreading, social relation, etc.). In Shu et al. [2020], the proposed model is based only on Social structure (such as those proposed for the toxicity task) and on information from spreading statistics, extracting both linguistic and topological features (such approach can be useful for Early Fake News detection task). Finally, we invite the reader to look at Nguyen et al. [2020b], who propose a similar graph representation to Shu et al. [2019], but also embed the tweets in chronological order, extracting the stance of the single users, the title of the publication website, the possible relationships among publishers, the social context, etc. This work exploits Graph Neural Networks (i.e., GraphSage by Hamilton et al. [2017]) and Transformer-based models (i.e., RoBERTa) to extract topological and linguistic features and then combine node representations. If the reader wants to look at some examples of knowledge-based models, we refer to the works related to the FEVER dataset (presented in Thorne et al. [2018]).

All of these works can be an inspiration for the future of online-toxicity detection, especially for their way of combining topological and linguistic features.

2.2 Technical problems of the field

This research field will be much more important in future. However, there are serious technical problems that limit the possibility of working in this environment and mixing a large amount of data from different domains. Despite the fact that we have the potential to find effective models to represent a great variety of social phenomena (especially Deep Learning models and their ability to extract features automatically), we must take legal and ethical aspects into account.

Working on social data also means on private data. To retrieve them, people need specific permissions (Twitter is the most used because it supports external developers, researchers, etc.) and people cannot freely provide the data to allow reproducibility of experiments or make datasets available for various tasks. The only way to release the data is to either issue anonymized tokens that allow other users with permissions to retrieve the same data or release complete but anonymized texts in agreement with the Social Company.

Ethically, everyone with permission can look at each user's private data as well.

Furthermore, even if they do not release such information openly online, someone can point out ethical issues (for example, I can also find information about people I know, for personal interest). However, users are made aware of these risks when they open an account.

Another problem is that, to remedy possible data leaks, even with a permit the download rate is really low. There are only few researchers/institutions with the potential to retrieve enough data to perform comprehensive experiments with topological information, linguistic information, user data, etc. This is a major bottleneck that limits research activity, and we hope this will change in future.

2.3 Means of response to online-toxicity

Typically, social media companies have responded to toxicity-problems with censorship and content blocking. Although they seemed effective in the past, now we realize they give rise to many side effects. Indeed, if a blocked post is inside an echo chamber, either is about some polarizing topic, the users involved can see it as an attack of political censorship by some enemy organization. Then, the result is greater polarization and more distrust of the content blocking policy, legitimizing a wide variety of problematic content, from conspiracy theories to racist and homophobic ones.

Typically this content blocking was done via user reporting, followed by a quick assessment by a responsible officer. But, due to the extreme subjectivity of the reports, the slowness of the procedure, and the large number of cases to be verified (in addition to the side effects just mentioned), this methodology is in crisis.

Twitter recently proposed an alternative, called Twitter Birdwatch[1]. Mainly designed for the phenomenon of disinformation, with this tool users can identify tweets with misleading information and add notes that provide informative context. This is a first step towards a (potential) highly effective response: counter-speech or counter-narrative.

The research community has also worked in this direction. Tekiroglu et al. [2020] provide an overview of the research landscape, with datasets, methodologies, and interpretations (focusing on the context of hate speech). The main focus is to exploit Natural Language Generators (or NLGs, such as GPT-2, from Radford et al.

[1] https://blog.twitter.com/en_us/topics/product/2021/introducing-birdwatch-a-community-based-approach-to-misinformation

[2019], a Transformer-based model) to generate counter-narrative responses automatically. Even though the direction is promising, and several non-governmental organizations (NGOs) have specialized operators who can help, we are still at an initial stage of this topic, and the main problems are the lack and the quality of data on which we train the models, which lead to generic/repetitive answers.

Counter-narrative combined with techniques for online-toxicity forecasting or early detection has the potential to be an effective pipeline for contrasting toxicity on Social Platforms. However, this must be done under the supervision of human beings, in a human-shared paradigm. This allows models to overcome the limitations of the single human operator, but also vice versa, preventing possible ethical problems (such as bias in models) without sacrificing the efficiency of machines.

Chapter 3

Toxicity detection: Environment and Data

3.1 Twitter and tweets

If we search for the question "What is Twitter?" the official answer[1] is:

> *"Twitter is a service for friends, family, and coworkers to communicate and stay connected through the exchange of quick, frequent messages. People post Tweets, which may contain photos, videos, links, and text. These messages are posted to your profile, sent to your followers, and are searchable on Twitter search."*

From datareportal.com [2] we can analyze some important statistics from Twitter. First, it provides 465.1 million users (by April 2022), and about 5.9% of all people on Earth use it. It is mainly used in North America compared to other regions of the world, and only 28.8% of Twitter's global users are female. Unlike other Social Media platforms, it is known to be the first platform to take a strong stand against hate speech and disinformation by proposing solutions more or less ethical (in addition to simply blocking content, they have also activated the "birdwatch" function[3] where users can make counter-narrative activities). The most famous decision in

[1] https://help.twitter.com/en/resources/new-user-faq
[2] https://datareportal.com/essential-twitter-stats
[3] https://help.twitter.com/en/resources/addressing-misleading-info

this direction was the ban of the former US president Donald Trump[4] on January 8^{th}, 2021 after the January 6^{th} attack on Capitol Hill.

On the official page[1] the reader can find the official definition of:

- Tweet: *"a Tweet is any message posted to Twitter which may contain photos, videos, links, and text"*.
- Retweet: *"a Retweet is a Tweet that you forward to your followers"*.
- Following: *"following someone means you've chosen to subscribe to their Twitter updates. When you follow someone, every time they post a new message, it will appear on your Twitter Home timeline"*.
- Reply: *"A reply is a response to another person's Tweet"*.

What we are going to do is detect toxic Tweets, so we are going to focus on the single message level. A Tweet has some important characteristics (imposed by the Social Platform): (i) a Tweet cannot exceed 280 characters; (ii) a Tweet cannot be edited once posted, only deleted. These features are fundamental (compared to other Social Platform messages) because first of all a message cannot be too long and the post we are going to analyze is the original one, with possible grammar errors or bad writing. Also for this, it is essential once the single message is carefully modeled, to move on to the full conversation. Indeed, if users need to write a long post they usually publish a "chain" of messages with all the information or, if the user finds an error in its original post, sometimes it prefers to correct itself in a subsequent comment instead of deleting it.

3.2 Retrieval of data and datasets available

Most of the research on Social Platforms takes place on Twitter. This is because it provides an easy way to access data with a dedicated platform called Twitter API[5]. However, since we are working on private data, there are severe limitations on the download rate and, above all, on the distribution of the data, and this is a limit for those who want to experiment or replicate works on Twitter. If someone wants to make Twitter data freely available, it can either distribute the IDs of tweets,

[4]https://blog.twitter.com/en_us/topics/company/2020/suspension
[5]https://developer.twitter.com/en/docs/platform-overview

users, etc., and everyone who has a Developer Account can retrieve the original information from it, or it can anonymize the tweets and, once authorized, publish the dataset. The first solution has some problems over time, because users and tweets can be deleted, and therefore doing a good analysis can become difficult.

We will mainly work on three datasets:

- **SemEval 2019 - Hateval**: from Basile et al. [2019] (specifically the version used by Barbieri et al. [2020]), which we will simply call Hateval. This is a dataset for Hate Speech Detection, well tested in the past. This is also useful to see how the models that we cannot test behave.
- **Intolerance**: this is a new dataset that we created, extracted from a corpus of conversations annotated at the single message level, provided by an anonymous University[6] in collaboration with Bocconi University. This allows us to see if a model can learn useful information from a more realistic scenario.
- **HateCheck**: from Röttger et al. [2021], this is a dataset of functional tests in order to identify where a model struggles and where it is successful in some particular linguistic contexts. It allows us to perform even a simple overview of error analysis.

3.2.1 SemEval2019 - Hateval

The first dataset we will use is a typical challenge dataset, with an already given training set, validation set, and test set. As mentioned earlier, it was presented by Basile et al. [2019], providing a total of 19600 tweets, 13000 for English and 6600 for Spanish, distributed on two main targets: immigrants (9091 tweets) and women (10509), with classes "Hate" and "Non-Hate". However, we used another version, presented by Barbieri et al. [2020] for Hate Speech detection activity in English only. The authors of this release provide 9000 training samples, 1000 validation samples, and 2970 test samples, with the same balance between positive (hateful) and negative (non-hateful) samples (the ratio of positive/negative samples is approx. $43/57$). Details can be found in Table 3.1.

This dataset is critical to our research because it delivers the results for a large number of models while also doing good hyperparameter optimization, which helped

[6]These conversations have been collected for future works with Bocconi University. We were allowed to use it for our tests, but we cannot release the data or details about the annotators.

us to interpret our results. The main evaluation metric is the Macro average F1 score (M-F1) on the two labels.

Basile et al. [2019] report the SoTA to 65.1, while in Barbieri et al. [2020] the top model reported is a solution based on BERTweet (details in Section 4.2.2) that reaches the score of 56.4[7] . As we will see in our experiments, although we use similar models, we will not reach such results with a typical BERTweet model (but we will present a particular variant that works very well) because we have not performed careful hyperparameter optimization. In any case, we are in line with RoBERTa-based models, and two of our non-transformer-based models outperform them (not reaching the SoTA unfortunately). Another note to be remarked is that the SoTA regards a slightly different dataset from the one used by us.

3.2.2 Intolerance

This second dataset has never been presented (and cannot be openly distributed at the time of writing), but it is meant to be a new and difficult dataset on which to test our model, maintaining the same configuration and experimental setup.

This dataset was annotated by highly specialized annotators (social and political researchers) on a collection of complete conversations, where every single post/comment has been labeled with a great variety of toxic (or related) labels: **profanity**, **insults**, **character assassination**, **outrage**, **hateful speech**, **dehumanisation**, **serious threat and harassment**, **discrimination**, **democratic threat** or **counter-speech**. Between these, a first group of labels (profanity, insults, character assassination, and outrage) can be grouped into a super set called **incivility**, while a second group of labels (hateful speech, dehumanisation, serious threat and harassment, and discrimination) can be grouped into another super set called **intolerance**.

From this initial collection of conversations, we have pulled a new dataset for binary classification on the "super label" intolerance. We "destroyed" any conversation relationship between tweets, generating training, validation, and test set by sampling items at random, splitting the data between the three sets following the 60/20/20 division, and maintaining the same balance between labels in the different sets. Table 3.1 shows that there are few positive samples (Toxic labels) compared to the negative ones (Non-Toxic labels). Hence, this dataset gives us the opportunity to:

[7] https://github.com/cardiffnlp/tweeteval

- test our models on tweets with possibly different linguistic characteristics compared to Hateval.

- Test our models on a more "realistic scenario", where in conversation there could be a great imbalance between the number of toxic messages and non-toxic ones.

- Since even in terms of absolute representation, there are only a few hundred positive samples, if our new model (the Syntactic Encoder, presented in Chapter 5) can obtain some interesting results, probably not acceptable in a real application but shows to capture some information in really difficult situations, it can be a hint on its ability to embed useful information also in scarcity of data.

3.2.3 HateCheck

Presented by Röttger et al. [2021], HateCheck is a functional test suite for hate speech detection models. It tests 29 model functionalities, 18 for different expressions of hate and 11 for expressions of non-hate (which however shares linguistic features with hateful expressions), through: (i) different types of derogatory hate speech; (ii) hate expressed through threatening language; (iii) hate expressed using slurs and profanity; (iv) hate expressed through pronoun reference negation and phrasing variants (specifically questions and opinions); (v) hate containing spelling variations such as missing characters or leet speak; (vi) non-hateful contrasts for slurs (particularly slur homonyms and reclaimed slurs), as well as for profanity; (vii) non-hateful contrasts that use negation (i.e. negated hate); (viii) non-hateful contrasts around protected group identifiers; (ix) contrasts in which hate speech is quoted or referenced to non-hateful effect, specifically counter speech (i.e. direct responses to hate speech which seek to act against it); (x) non-hateful contrasts that target out-of-scope entities such as objects rather than a protected group.

To generate test cases at scale, authors use templates, in which they substitute tokens for protected group identifiers (e.g., "I hate [IDENTITY].") and slurs (e.g., "You are just a [SLUR] to me."), guaranteeing an equal number of cases addressed to different protected groups. For this reason, this dataset is also useful for performing other types of tests: (i) performance on individual functional tests, for the terms "n*gga", "f*g", "f*ggot", "q*eer" and "b*tch", referring to "reclaimed slurs"

Dataset	Train-1	Train-0	Val-1	Val-0	Test-1	Test-0
Hateval	3783	5127	427	573	1252	1718
Intolerance	342	4335	1444	115	1444	115

Table 3.1: Number of positive (Hate/Toxic) and negative (Non-Hate/Non-Toxic) samples on Training set, Validation set, and Test set on Hateval and Intolerance.

functional test by which slur is used in non-hateful case; (ii) performance among target groups, specifically women, trans people, gay people, black people, disabled people, Muslims, and immigrants, comparing the performance of the models on the cases that target these groups.

In the end, Hatecheck provides 3,901 samples, 3,495 of which come from 460 templates, and the remaining 406 cases do not use a template.

Chapter 4

Tweet embeddings: sequential models

Recursive Neural Networks (RecNN in short, which we will cover in detail in Chapter 5), to the best of our knowledge, have never been exploited for hate speech detection. Furthermore, in general, they have not been tested so much mostly due to implementation difficulties and for the advancement of Transformer-based Models, which has moved away researchers from testing them.

To make a correct analysis of the strengths of our model, we need well-known solutions proposed in the past, in particular, a non-trivial baseline (the BiLSTM) and a more advanced model that is hard to beat but that can be used as a reference for our performances (BERTweet). In this chapter we will show the main characteristics of these models, taking for granted the functioning of Recurrent Neural Networks (RNN) and the functioning of Long Short-Term Memory mechanism (LSTM). The reader can find further details in Goodfellow et al. [2016].

For a summary only, RNNs are a particular type of Neural Networks which work on sequences (like sentences), applying the same processing at each step. Due to the possibility of encountering long sequences, we can experience the phenomenon of the so-called "vanishing gradient", for which the inputs at the beginning of the sequence have a minor effect on the final output (although they can be important) and hardly affects the parameters training process. In response to that, researchers designed LSTM cells, which thanks to a system of "gates" (input gate,

output gate, update gate and forget gates) the system can learn to "forget" useless information and "keep" useful information, computing a cell memory vector c and a hidden states h. In this case, the most important aspect of system design is the definition of the gate functions (and which vectors are used to calculate them).

This chapter is strongly inspired by Voita [2020], from which we have drawn also diagrams to show how the different systems work. We highly recommend visiting it for further explanations.

4.1 BiLSTM over GloVe embeddings

The model we used as a "baseline" is a BiLSTM on GloVe embeddings (in the next section we will put all the related information). This type of model has been studied extensively in the past, and it is one of the most used for sentence classification. The idea behind it is to embed sentence information based on word order, left to right and right to left. As expected, the words are replaced by embedding vectors (taken from GloVe "as it is", not fine-tuned) and attaching two special learnable vectors, at the beginning a **beginning-of-sequence vector (<BoS>)** and at the end an **end-of-sequence vector (<EoS >)**. You can see a schematic view in Figure 4.1.

4.1.1 GloVe

GloVe (together with Word2Vec, from Mikolov et al. [2013]) is one of the typical neural word embeddings exploited in NLP applications. Presented in Pennington et al. [2014], its name is short for "Global Vectors" because the model is also based on capturing global corpus statistics. It can be seen as a combination of count-based methods and prediction methods (like Word2Vec). GloVe uses co-occurrence count to measure the association between word w and context c (denoted $N(w, c)$) and use them to construct the loss function (see Figure 4.2).

In this work, we will use a particular version of GloVe, trained on Twitter (therefore suitable for our objectives) on over 2 billion tweets, 27 billion tokens, 1.2 million words in the vocabulary, uncased, 50 in vector dimension, published by StanfordNLP[1]. We will use it as input both for the BiLSTM in this chapter and for the new Syntactic Encoder, discussed in depth in Chapter 5.4. The size of the vector

[1] https://nlp.stanford.edu/projects/glove/

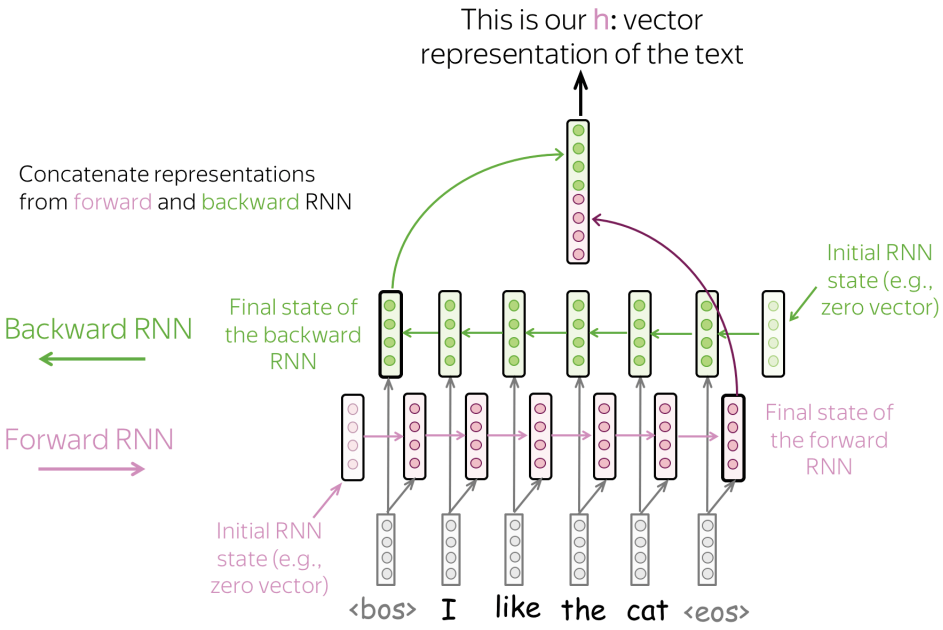

Figure 4.1: Schematic view of a BiLSTM model for sentence classification, from Voita [2020]

is not the maximum one (which is 200). However, since we want to show that the syntactic structure is effective, we do not want huge word embedding vectors, to make this task more difficult for our model.

4.1.2 BiLSTM

The BiLSTM is a particular type of *bidirectional* LSTM where, given an input sequence $\{x_1, x_2, ..., x_N\}$, we apply two separate LSTMs: one is called **forward** LSTM which reads the sequence from left to right, and the other is called **back-**

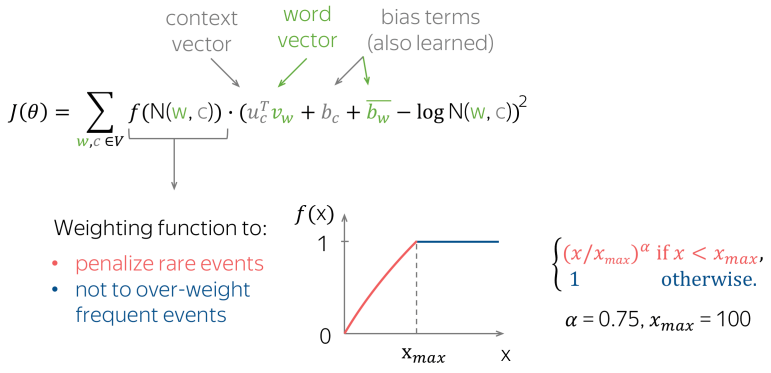

Figure 4.2: Schematic view for the computation of GloVe loss function, from Voita [2020]

ward LSTM, which reads the sequence from right to left. What we output is a sequence of forward hidden states $\{h_1^f, h_2^f, .., h_N^f\}$ and a sequence of backward hidden states $\{h_1^b, h_2^b, ..., h_N^b\}$. Remember from the introduction of this section that we have attached also the two learnable vectors <BoS>and <EoS>, and also from them we calculate two final hidden states each (h_{BoS}^b and h_{BoS}^f for the first, h_{EoS}^b and h_{EoS}^f for the second).

The final vector representing the embedding of our sentence is the concatenation of h_{EoS}^f with h_{BoS}^b (formally, $h = [h_{EoS}^f | h_{BoS}^b]$), which is essentially the concatenation of the last hidden vector calculated by each LSTM, as in Figure 4.1. Once we have extracted this vector, we perform the prediction through a special FeedForward Network called Multi-Layer Perceptron (MLP for short), which is a sequence of Fully Connected layers, each followed by a specific non-linear function. It returns as output a probability distribution on the classes of the specific classification task (binary in our case). The MLP applied to this vector is the same that will be presented in Section 5.4.6 to make a fair comparison between these models based on GloVe embeddings. We refer to that subsection for further details.

4.2 BERT-based solutions

In recent years, the NLP research has been dominated by Transformer-based solutions[2], giving rise to the so-called "Foundation Models" (also called Large Language Models), which are (i) characterized by a huge number of parameters, (ii) trained on broad data at scale and (iii) can be adapted to a wide range of downstream tasks (Bommasani et al. [2021] provided this definition).

The results achieved by these models are extremely high, but they are not free from technical and ethical problems. The main point is that figuring out which aspect of the language they are looking at is extremely difficult, and researchers are still arguing (have they embedded some syntactic information? Do they just observe word co-occurrences? Is it similar to Natural Language Understanding, or is it more a really good pattern matching?). The 2021 year is recognized as the year of Large Language Models due to the explosion of such works (companies invest a lot in them), and for this, important researchers have taken a stand to make people aware of the risk that such models pose (Bommasani et al. [2021]).

[2]Transformers were first presented by Vaswani et al. [2017] and they have brought about a turning point in many computational fields

Of all these models, we focus on the most famous one: BERT (from Devlin et al. [2019]) and, more specifically, its variant BERTweet (Nguyen et al. [2020a]). In the next subsection, we will briefly explain some technical details of these two models (and the difference between BERTweet and BERT) and we will recall only some notions of the Transformer mechanism. However, for the full picture we refer to the original papers, or Voita [2020] (from which the images are also taken), or Jurafsky and Martin [2021].

We perform experiments on BERT for various reasons. First of all, there are various pre-trained versions available online, for example from the Hugging Face library[3], which can provide ready-to-fine-tune models which usually give good performance. However, this is also due to the large number of parameters they provide. We do not expect to reach BERT performance because our models are much smaller (technical details in Chapter 6), but it can still be interesting to compare and understand where a simple model can achieve comparable results to these huge models.

For example, BERT is recognized for struggling with negatives and small changes in the syntactic structure which however lead to a total modification of the meaning of the sentence.

4.2.1 BERT

BERT stands for "Bidirectional Encoder Representations from Transformers". It is essentially a Transformer's encoder, which consists of a sequence of "encoder" blocks as shown in Figure 4.3. Within these computational units, the self-attention mechanism is widely exploited (for further information we refer to Vaswani et al. [2017]).

We give in input to BERT the tokenized sentence (with the related embeddings) by adding two particular tokens:

- [CLS] token: this is a special token placed at the beginning.
- [SEP] token: called also token-separator, we put this token between different sentences.

[3]https://huggingface.co/docs/transformers/index

In addition to token embeddings we also put positional embeddings to keep track of word order and segment embeddings, to allow the model to easily distinguish between different sentences (we show this in Figure 4.4). During training, we give in input to BERT a pair of sentences, so what BERT sees is a [CLS] token, followed by the tokens of the first sentence tokens, then a [SEP] token followed by the tokens of the second sentence and finally another [SEP] token.

Some of the new features introduced in BERT are the training objectives from unlabeled data. In detail, they are called:

- **Next Sentence Prediction Objective (NSP)**: this is a binary classification task in which, from the final layer, we extract the final representation of the [CLS] token to predict whether the two input sentences given are consecutive or not (in the original texts). In training, 50% of examples are positive and 50% are negative. This task teaches the model to understand relationships between sentences or, as we will do, also to reason on the sentence and execute text classification.

- **Masked Language Modeling Objective (MLM)**: in MLM objective, at each step, we (i) select some tokens with probability $p = 0.15$, (ii) then replace them either with a special token called [MASK] with probability $p = 0.8$, or with a random token with probability $p = 0.1$ or with the original token with probability $p = 0.1$, and (iii) finally predict the original token (see Figure 4.5 for a schematic view).

Of these two, the most important objective is MLM. This is because it allows BERT to produce a "contextual" word embedding for each word after learning (thanks to the self-attention mechanism, looking at the whole sentence), which can be used for the same activities in the same way we use "static" word embeddings, such as Word2Vec or GloVe. Unlike the static ones, we can also fine-tune BERT to learn embeddings suitable for the task we want to carry out, giving a great power of representation.

After this step called **pre-training**[4] it's time to fine-tune it for downstream tasks. In Figure 4.6 the reader can see a schematic view of the pre-training step followed

[4]Usually the available models are already pre-trained because this step requires an enormous amount of computational power, time, and data

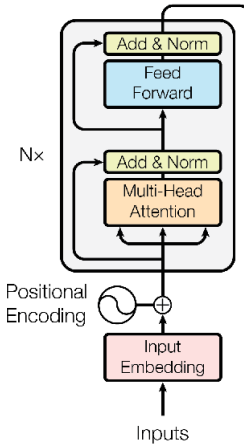

Figure 4.3: Transformer's encoder block representation, from Vaswani et al. [2017].

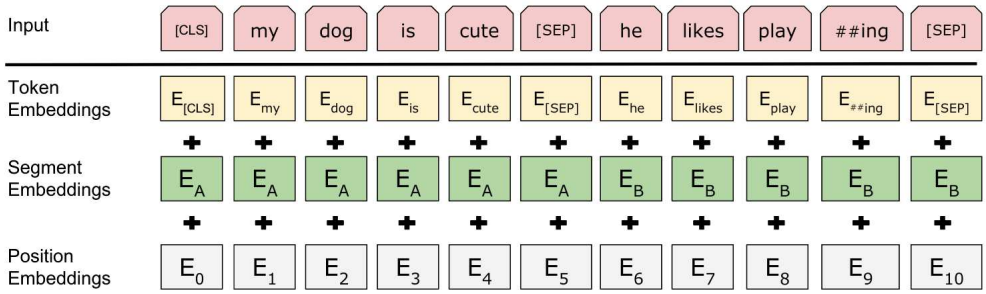

Figure 4.4: BERT's input format, from Devlin et al. [2019].

by fine-tuning, as presented in the original paper.

Toxicity detection can be seen as a particular type of sentence classification. What we do is to "attach" an MLP layer, providing as input the final representation of the [CLS] token and then backpropagate both on the MLP parameters and on the BERT parameters (with regularization tricks).

4.2.2 BERTweet

Now that we have briefly introduced the BERT model, it's time to talk about its tweet-adapted variant: BERTweet. Presented by Nguyen et al. [2020a], BERTweet is a model with the same architecture as BERT but uses RoBERTa's pre-training procedure. RoBERTa is an optimized version of BERT proposed by Liu et al. [2019], where the authors carefully chose the model hyperparameters showing that the original BERT was extremely undertrained.

BERTweet is trained over a corpus of 850 million tweets in English, where each tweet has at least 10 word tokens and a maximum of 64. We will work on sequences

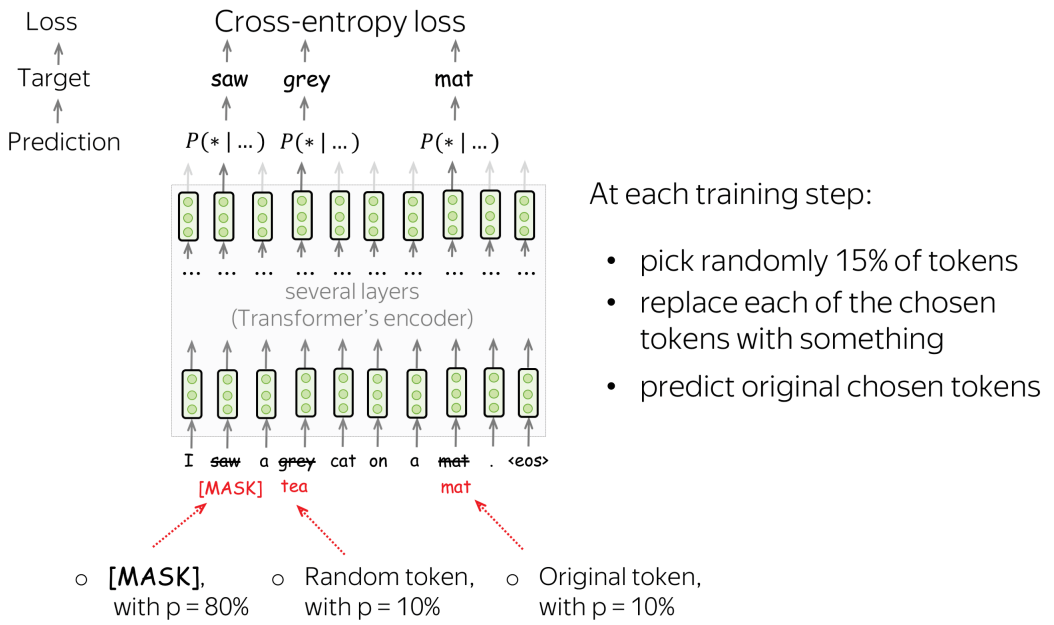

Figure 4.5: Schematic view of BERT's pre-training through MLM objective, from Voita [2020].

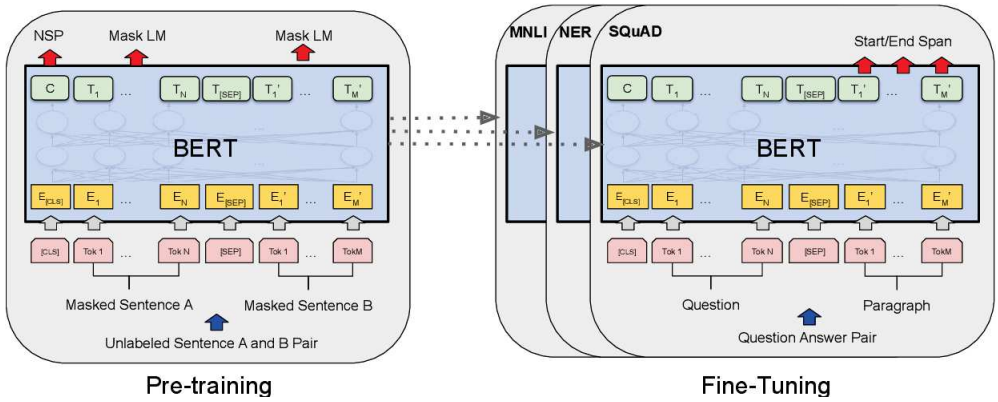

Figure 4.6: Pre-training and fine-tuning schematic representation, from Devlin et al. [2019].

of up to 130 tokens, and this could lead to generalization problems that should be analyzed in future works.

The authors proved that BERTweet is capable of outperforming its baselines RoBERTa and XML-R (from Conneau et al. [2020]) and beating the previous SoTA models in three downstream Tweet NLP tasks: POS tagging, NER (Named Entity Recognition), and text classification. For all these reasons, we believe that this model is also suitable for achieving good performance in our task and keep it as a reference for our work.

Chapter 5

Tweet embeddings: syntactic models

Until now we have presented models that work on word sequences. What they can capture is typically considered semantic/contextual information and currently, there is a strong debate about their capacity of capturing syntactic information. Especially for BERT, the research community agrees Transformer-based models can capture some sort of syntactic features, but we do not have clear yet what they can represent. Furthermore, some works show that it has strong limitations in generalizing this syntactic knowledge (as in Weissenhorn et al. [2022]). So, we are trying to exploit the syntactic structure as much as possible, with models that provide fewer parameters than BERT but can reach good performance, and have the possibility to keep track of syntactic features. We want to show that making use of explicit syntactic information can be a direction for future research.

5.1 Syntactic-aware models

The comparison between semantic/contextual models and syntactic ones has recently gained interest in the research community, because of the uncertainty on how Transformer-based models work and the will of including in modern tools also pre-DL solutions, leading to more guided and explainable models.

KERMIT by Zanzotto et al. [2020] deeply inspired our work, and it has been also tested in the context of Hate Speech Recognition (HSR) by Mastromattei et al. [2022]. KERMIT is a model for sentence embedding (exploited in the task of sentence classification) where, given a sentence, it extracts a "semantic" representation

thanks to a Transformer-based model and a "syntactic" one, embedding the relative Phrase Structure Tree in a vector exploiting tree kernels technology. Finally, they concatenate these two vectors and perform the classification passing through a Multi-Layer Perceptron.

Mastromattei et al. [2022] present its variant KERM-HATE, used as said above for HSR, where they showed that syntactic-aware models can outperform BERT-based models and reduce possible biases since it focuses on structural features, which are "ethically unbiased". However, they finally show that even syntax-based models absorb prejudice from data. They also perform a post hoc explanation, which confirms our previous assumption that syntax-based models provide more explainable decisions.

Though this is the main work that inspired us, our solution is different. First of all, we focus on dependency trees, where the structure itself is represented by words and their relations. Then, our model is simpler, with a smaller final syntactic vector (for instance KERM-HATE provides in output a syntactic vector of dimension $d = 4000$, but in our experiments, we will output a vector of dimension $d = 100$). Finally, we do not combine any semantic vector. Although we agree a complete work should combine syntactic and semantic vectors, in our investigation we want to find clear syntactic-based models, without exploiting a huge number of parameters, and show that they can capture some interesting aspects of the language, on which researchers can base future works in this direction, with much more complex and accurate models.

5.2 Dependency Trees and Tweebo Parser

This theoretical overview about dependency trees is taken from Jurafsky and Martin [2021]. For further information, we suggest the reader to take it for reference.

Dependency grammars are an important family of grammar formalism, where the syntactic structure of a sentence is described in terms of binary relationships among words. Such relations are described by directed arcs from heads to dependents (possibly labeled), with a particular root node, which explicitly marks the root of the tree. But what do these relations mean?

As introduced before, for each dependency relation we have a head and a dependent, and the head-dependent relationship is made explicit by directly linking

heads to words that are directly dependent on them. We have also the possibility to define the type of relation (but we haven't exploited this case in our work).

More formally, a general dependency structure is a graph $G = (V, E)$ where the set of nodes V corresponds to words and the set of arcs E corresponds to the head-dependent relations. In the specific case of the dependency tree, we have a directed graph where (i) there is a single designated root with no incoming arcs and (ii) each vertex has one incoming arc (except the root node).

Concerning models applied to a sequence of words (e.g., BiLSTM on word embeddings), dependency trees give us the possibility to work on another type of structure. This permits to extract possible information about the style of writing, rhetorics, and also to combine word embeddings in a "syntactic-style" way. To do so, we have exploited the Tweebo parser proposed in Kong et al. [2014], presented as the first syntactic dependency parser designed explicitly for English tweets, and freely available online[1]. The reader can see an example of a tree obtained from this parser in Figure 5.1

Tweebo parser combines a second-order Turbo parser (as proposed by Martins et al. [2013]) and the POS-tagger by Owoputi et al. [2013], producing a dependency tree for each sentence in the tweet with labeled nodes (the POS tag of the corresponding word) and unlabeled arcs. The possible POS tags are reported in Table 5.1. Some words are not assigned to any dependency tree (like user tags, hashtags, and punctuation, which have no relevant syntactic roles) and, as we will see in the experiments, also this filtering work can be effective, even if we ignore the syntactic trees.

5.3 Recursive Neural Networks

Embedding structural information into a vector is not a trivial step. Since we are working on trees, we aim to translate each dependency tree into a "tree embedding", trying to force the model to visit such a structure and learn to exploit topological information.

Recursive Neural Networks (RecNN, not to be confused with Recurrent Neural Networks) are a specific family of Deep Neural Networks that can be useful in

[1]http://www. ark.cs.cmu.edu/TweetNLP

N	common noun
O	pronoun (personal, not possessive)
^	proper noun
S	nominal + possessive
Z	proper noun + possessive
V	verb including copula, auxiliares
L	nominal + verbal or vice versa
M	proper noun + verbal
A	adjective
R	adverb
!	interjection
D	determiner
P	pre- or postposition, or subordinating conjuction
&	coordinating conjuction
T	verb particle
X	existential there, predeterminers
Y	X + verbal
#	hashtag (indicates topic/category for tweet)
@	at-mention (indicates a user as a recipient of a tweet)
~	discourse marker, indications of continuation across multiple tweets
U	URL or email address
E	emoticon
$	numeral
,	punctuation
G	other abbreviations, foreign words, possessive endings, symbols, garbage

Table 5.1: POS tags from Owoputi et al. [2013]

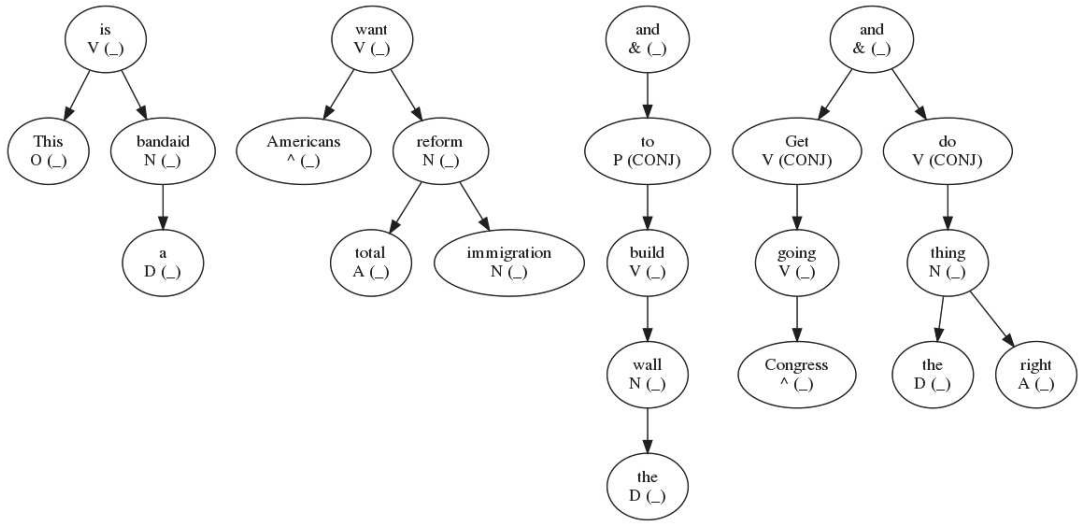

Figure 5.1: Example of dependency trees obtained from the tweet: "@user This is a bandaid! Americans want total immigration reform and to build the wall! Get going Congress and do the right thing! #BuildThatWall #ImmigrationReform"

this direction. The idea behind is to apply the same processing recursively over a particular structure, and in this way, the goal is to extract a sort of structure-based representation.

To start our research we have taken strong inspiration from Socher et al. [2013] and especially Tai et al. [2015]. In the last one, the authors proposed two possible variants of Tree LSTM models (a RecNN for trees, using LSTM units), one for a fixed number of children, called N-ary Tree LSTMs, and the other for a variable number of children, called Child-Sum Tree LSTMs (Figure 5.2), where, given a parent p and its children $C(p)$, to compute the hidden state of p they apply a set of weights to p and $C(p)$, combined. Since we are working with dependency trees where every node has a variable number of children, we report here in detail how the Child-Sum version works, which can be seen as a simplified version of the model proposed in Section 5.4.

As a classic LSTM unit, we compute input gate i, forget gate f, output gate o and update gate u. Assuming we have a node j and its children set $C(j)$, we compute these gates based on:

- the node representation x_j;
- the "cumulative" hidden representation $\tilde{h}_j$ of $C(j)$

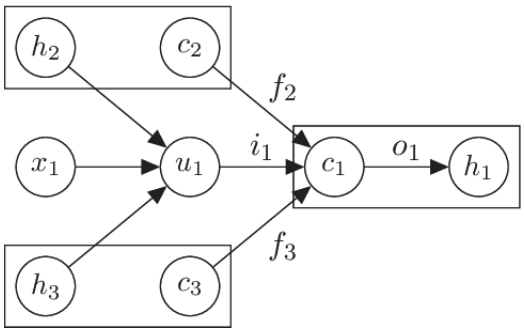

Figure 5.2: Original representation of the Tree-LSTM, from Tai et al. [2015]

Then we compute the new cell memory c_j and new hidden representation h_j also according to a "cumulative value" of all c_k, where $k \in C(j)$. In formulas:

$$\tilde{h}_j = \sum_{k \in C(j)} h_k, \tag{5.1}$$

$$i_j = \sigma(W^{(i)} x_j + U^{(i)} \tilde{h}_j + b^{(i)}), \tag{5.2}$$

$$f_{jk} = \sigma(W^{(f)} x_j + U^{(f)} h_k + b^{(f)}), \tag{5.3}$$

$$o_j = \sigma(W^{(o)} x_j + U^{(o)} \tilde{h}_j + b^{(o)}), \tag{5.4}$$

$$u_j = \sigma(W^{(u)} x_j + U^{(o)} \tilde{h}_j + b^{(u)}), \tag{5.5}$$

$$c_j = i_j \odot u_j + \sum_{k \in C(j)} f_{jk} \odot c_k, \tag{5.6}$$

$$h_j = o_j \odot \tanh(c_j) \tag{5.7}$$

Notice that we have a specific forget gate for each child. The function $\odot$ represents an element-wise product between vectors.

This model does not take count of the order in which children appear and performs a simple sum of the hidden representations to compute the gates and a sort of weighted sum for the new memory cell. For us, this is a relevant weakness since we believe that such an order is important, and we are going to account for this in the next Section.

5.4 Syntactic Encoder

We now present in detail our new model and the idea behind it. We can divide it into four steps:

- given a tweet, create the associated dependency trees (one per each sentence)

using the Tweebo parser and apply minimal preprocessing to "clean" the words.

- Apply a BiLSTM to mix the word embeddings concatenated with their relative POS tags, obtaining a new contextual representation for each word.
- Perform a **top-down filtering** to carry information from ancestors to descendants with an LSTM over each path root-leaf.
- Wrap up information according to a well-defined **bottom-up** procedure (subsection 5.4.4).
- Run a standard LSTM over the tree embeddings and apply a final MLP with softmax to perform the prediction

We are going to dedicate a subsection to each step, to let the reader understand in deep how this system works. The strength points of this model are:

- its ability to distribute information depending on the word-ordering (BiLSTM) and on the tree structure, from ancestors to descendants and vice versa.
- We have no assumptions on the number of children of each node, so it applies to general trees.
- taken the node j and its children set $C(j)$, we process the elements in $C(j)$ maintaining the relative order in the sentence and performing a more accurate computation with respect to the simple sum.

Those ideas have been also proposed in the past, like in Teng and Zhang [2016] and Chen et al. [2017]. But, to the best of our knowledge, this is the first time that someone tries to perform an exploration of the syntax tree so carefully, taking account of both ascendants, descendants, and previous siblings.

5.4.1 Tweet parsing and preprocessing

As described in Section 5.2, we pass all the tweets to Tweebo Parser, which gives in output the corresponding dependency trees and the POS tag of each word. After this, we need to find a vectorial representation for each word.

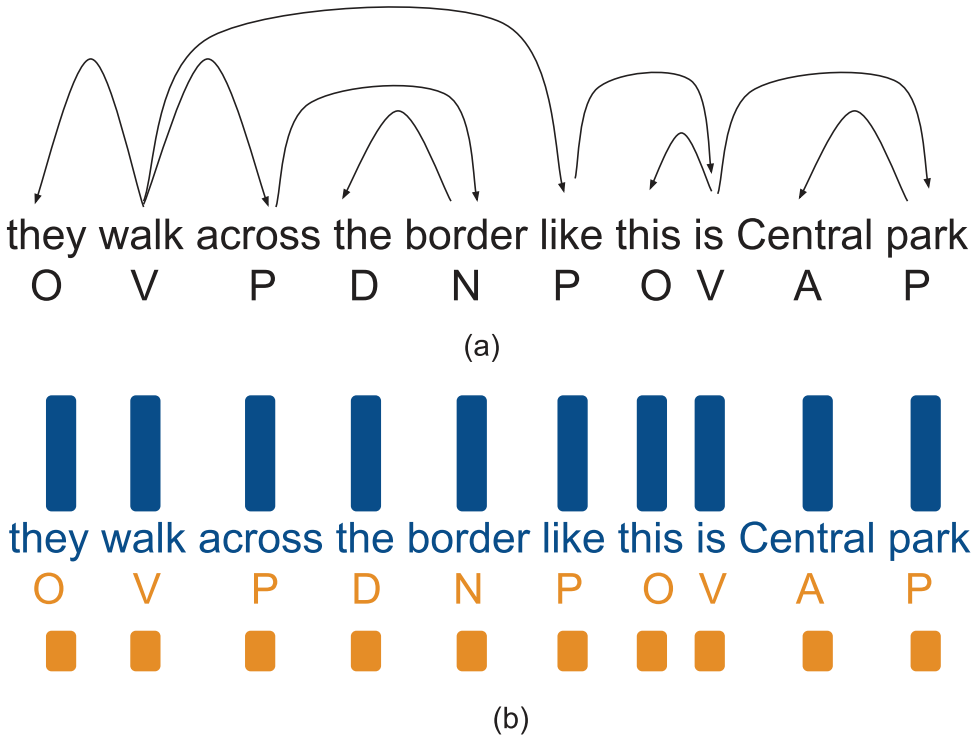

Figure 5.3: Upper (a): example of dependency tree and POS tags obtained from a sentence. Lower (b): example of sequence of embeddings

To do so we use GloVe embeddings trained on tweets from StanfordNLP[2] (we have talked about this in Section 4.1.1). To reduce the number of unknown words, we have divided the common abbreviations (e.g., I'm → I am, You'll → You will, etc.) managing accurately also the dependency tree (we divide the two words and add an arc from the first word to the second). If a word is totally unknown, we assign to it the average of the word embeddings present in the same sentence. On the other side, we represent the POS tag as a one-hot vector (we have 25 possible tags, so we use a vector of 25 dimensions). After this process, we have "clean" words and tweets divided into sentences/dependency trees (the reader can observe a schematic representation of the embeddings and their relations in a single sentence/tree in Figure 5.3).

5.4.2 BiLSTM

After the first step, we are ready for the first processing. We run on each sentence a BiLSTM to extract information not only from the single word but also from the

[2]https://nlp.stanford.edu/projects/glove/

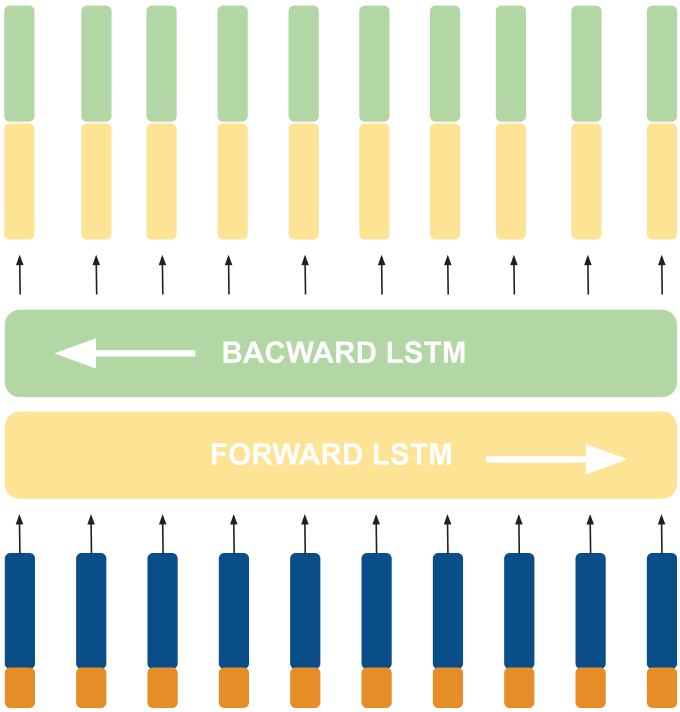

Figure 5.4: Graphical representation of the BiLSTM step

context. The input given to the BiLSTM is a concatenation of word embedding and POS tag of the type $[word|POS]$. As usual, for each word we concatenate the forward representation and the backward one (Figure 5.4). Differently from the classic BiLSTM we do not add a <BoS> and an <EoS> vectors, since we are not extracting a sentence embedding. It could be interesting in the future to exploit such a trick.

5.4.3 Top-down filtering

We now present our first "exploration" of the tree. Essentially we run an LSTM on each root-leaf path. If two paths share a portion, let's say until node j, we do not run two separate procedures, but we perform a unique process until node j, and then we "split" the LSTM between j's children. In this way, we obtain for each node a new representation that depends only on the node representation and its ancestors (taking into account the different relations head-dependent and higher-order relations).

More formally, for each word j we obtain a new vector s_j by running an LSTM on the unique path from the root r of the parse tree to node j, in a top-down fashion, assigning to each node k in the path from r to j the hidden state s_k. So for input

gate, output gate, forget gate, update gate, and hidden state we have to learn two matrices of parameters A and B, plus bias b. In Figure 5.5 the reader can see an example with only two children, but the system can work with an arbitrary number.

Assuming we are processing the node j, given p its parent node, and g is a general non-linear function, the equations obtained for the TD-LSTM (Top-Down LSTM) are:

$$i_j = \sigma(A^{(i)}s_p + B^{(i)}w_j + b^{(i)}) \tag{5.8}$$
$$f_j = \sigma(A^{(f)}s_p + B^{(f)}w_j + b^{(f)}) \tag{5.9}$$
$$o_j = \sigma(A^{(o)}s_p + B^{(o)}w_j + b^{(o)}) \tag{5.10}$$
$$u_j = g(A^{(u)}s_p + B^{(u)}w_j + b^{(u)}) \tag{5.11}$$
$$c_j = i_j \odot u_j + f_j \odot c_p \tag{5.12}$$
$$s_j = o_j \odot g(c_j) \tag{5.13}$$

After the computation is made over all the nodes (i.e., for each node j we have the relative hidden state s_j) we perform the bottom-up process as in Section 5.4.4.

Assuming the root r with a dummy parent p_r, the initial LSTM vectors s_{p_r} and c_{p_r} are imposed to be null vectors ($s_{p_r} = \bar{0}$, $c_{p_r} = \bar{0}$).

5.4.4 Bottom-up processing

We now want to obtain a final **tree embedding**, processing the different node vectors taking into account:

- the structure of the tree;
- the relation parent-child and higher-order;
- the node order between children of the same parent.

Let p be an internal node of a tree, and let $1, 2, \ldots, N$ be its children nodes. We write $s_p, s_1, s_2, ..., s_N$ to denote node vectors obtained from the TD-LSTM in Section 5.4.3, associated to the above nodes.

Let T_j be the subtree rooted at child node j. When we process node j as a child of p, we already have available:

- a hidden state h_{j-1} and a cell memory value c_{j-1} representing the processing of all of the left siblings of j;

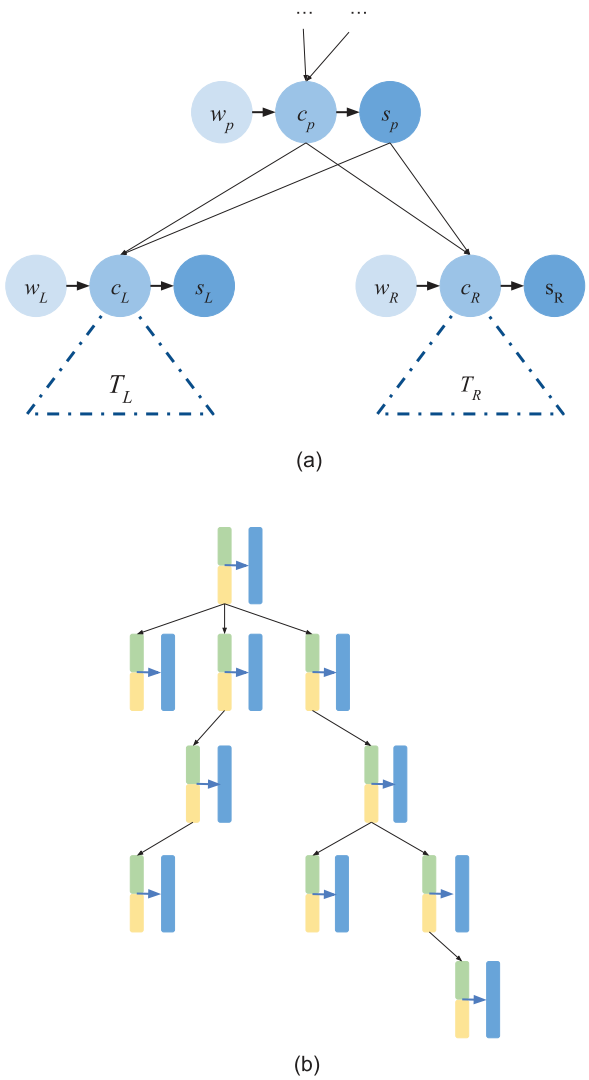

Figure 5.5: Schematic view of how the top-down LSTM processes the tree. Upper (a): computation of the s vector for a node and its children (assuming it has two children). Lower (b): representation of the path covered by the LSTM in its top-down traversal. It takes in input the yellow and green vectors (from the BiLSTM) of the node and the blue one of the parent, returning the blue vector of the node. The black arrows show the order of computation (it follows the dependency tree edges).

- a hidden state x_j representing the processing of the entire subtree T_j;
- the embedding s_p at parent node p from the TD-LSTM.

Essentially, this system produces two hidden states for each node:

- h_j, which is associated with the sequential processing of the LSTM from the left siblings of node j;
- x_j, which is the hidden state resulting from the bottom-up procedure after the processing of the whole subtree rooted in j.

The idea is as follows: taken the node p, we run the LSTM over the sequence of its children from left to right. Once we have computed the last child representation h_N, we apply a specialized neural network (with a specific learnable matrix X and non linear function g_x) and compute x_p associated to the node p:

$$x_p = g_x(Xh_N) \tag{5.14}$$

After that, p is ready to be processed with its siblings to compute the representation of its parent.

For each gate we have three learnable matrices: U for h_{j-1}, Y for x_j and Z for s_p, plus the bias vector b, and g is a general non linear function.

$$\begin{aligned}
i_j &= \sigma(U^{(i)}h_{j-1} + Y^{(i)}x_j + Z^{(i)}s_p + b^{(i)}) && (5.15)\\
f_j &= \sigma(U^{(f)}h_{j-1} + Y^{(f)}x_j + Z^{(f)}s_p + b^{(f)}) && (5.16)\\
o_j &= \sigma(U^{(o)}h_{j-1} + Y^{(o)}x_j + Z^{(o)}s_p + b^{(o)}) && (5.17)\\
u_j &= g(U^{(u)}h_{j-1} + Y^{(u)}x_j + Z^{(u)}s_p + b^{(u)}) && (5.18)\\
c_j &= i_j \odot u_j + f_j \odot c_{j-1} && (5.19)\\
h_j &= o_j \odot g(c_j) && (5.20)
\end{aligned}$$

Also here we need to treat some “particular” cases. Specifically:

- for what concerns the leaves, which have no children, assuming we have the leaf node l, we compute the relative BU (bottom-up) embedding x_l with a specific neural network, with learnable matrix L and non-linear function g_l giving in input the TD embedding s_l:

$$x_l = g_l(Ls_l) \tag{5.21}$$

- Given a children sequence $1, 2, ..., N$, we initialize h_0 and c_0 as null vectors (as before, $h_0 = \bar{0}$, $c_0 = \bar{0}$).

The last trick regards the embedding associated with the root word. We want to make "explicit" that a specific node is the final one. To do so, we apply another neural network with learnable matrix R and non-linearity g_r with its BU embedding as input, to find the final tree-embedding:

$$r = g_r(Rx_r) \tag{5.22}$$

The reader can view a schematic representation of the bottom-up process presented above in Figure 5.6.

5.4.5 LSTM over tree embeddings

As mentioned in Section 5.2, from each tweet we can extract several sentences and a dependency tree for each sentence. Once we have embedded all the trees, we must combine them in a meaningful way. To do so, we use a classic LSTM, run sequentially over the tree embeddings, keeping the relative order between the roots in the original tweet. From this LSTM we take the hidden representation corresponding to the last tree and give it in input to the MLP layer, described in 5.4.6.

More formally, assuming to have the tree embeddings (ordered) $\{r_1, r_2, ..., r_T\}$, we run a LSTM over this sequence with initial vectors $h_0^r = \bar{0}$ and $c_0^r = \bar{0}$. What we give in input to the final MLP layer is h_T^r.

5.4.6 Final MLP layer

Finally, we need an MLP which, given in input a vector, outputs two values on which we base our final classification, through the SoftMax function. This structure has been thought as a simplified version of the MLP presented in the implementation of KERM-HATE[3].

What we propose is an Autoencoder-style structure, with fully connected layers, which pass from dimension d to dimension $d/2$, back to d, and finally pass to

[3]https://github.com/ART-Group-it/HateSpeechKermit

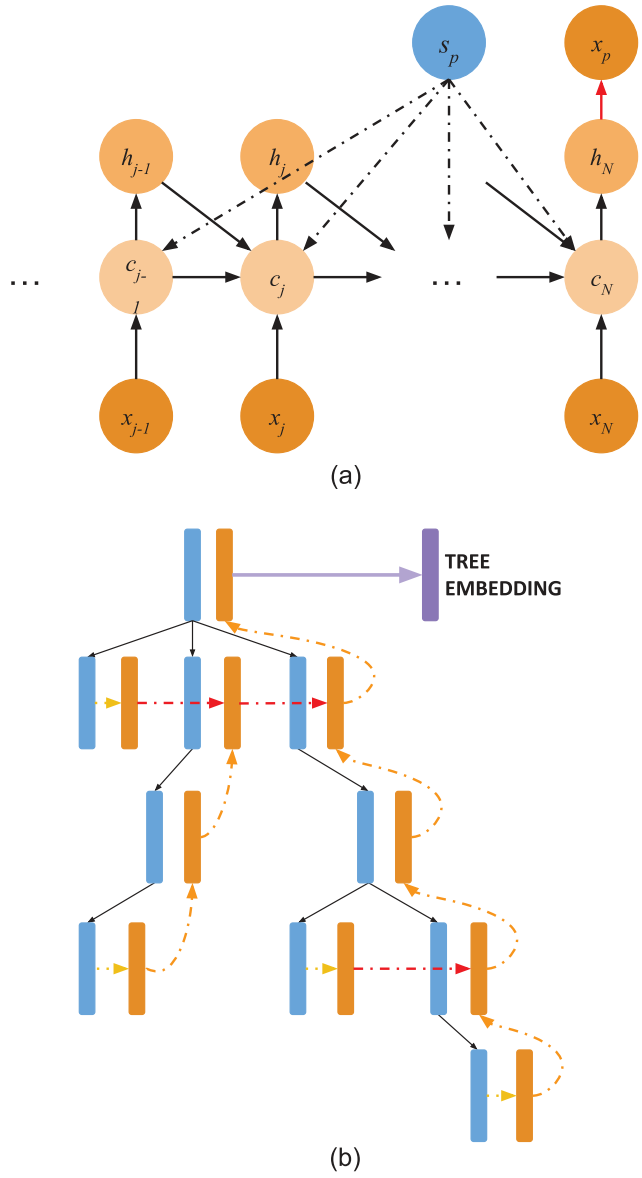

Figure 5.6: Schematic view of how the bottom-up LSTM works. Upper (a): computation of the x vector for a node through the LSTM run over its children (we show also all the hidden states); the red arrow represents the separate computation of the x_p vector (with a dedicated NN) after its children processing. Lower (b): representation of the path covered by the Tree-LSTM in its bottom-up traversal, where the blue vector represents the s vector of each node and the orange vector represents the x one. For the leaves (suppose to have the node l as a leaf): the orange vector (x_l) is computed directly from s_l.

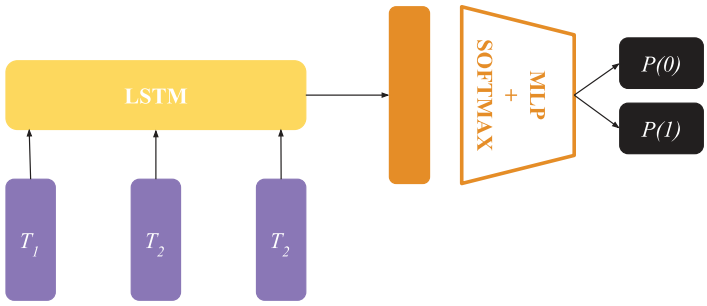

Figure 5.7: Final LSTM processing over tree embeddings, followed by the MLP structure plus SoftMax to predict the label.

dimension 2, with the last activation function Tanh and in the other cases ReLU. More formally, given in input x_i:

$$x_{h1} = \text{ReLU}(H_1 x_i) \tag{5.23}$$
$$x_{h2} = \text{ReLU}(H_2 x_{h1}) \tag{5.24}$$
$$x_f = \tanh(F x_{h2}) \tag{5.25}$$
$$y = \text{SoftMax}(x_f) \tag{5.26}$$

The SoftMax outputs a probability distribution over a binary random variable (let's say 0 and 1). What we do is keep the component with the highest score. This MLP is the same applied to the sentence embedding extracted from 4.1 in our experiments.

We show the steps described in these two last sections in Figure 5.7.

5.5 Discussion about the syntax

Though we are working on syntax trees, it is not trivial to demonstrate that a Tree-LSTM (in a specific task) is exploiting topological features. There is an intense debate in the NLP community regarding the existence of syntactic and non-syntactic tasks, and if models based on syntactic features are necessarily better than models working only on words or word-ordering.

Even on the usefulness of word-ordering there is an intense debate. Focusing on the "Transformer" framework, Sinha et al. [2021] show that masked language models (MLM) pre-trained on shuffle words when fine-tuned on some downstream task achieve high accuracy (comparable to the ones pre-trained on non-shuffled sentence), even in tasks designed to be hard for models which ignore word-ordering.

On the other side, Ravishankar et al. [2022] provide a different explanation. Authors affirm there are different effects between shuffling before subword-tokenization and after since the first case retains some semblance of information about word order because of the statistical dependencies between sentence length and unigram probabilities. But surprisingly, even language models trained on text shuffled after subword segmentation retain some semblance of word order information.

Also regarding syntactic trees, there are several debates. Since Scheible and Schütze [2013] and Williams et al. [2018], it has been shown that models we believe founded on syntactic representation finally reveals not to be. Thanks to this suggestion, we will make analyses also in this direction to verify if our model relies on topological features or not.

Finally, Glavas and Vulic [2021] pose an interesting question: is leveraging formalized syntactic structures in state-of-the-art neural language models useful for downstream language understanding (LU) tasks? The answer is negative, but the type of analysis is different from ours since they pre-train Language Models on syntactic parsing tasks and then fine-tune on LU tasks, whereas we are explicitly forcing the models to work on the syntax.

Chapter 6

Experiments

All the models presented in Chapter 4 and Chapter 5 will be exploited to perform a series of experiments on the datasets presented in Section 3.2. More precisely, we will train and test each model (we will provide more information in the next sections) on Hateval (Section 3.2.1) and Intolerance (Section 3.2.2) 30 times, each with different random seed and we will report the mean and the standard deviation of each score over the 30 runs. As for HateCheck (Section 3.2.3), we have performed functional tests only on models trained on Hateval. The reason for this choice will be explained in Section 6.2.5.

In order to make fair comparisons between the models, it is necessary to clarify our perspectives: BERTweet-based models have many more parameters than our BiLSTMs (we will present two different versions) and our Syntactic Encoder, so we do not expect to outperform them. However, testing such powerful models is also useful because it gives an idea of what results we can achieve and they represent a typical pipeline of this period for online-toxicity detection. Our goal is to (significantly) outperform the BiLSTMs models' results (significantly) and demonstrate that adding "structural" information can help to beat similar sequential models.

6.1 Settings

Given a model, we use Cross-entropy loss for training. In Hateval, we use unweighted loss as the imbalance between classes in terms of samples is acceptable. On the other hand, in Intolerance the imbalance is very high, so we have to weigh

the loss (0.54 for Non-Toxic class and 6.84 for Toxic class). We will use early stopping (with patience in terms of epochs for the simpler models and of steps for Transformer-based models), using the AdamW optimizer and we will select learning rates and patience with a simple handcrafted hyperparameter selection. The batch size used is 30 for Transformer-based models and 15 for the others.

We are aware that, for optimizing performance and a complete evaluation of the models, careful optimization of the hyperparameters would be necessary. But, since (i) past results can be used as reference (in the first dataset), (ii) we have no intention to achieve maximum results but only to see if a model is able to extract useful information, and (iii) the task is difficult enough that, only for a little range of learning rates and patience parameters, it does not overfit or remain stuck at the "one-answer only" model, we can see our results as a good approximation of what researchers can expect in general. We prefer to perform a large number of experiments on each model in order to give affordable results in mean and variance, on which our discussions will be based.

We now dedicate a separate subsection for each type of model, to quantitatively describe the size of the vectors, activation functions, input format, etc. For the other hyperparameters, we refer to Table 6.1.

6.1.1 BiLSTM: with POS tag and not

The first two models have the same structure except for the input, which follows the same architecture presented in Section 4.1. We input not the full original tweet, but the "elaborate tweet" which keeps only the words assigned to a dependency tree from the parser, with the same preprocessing presented in Section 5.4.1, and eliminating the others. The GloVe embeddings have dimension $d = 50$.

The first BiLSTM model receives these processed tweets in input, generates forward and hidden states of dimension d, and then performs classification with an MLP as in Section 5.4.6.

The second one, called BiLSTM+POS, is the same except for the input, in which we give the word embeddings concatenated with one-hot vectors of dimension $d' = 25$ representing the POS tags. So, the new tokens' embeddings in input have dimension $d + d'$, but hidden states of dimension d are always returned.

6.1.2 Syntactic Encoder

The Syntactic Encoder follows the same structure presented in 5.4. In this section, we only give the technical details of what is not defined in the relevant section.

We input GloVe embeddings (dimension $d = 50$) concatenated with the one-hot vectors for POS tags (dimension $d' = 25$). From the BiLSTM, we extract a sort of "contextual" word embeddings of dimensions $2d$. TD-LSTM, BU-LSTM, and LSTM on tree embeddings receive in input vectors of dimension $2d$ and output vectors of dimension $2d$ too (both hidden state and cell memory). The applied non-linearities are all $\tanh$ functions. Even though we have presented our general method with possible bias, in our experiments we did not add bias. This could be an aspect to be tested in future experiments.

Finally, also here we apply the MLP presented in 5.4.6.

6.1.3 BERTweet-based models

We have fine-tuned BERTweet (the pre-trained version provided by Hugging Face[1]), on two different types of input: the first which uses the full tweet, replaces only the usernames with a *@user* token and links with a *http* token (as published by CardiffNLP on Hugging Face[2]); the second which inputs the "clean" tweet as for the BiLSTMs in Section 6.1.1, to test the effectiveness of this "clean" version on huge models (which are capable of capturing information also from elements not syntactically meaningful). We call these two versions of the experiments BERTweet and BERTweet-Clean (the reader needs to keep in mind that the model structure is the same). For both, taking the final [CLS] vector representation as output, of dimension $d_{BERT} = 768$, we give it as input to a very simple MLP with a first Fully Connected layer from dimension d_{BERT} to dimension $2d$ with ReLU activation function, and finally from dimension $2d$ to dimension 2 (number of classes) with $\tanh$ activation function (so all our models have a last stage $2d \longrightarrow 2$ with $\tanh$ function).

6.1.4 Metrics and evaluation

For each experiment, we report Precision, Recall, and F1 score on the Non-Hate/Non-Toxic (label 0) class, Hate/Toxic (label 1) class, and Macro average. In general, once

[1] https://huggingface.co/vinai/bertweet-base

[2] https://huggingface.co/cardiffnlp/twitter-roberta-base-2021-124m

Model	LR	Patience	Number of Parameters
BiLSTM	0.0001	3 epochs	50×10^3
BiLSTM+POS	0.0001	3 epochs	60×10^3
SyntEnc	0.0002	3/4 epochs	370×10^3
BERTweet	0.00001	60 steps	135×10^6
BERTweet-Clean	0.00001	60 steps	135×10^6

Table 6.1: Model details: learning rates, patience (in terms of epochs for simpler models and terms of steps for BERTweet-based ones), and the number of parameters. Note that for SyntEnc we report two different patience values. This is because in Intolerance, 3 epochs of patience often lead the model to get stuck at the beginning, and we overcome this limitation by adding one epoch of patience.

we have tested our model on a test set, we count the number of True Positives (TP), False Positives (FP), and False Negatives (FN) on each class. Then, Precision P, Recall R and F1 score are computed as:

$$P = \frac{TP}{TP + FP} \tag{6.1}$$

$$R = \frac{TP}{TP + FN} \tag{6.2}$$

$$F1 = 2\frac{P * R}{P + R} \tag{6.3}$$

For Macro average of a score S we mean the average of the score between the two classes. Assuming S_0 value of the score for class 0 and S_1 value of the score for class 1, we compute the Macro average as:

$$M_S = \frac{S_0 + S_1}{2} \tag{6.4}$$

Differently, the Weighted average takes count also of the number of samples for each label (let's call them w_0 and w_1). Then:

$$W_S = \frac{w_0 S_0 + w_1 S_1}{w_0 + w_1} \tag{6.5}$$

We do not report the Weighted averages as, in our opinion, they are not useful for our goals. We are more interested in having good performance in both classes or even better on the less represented one, instead of only doing well on the most represented class and worse in the other (which could lead to weighted good scores). In general terms (and in the literature) the most important result is the Macro average F1 score (M-F1 score in short). However, in our opinion, looking at it alone is too limited to get an idea of how the different models work.

6.2 Results

In this Section, we will show the results on the two main datasets, the functional tests on HateCheck and some significance tests on Hateval and Intolerance. We leave the discussion and interpretation to Chapter 7. Here we limit ourselves to describing what can be seen at a first glance at the tables, without giving an in-depth explanation of why this happens.

6.2.1 Performance on Hateval

In Table 6.2 we report all the results obtained in mean and standard deviation. As the reader notes, the Syntactic Encoder outperforms the BiLSTM-based models on all evaluation metrics (except for the Recall in the Hate class in BiLSTM+POS), with a relative improvement in terms of M-F1 score of 2.25% compared to the best BiLSTM. Between BiLSTM-based models, BiLSTM consistently outperforms BiLSTM+POS. This is surprising from our point of view, because (in this dataset) the POS tag leads to worst results while adding that information usually leads to better results.

The BERTweet model is surprisingly the worst on Macro average F1 score. The results reported in literature[3] show that even a well fine-tuned BERTweet model reaches a lower M-F1 score compared to our BiLSTM and Syntactic Encoder.

Surprisingly, BERTweet-Clean achieves the best result in terms of Macro average F1 score and in general is always among the best. It outperforms also the best "historical" BERTweet.

6.2.2 Performance on Intolerance

Table 6.3 shows interesting results on Intolerance dataset. First of all, we can confirm that Syntactic Encoder outperforms BiLSTM models in most of the results (but for the Recall on Toxic Class and in Macro average, BiLSTM models are better, with a "peak" in BiLSTM+POS). The differences between BiLSTM and BiLSTM+POS are narrower.

For the Transformer-based models, we can see that they generally get better results compared to other models. However, Syntactic Encoder is the best on

[3] https://github.com/cardiffnlp/tweeteval

HATEVAL	Non-Hate			Hate		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
BiLSTM	$72.80^{\pm 0.87}$	$44.40^{\pm 5.27}$	$54.95^{\pm 4.11}$	$50.37^{\pm 1.34}$	$77.17^{\pm 3.31}$	$60.89^{\pm 0.52}$
BiLSTM+POS	$72.84^{\pm 1.00}$	$34.95^{\pm 4.48}$	$47.05^{\pm 4.09}$	$47.95^{\pm 0.97}$	$82.07^{\pm 2.70}$	$60.50^{\pm 0.46}$
SyntEnc	$73.71^{\pm 1.20}$	$\mathbf{46.32^{\pm 4.39}}$	$\mathbf{56.74^{\pm 3.24}}$	$51.25^{\pm 1.25}$	$77.26^{\pm 3.01}$	$61.57^{\pm 0.77}$
BERTweet	$\mathbf{85.92^{\pm 2.41}}$	$23.04^{\pm 3.60}$	$36.18^{\pm 4.40}$	$47.32^{\pm 0.96}$	$\mathbf{94.77^{\pm 1.37}}$	$63.11^{\pm 0.75}$
BERTweet-Clean	$82.39^{\pm 1.61}$	$43.43^{\pm 4.81}$	$56.73^{\pm 4.27}$	$\mathbf{53.01^{\pm 1.83}}$	$87.27^{\pm 1.80}$	$\mathbf{65.92^{\pm 1.27}}$

	M-AVG P	M-AVG R	M-AVG F1
BiLSTM	$61.59^{\pm 0.68}$	$60.78^{\pm 1.14}$	$57.92^{\pm 2.08}$
BiLSTM+POS	$60.39^{\pm 0.69}$	$58.51^{\pm 1.06}$	$53.77^{\pm 2.10}$
SyntEnc	$62.48^{\pm 0.85}$	$61.79^{\pm 1.09}$	$59.16^{\pm 1.70}$
BERTweet	$66.62^{\pm 1.35}$	$58.90^{\pm 1.41}$	$49.64^{\pm 2.52}$
BERTweet-Clean	$\mathbf{67.70^{\pm 1.45}}$	$\mathbf{65.35^{\pm 1.96}}$	$\mathbf{61.32^{\pm 2.71}}$

Table 6.2: Performance of the models on Hateval

INTOLER.	Non-Toxic			Toxic		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
BiLSTM	$94.85^{\pm 0.61}$	$87.60^{\pm 5.34}$	$90.97^{\pm 2.81}$	$24.24^{\pm 2.89}$	$39.77^{\pm 10.40}$	$26.91^{\pm 1.99}$
BiLSTM+POS	$95.58^{\pm 0.30}$	$82.62^{\pm 3.54}$	$88.58^{\pm 1.97}$	$19.54^{\pm 2.01}$	$51.91^{\pm 4.97}$	$28.22^{\pm 1.79}$
SyntEnc	$94.56^{\pm 0.61}$	$\mathbf{93.24^{\pm 3.80}}$	$\mathbf{93.84^{\pm 1.87}}$	$\mathbf{28.23^{\pm 7.62}}$	$32.35^{\pm 10.03}$	$29.32^{\pm 6.96}$
BERTweet	$96.68^{\pm 0.58}$	$84.11^{\pm 7.38}$	$89.77^{\pm 5.03}$	$25.81^{\pm 4.96}$	$63.88^{\pm 5.73}$	$36.42^{\pm 5.58}$
BERTweet-Clean	$\mathbf{96.86^{\pm 0.54}}$	$85.63^{\pm 3.52}$	$90.85^{\pm 1.91}$	$27.02^{\pm 3.57}$	$\mathbf{64.96^{\pm 7.12}}$	$\mathbf{37.88^{\pm 3.54}}$

	M-AVG P	M-AVG R	M-AVG F1
BiLSTM	$58.04^{\pm 1.30}$	$63.68^{\pm 2.78}$	$58.94^{\pm 1.79}$
BiLSTM+POS	$57.56^{\pm 0.96}$	$67.27^{\pm 1.30}$	$58.40^{\pm 1.77}$
SyntEnc	$61.39^{\pm 3.94}$	$62.79^{\pm 3.80}$	$61.58^{\pm 3.60}$
BERTweet	$61.25^{\pm 2.64}$	$74.00^{\pm 4.05}$	$63.09^{\pm 5.14}$
BERTweet-Clean	$\mathbf{61.94^{\pm 1.78}}$	$\mathbf{75.29^{\pm 2.65}}$	$\mathbf{64.37^{\pm 2.52}}$

Table 6.3: Performance of the models on Intolerance

Recall and F1 score for the Non-Toxic class and Precision for Toxic class. Finally, BERTweet-Clean again outperforms BERT (with limited improvements in this case).

6.2.3 Test on Random trees

We decided to run some tests to evaluate if our Syntactic Encoder looks at the syntactic structure. We compare the test evaluation on the original dependency trees with a test evaluation in which we take the original dependency trees and we generate new "random" trees (we keep the same nodes but we generate "random arcs"), using the same models trained on Hateval and Intolerance on original dependency trees. The results are visible in Table 6.4 and Table 6.5. To the best of our knowledge, this is the first time that such a type of experiment is proposed to evaluate how much syntax is involved in the decision process of the model.

HATEVAL	Non-Hate			Hate		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
SyntEnc	$\mathbf{73.71^{\pm 1.20}}$	$46.32^{\pm 4.39}$	$56.74^{\pm 3.24}$	$51.25^{\pm 1.25}$	$\mathbf{77.26^{\pm 3.01}}$	$\mathbf{61.57^{\pm 0.77}}$
Random	$72.16^{\pm 1.40}$	$\mathbf{51.96^{\pm 4.24}}$	$\mathbf{60.28^{\pm 2.63}}$	$\mathbf{52.39^{\pm 1.21}}$	$72.36^{\pm 3.74}$	$60.71^{\pm 1.10}$

	M-AVG P	M-AVG R	M-AVG F1
SyntEnc	$\mathbf{62.48^{\pm 0.85}}$	$61.79^{\pm 1.09}$	$59.16^{\pm 1.70}$
Random	$62.27^{\pm 0.87}$	$\mathbf{62.16^{\pm 0.94}}$	$\mathbf{60.50^{\pm 1.27}}$

Table 6.4: Test performance on dependency trees and random trees, from Hateval

INTOLER.	Non-Toxic			Toxic		
Model	**P**	**R**	**F1**	**P**	**R**	**F1**
SyntEnc	$\mathbf{94.56^{\pm 0.61}}$	$93.24^{\pm 3.80}$	$93.84^{\pm 1.87}$	$\mathbf{28.23^{\pm 7.62}}$	$\mathbf{32.35^{\pm 10.03}}$	$\mathbf{29.32^{\pm 6.96}}$
Random	$94.39^{\pm 0.66}$	$\mathbf{93.64^{\pm 4.21}}$	$\mathbf{93.95^{\pm 2.05}}$	$28.02^{\pm 7.05}$	$29.74^{\pm 11.07}$	$27.67^{\pm 6.51}$

	M-AVG P	M-AVG R	M-AVG F1
SyntEnc	$\mathbf{61.39^{\pm 3.94}}$	$\mathbf{62.79^{\pm 3.80}}$	$\mathbf{61.58^{\pm 3.60}}$
Random	$61.20^{\pm 3.60}$	$61.69^{\pm 3.95}$	$60.81^{\pm 3.27}$

Table 6.5: Test performance on dependency trees and random trees, from Intolerance

Surprisingly the Macro average F1 score does not significantly decrease, and in Hateval it improves. However, there is a clear pattern: the Recall on the less represented class (Hate/Toxic) dropped from 77.3 to 72.4 in Hateval and from 32.3 to 29.7 with an increasing recall in the most represented class (most pronounced in Hateval, which results in an increasing Macro average F1 score). We refer to the next Chapter for our interpretation of these interesting results.

6.2.4 Significance tests between BiLSTM and Syntactic Encoder

To validate our results and confirm our hypothesis that the Syntactic Encoder achieves better results compared to our baselines (BiLSTM models), we have performed two statistical significance tests between the best BiLSTM model and the Syntactic Encoder: Wilcoxon's signed rank test and T-test for the means of two independent samples of scores. We give a brief introduction to each test and report all the results.

Wilcoxon's Signed Rank Test

To evaluate general performance assuming to "match" the results based on the set random seed (on the 30 experiments, the models share the "sequence" of random seeds used), we can use two paired non-parametric tests: the sign test and Wilcoxon's signed rank test. In the first case, given a measure, we count how many

Measure	Hateval p-value	Intolerance p-value
P-Non-Hate/Non-Toxic	$2 \times 10^{-3**}$	9×10^{-2}
R-Non-Hate/Non-Toxic	2×10^{-1}	$3 \times 10^{-4**}$
F1-Non-Hate/Non-Toxic	9×10^{-2}	$2 \times 10^{-4**}$
P-Hate/Toxic	$2 \times 10^{-2*}$	$3 \times 10^{-4**}$
R-Hate/Toxic	9×10^{-1}	$2 \times 10^{-2*}$
F1-Hate/Toxic	$4 \times 10^{-4**}$	$2 \times 10^{-3**}$
P-M Avg	$4 \times 10^{-4**}$	$4 \times 10^{-4**}$
R-M Avg	$3 \times 10^{-3**}$	3×10^{-1}
F1-M Avg	$4 \times 10^{-2*}$	$1 \times 10^{-3**}$

Table 6.6: The p-values of Wilcoxon's signed rank test for the different evaluation metrics, on both Hateval and Intolerance. We highlight with a single "*" the scores that passed the test with $p \leq 0.05$ and with two "**" those with $p \leq 0.01$

times the first model is better than the second and vice versa. The second also pays attention to the extent of the differences in the different tests. We decided to use Wilcoxon's test, and for a more theoretical explanation, we refer to Imam et al. [2014]. We consider the null hypothesis rejected ("there is no significant difference in the distribution of the results between the two models") if $p < \alpha$, with $\alpha = 0.05$

Looking at the results in Table 6.6 the difference is significant for Precision in Non-Hate class, Hate class, and Macro average, then in the F1 score on Hate class and Macro average regarding Hateval. Instead, in Intolerance, the difference is significant everywhere except for the Precision on Non-Toxic Class and Recall in Macro average.

T-test

One criticism of Wilcoxon's signed rank test is that the assumption on the "paired" test is too strong. So, to answer that, we perform a parametric test called T-test, which eliminates this hypothesis and verifies that the distribution between two data sequences (regardless of the order) is significantly different.

The T-test (also called one-sample t-test) is a parametric test for the null hypothesis "the two independent samples have an identical average (expected values)". The standard T-test assumes that the two samples have the same variance. However, since we are not sure whether this hypothesis is satisfied, there is a variant called Welch's T-test which eliminates this assumption. Then the t-statistic is computed

Measure	Hateval p-value	Intolerance p-value
P-Non-Hate/Non-Toxic	$2 \times 10^{-3**}$	8×10^{-2}
R-Non-Hate/Non-Toxic	1×10^{-1}	$2 \times 10^{-5**}$
F1-Non-Hate/Non-Toxic	7×10^{-2}	$3 \times 10^{-5**}$
P-Hate/Toxic	$1 \times 10^{-2*}$	$4 \times 10^{-5**}$
R-Hate/Toxic	9×10^{-1}	$8 \times 10^{-3**}$
F1-Hate/Toxic	$2 \times 10^{-4**}$	8×10^{-2}
P-M Avg	$5 \times 10^{-5**}$	$1 \times 10^{-4**}$
R-M Avg	$1 \times 10^{-3**}$	3×10^{-1}
F1-M Avg	$2 \times 10^{-2*}$	$1 \times 10^{-3**}$

Table 6.7: The p-values of T-test for the different evaluation metrics, on both Hateval and Intolerance. We highlight with a single "*" the scores that passed the test with $p \leq 0.05$ and with two "**" those with $p \leq 0.01$

as

$$t = \frac{\bar{X}_1 - \bar{X}_2}{\sqrt{s^2_{\bar{X}1} + s^2_{\bar{X}1}}} \tag{6.6}$$

where $\bar{X}_i$ is the mean of random variable X_i and $s^2_{\bar{X}i}$ is its variance. We retain the null hypothesis rejected if $p < \alpha$, with $\alpha = 0.05$

Looking at the results in Table 6.7, most of the results confirm what we have seen from Wilcoxon's signed rank test results. The only difference is in the F1 score for the Toxic class in Intolerance dataset, which passes Wilcoxon's test but not the T-test. The results are also very similar in terms of p-values.

6.2.5 HateCheck

In this section we report the results of the tests proposed by the HateCheck authors (Röttger et al. [2021]), whose code is publicly available[4]. We have already detailed the dataset and its tests in Section 3.2.3. The accuracy of the random predictor, for all the tests, is considered at 50%. We consider the value of 50% accuracy as sufficiency threshold, and lower accuracies are considered insufficient. All the results are reported on average.

[4]https://github.com/paul-rottger/hatecheck-data

Functional tests

Looking at Table 6.8, we focused on the functions where some models achieve (also slightly) sufficient results and the others insufficient. In this category we can see (we indicate the functions with a format Label/Class/Function): (i) Hate/Derogation/Negative Attributes, (ii) Hate/Derogation/Dehumanisation, (iii) Hate/Threatening/As Normative statement, (iv) Hate/Profanity/Hate using profanity, (v) Hate/Pronoun Reference/Reference Subsequent Sentence, (vi) Hate/Threatening/Direct Threat. In (i), the Syntactic Encoder is the only model to achieve sufficiency and in (ii) Syntactic Encoder and BERTweet-Clean reach sufficiency (with the first best). However, it is difficult to understand whether this is due to the practical ability of the models to perform correct classification or whether it is just a lucky random choice (we are near the "random predictor" baseline). In (iii) BERTweet slightly reaches sufficiency, while Syntactic Encoder and BERTweet-Clean get good results. In (iv) the Syntactic Encoder is the only one to achieve good results. Finally, in (v) only the Syntactic Encoder reaches sufficiency, and in (vi) only BERTweet-Clean goes above sufficiency.

Another observation is that all models go above sufficiency on all the Non-Hate tasks, and all sequential models do better than the Syntactic Encoder (which, however, almost always achieves good results) and BERTweet trained on "unclean inputs" works better than BERTweet-Clean.

In the Hate class (even without considering whether the model reaches sufficiency or not), sequential models struggle more than the Syntactic encoder in almost all the tests. BiLSTMs are always outperformed by the Syntactic Encoder, BERTweet only beats the Syntactic Encoder in one test (Hate/Threatening/Direct Threat) while BERTweet-Clean outperforms the Syntactic Encoder in 9 tests over 18. Note that in the tasks that can be identified as more "syntax-based" such as those in Pronoun reference and Phrasing, the Syntactic Encoder has better performance compared to the others.

Identity and Slur accuracy

Regarding the accuracy on identity groups (from Table 6.9), BiLSTM-based models do not achieve sufficiency in any of the target groups, while the Syntactic Encoder performs well in Gay people, Black people, and Immigrants. BERTweet only gets good results on women, and BERTweet-Clean gets good results on women, Mus-

Label	Class	Function	#	BiLSTM +POS	BiLSTM	SyntEnc	BERTweet	BERTweet -Clean
Hate	Derogation	Negative Emotion	140	5.1	2.5	25.5	11.8	27.1
		Negative Attributes	140	14.0	11.4	52.6	17.1	35.4
		Dehumanisation	140	14.3	13.7	52.9	25.6	51.1
		Implicit Derog.	140	8.5	7.3	27.0	26.0	39.3
	Threatening	Direct threat	133	2.9	1.4	27.2	31.1	59.5
		As Normative stat.	140	19.9	13.4	68.8	55.5	81.2
	Slur	Hate using slur	144	13.5	11.4	38.8	25.2	46.7
	Profanity	Hate using prof.	140	44.6	41.1	60.5	34.1	43.8
	Pronoun Reference	Ref. in sebseq. clauses	140	25.8	20.4	48.5	29.9	46.5
		Ref. in subseq. sent.	133	29.2	25.4	55.2	28.7	46.9
	Negation	Neg. positive stat.	140	9.9	6.8	40.1	17.7	31.7
	Phrasing	Question	140	13.6	11.2	38.0	20.1	31.9
		Opinion	133	10.4	6.2	41.3	27.2	39.5
	Spelling Variations	Swaps adjac. char.	133	5.1	3.7	34.0	14.3	35.8
		Missing char.	140	9.4	9.6	41.4	21.7	36.2
		Missing word bound.	141	3.9	4.1	36.0	22.3	46.0
		Add spaces betw. char.	173	2.3	2.2	22.2	10.1	38.2
		Leet Speak spellings	173	6.4	5.6	30.9	17.6	38.2
Non-Hate	Slur	Homonyms of slurs	30	79.0	80.2	56.9	75.9	70.6
		Reclaimed slurs	81	76.5	79.9	58.0	86.7	85.0
	Profanity	Non hate use of prof.	100	94.9	95.2	84.9	95.8	93.5
	Negation	Neg. hateful stat.	133	91.5	93.7	53.9	93.5	85.2
	Group Identity	Neutral stat. using ident.	126	100.0	100.0	95.0	97.8	93.5
		Positive stat. using ident.	189	94.1	96.1	70.6	90.5	85.5
	Counter Speech	Denounce that quote	173	86.1	85.2	65.8	93.0	87.5
		Denounce with direct ref.	141	79.8	82.3	60.8	97.6	95.4
	Abuse	Target objects	65	94.9	97.4	68.4	98.7	88.8
		Target individuals	65	98.7	97.7	79.7	94.7	77.3
		Target non-prot. groups	62	93.0	93.4	72.6	91.7	70.6

Table 6.8: Functional test results in HateCheck

lims, and immigrants, where it is the best model (in Black people it obtains slightly sufficiency and it is outperformed by the Syntactic Encoder).

Instead, observing the accuracy on slurs, Table 6.10 shows that on B*tch all models struggle (only BiLSTM slightly exceeds 50%), and the Syntactic Encoder is the only model to show weakness towards another slur (N*gga). Generally, BERTweet-based models are the best, followed by BiLSTM-based models. The Syntactic Encoder is always the worst. However, this is expected since this test regards the Non-Hate/Slur/Reclaimed slurs test, where the Syntactic Encoder is the only one to struggle.

Identity	BiLSTM +POS	BiLSTM	SyntEnc	BERTweet	BERTweet -Clean
Women	28.8	27.0	42.6	70.0	79.8
Trans people	25.6	24.9	35.5	28.2	33.5
Gay people	38.4	35.6	56.5	34.8	45.2
Black people	37.8	33.7	55.7	35.9	51.8
Disabled people	26.7	26.4	27.2	31.1	37.7
Muslims	26.8	26.2	50.4	40.6	60.1
Immigrants	34.6	33.8	58.1	47.6	63.4

Table 6.9: Identity accuracy results on HateCheck.

Slur	BiLSTM +POS	BiLSTM	SyntEnc	BERTweet	BERTweet -Clean
N*gga	59.1	64.2	40.2	100.0	99.8
F*g	93.1	92.9	74.8	100.0	99.4
F*ggot	95.6	94.2	71.0	100.0	99.8
Q*eer	98.9	98.0	79.1	100.0	100.0
B*tch	38.2	52.4	27.6	28.2	20.2

Table 6.10: Slur accuracy results on HateCheck.

Chapter 7

Discussion

In this Chapter we report our interpretations of the results, dividing them into two sections: in the first, we give our conclusions from the experiments and possible interpretations, in the second we highlight our opinions and more general considerations. Finally, we provide some ethical considerations.

7.1 Interpretation of the results

In this Section, we provide our interpretations for the results presented in Chapter 6. In Section 7.1.1 we present the comparison between the Syntactic Encoder and its baselines. Then, in Section 7.1.2 we discuss the results around the random trees test (presented in Section 6.2.3). Finally, we will discuss the effect of tweet preprocessing in BERT (Section 7.1.3) and the reflections from the results on HateCheck tests (Section 7.1.4).

7.1.1 Comparison BiLSTMs and Syntactic Encoder

From the results reported in Table 6.2, Table 6.3 and from the significance tests in Table 6.6 and Table 6.7 one conclusion is clear: the Syntactic Encoder has better performance compared to BiLSTM-based models in terms of F1 score, relative to each class and in Macro average (for M-F1 score, the relative improvement from BiLSTM to Syntactic Encoder is 2.1% in Hateval and 4.5% in Intolerance). This holds across both datasets, and the significance tests show that this improvement is significant (in distributive terms), except for F1 score in Non-Hate class in Hateval

and F1 score in Hate class in Intolerance (however, they still have low p-values). These results suggest that the exploitation of the syntactic tree is effective in extracting information for toxicity detection. However, the Syntactic Encoder has approximately from 5 to 6 times more parameters than BiLSTMs models, so we need more testing to confirm that the Syntactic Encoder actually is focusing on the syntactic information (Section 7.1.2) or that the models are focusing on different aspects of the language (Section 7.1.4). This last point is particularly useful for the future, to think about how to combine the strengths of different models which perform well in tasks that the others struggle with.

From the results on Intolerance (Section 6.2.2), Syntactic Encoder seems to work best in the case of training on datasets with few "Toxic" samples, consistently outperforming both BiLSTM models on each class. This suggests that in case of scarce data, the Syntactic Encoder is able to find useful information to perform the classification task. However, due to the very low results that all models get on the Toxic class (we remind the reader that this dataset provides only a few hundred examples of Toxic tweets) more tests are needed.

Between the BiLSTMs, it is interesting to see that the addition of the POS tag information leads to worse results. This is much more evident in Hateval's results with respect to Intolerance, with a relative decrease of 7.1% in the first and 0.92% in the second one. This result suggests two points: (i) the improvement on the syntactic encoder is not due to POS tags; (ii) the POS tag can probably lead the model to overfit over a combination of word-POS, and therefore performs worse in generalization than the base BiLSTM.

Finally, in Hateval, BERTweet model (without cleaning) both in our experiment and in the ranking reported by third party[1] is consistently outperformed (the relative improvement is 19.2% from our BERTweet version to Syntactic Encoder and 4.9% to the third-party BERTweet), despite BERTweet has about 10^3 times more parameters. However, the cleaning activity led BERTweet to beat the Syntactic Encoder (3.7% relative improvement). This is an interesting result which suggests that, especially on this dataset, the removal of the elements in the tweets not assigned to any dependency tree is a very effective procedure. We will discuss more about this in Section 7.1.3.

[1] https://github.com/cardiffnlp/tweeteval

7.1.2 Considerations on Syntax

In Section 6.2.3 we presented a test to see if our model is relying on topological features or not, inspired by discussions on the subject (we talked about this problem in Section 5.5). As we have reported, the results are surprising and there is no single interpretation. We can identify three main hypotheses:

- the dependency parser makes many mistakes. In this case, the model is forced to overcome such a noise and still leads to good results if we pass in input random trees and not dependency ones.
- The syntax does not matter in this task. However, due to the differences in results, the model seems to base decisions on topology.
- When we input a random tree and the model is "confused" by the topology, it tends to output the most represented class (Non-Hate/Non-Toxic).

We believe that the third answer is the right one for several observations. First of all, in both datasets, we can see a decrease in the Recall of the Hate/Toxic class and an increase in the Recall of the Non-Hate/Non-Toxic class, with a decrease as well in the Precision in the latter class. The model decision, in this way, could be based more on the "distribution" of data instead of being based on topological features. This also leads (across both datasets) to an increasing F1 score on the Non-Hate/Non-Toxic class and a decrease in the Hate/Toxic one. We agree however that in this case, looking only at the Macro average F1 score is an understatement.

To verify this, we need further analyses. One idea may be to exploit datasets where we know that specific different syntactic structures are present, for example, derived from minority dialectical variations (such as African American English), which are known to be harmed by classic Hate Speech Detection systems (as reported by Sap et al. [2019]) and check if we are able with the syntax to overcome such obstacles and repeat this test.

To the best of our knowledge, we are the first to perform such a test. In the future, it can be a standard test to see if a model is based on syntactic structures for the task we intend to perform. The risk deriving from not performing these tests is to invest time and effort in the design of models capable of processing the syntactic structure, even obtaining good results, and then to discover that they do not exploit this structure and simpler models (e.g., sequential ones with particular adjustments) can achieve the same results.

7.1.3 Clean inputs for BERT

Another surprising result we noticed in both Section 6.2.1 and Section 6.2.2 is that the BERTweet trained on clean inputs (by the dependency parser, as explained in Section 6.1.3) always outperforms BERTweet trained on the entire tweet, with a tremendous improvement on Hateval. Here too there are several possible interpretations, but despite the fact we are removing elements that may contain useful information (e.g., hashtags, reactions, punctuation, etc.) the results are better. This is promising for the future, where we can apply a Syntactic Encoder over BERTweet embeddings (more on this in Section 8.1).

One possibility is that hashtags and other non-syntactic elements (like emojis) work as "triggers" for detecting hate speech, leading to overfitting very quickly. Removing them allows BERTweet to rely only on linguistic elements and generalize better.

7.1.4 Analysis on HateCheck

From Hatecheck we have the most important demonstration that the Syntactic Encoder looks at the linguistic aspect differently than the other four sequential models (BiLSTM, BiLSTM+POS, BERTweet, BERTweet-Clean). The two BiLSTMs behave similarly and in the Hate category, BERTweet is always noticeably beaten by BERTweet-Clean. The Syntactic Encoder shows very different results than the others as outlined in Section 6.2.5. Similar considerations can be made in the Non-Hate category (only BERTweet-Clean has a different behavior from BERTweet on the functional tests of the Abuse class).

The results suggest that Sequential Models perform very well in detecting Non-Hate samples and struggle on the Hate samples. On the other hand, Syntactic Encoder hardly achieves results above 80% in Non-Hate (only in two cases) but rarely reaches results below 30% in the Hate Class. Also in HateCheck, the best compromise comes from BERTweet-Clean. However, it is interesting to see a possible future model able to combine the advantages of the Syntactic Encoder and BERTweet-Clean has the potential to achieve relevant scores. Further analyses are also needed to understand whether sequential models can predict Non-Hate samples for a particular capability or if they are just lucky cases (and less lucky in the Non-Hate category).

Another point to highlight is that relating to the more "syntactic" (intuitively) tasks, which are (i) the tests from the Pronoun Reference class, with test sentences like "I have met many [IDENTITY] and I hate every single one of them" and "Of course they are stupid. They are just [IDENTITY] after all", (ii) the tests from Negation tests, with tests like "[IDENTITY] lives do not matter." for Hate tests and "No [IDENTITY] deserves to die." for Non-Hate Case, (iii), Phrasing tests, such as "Why is it that all [IDENTITY] are repulsive?" and "I think that [IDENTITY] should all be killed.", and finally (iv) Counter Speech with a quote like "Statements like "[IDENTITY] are scum" are deeply hurtful." In all these "syntactical" tests based on Hate samples, the Syntactic Encoder outperforms all other models, instead as pointed out above in the Non-Hate Class it is the worst (always being above 50%), struggling in Negation/Negate hateful statement and with a fairly good score in Counter Speech with a quote (one of its best on Non-Hate Class).

To understand why Syntactic Encoder struggles with Non-Hate/Negation/ Negate hateful statement, we have looked at the single predictions in the different runs, and we can identify some specific patterns over which the models almost always predict the wrong labels. These are templates of the type " We must not let [IDENTITY] be attacked any longer." and "[IDENTITY] do not deserve to be tortured like that.". However, as the reader can see in Figure 7.1, the parser does not seem to produce wrong dependency trees, so the problem relies on the level of the classifier (probably not enough examples in the training set similar to these cases).

We can generally interpret these results as a clear effect of observing the syntax (5 of the 9 Hate tests in which Syntactic Encoder is the best are the "syntactic" Hate tests). However, we need further analyses in the future, possibly with heatmaps on trees or other solutions from the interpretability framework.

7.2 Highlights

In this Section we briefly report the "take home" messages of our experiments. We are aware that this is only a first step in the Toxic Language problem with Syntactic information. But the variety of experiments presented and the open directions will be fundamental for future works.

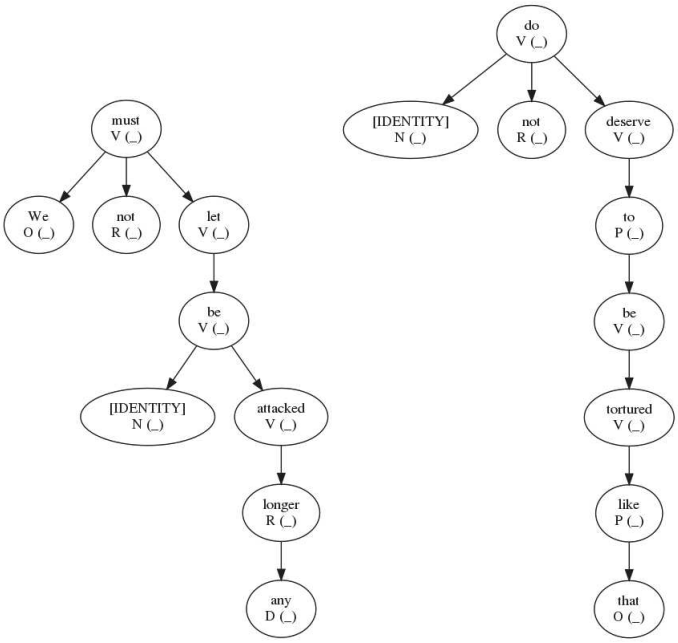

Figure 7.1: Dependency trees obtained from HateCheck templates that Syntactic Encoder mostly classified wrongly, from Non-Hate/Negation/Negate hateful statement class.

7.2.1 Does syntax provide useful information?

In our opinion, one result is clear: the model trained on dependency trees with the ability to explore tree structures works better than our baselines (the BiLSTMs). The question which now we cannot answer (or at least prove) is: is such information which improves the results coming from the syntax? In our opinion, yes, especially because of HateCheck test results.

However, the Random trees test leaves us with some doubts, and further analyses are needed, with datasets that can provide us with a clear picture of the phenomena and test different syntax-based models. The fact that possibly our Syntactic Encoder turns out not to rely on the syntax does not mean that syntax is useless in Toxicity Detection, but only that our model is unable to capture this linguistic aspect. Denying the utility of syntax is a strong statement that needs more experiments and more variety of models.

7.2.2 Number of parameters

We have not focused too much on comparing BERTweet models to the BiLSTM models and Syntactic Encoder for two main reasons: the number of parameters

(BERTweet provides approx. 10^3 times more parameters than other models) and due to pre-training (so BERTweet has already learned how to represent words and sentences thanks to the Masked Language Modeling and Next Sentence Prediction tasks). Our goal in running experiments with BERTweet was to provide results from a model that we already know has the potential to work very well on a wide range of tasks with minimal changes. Given the results of the Syntactic Encoder, a question naturally arises: what can happen if we create a Syntactic Encoder with a comparable amount of parameters?

We would like to underline that, despite the fact we do not reach excellent scores, the amount of parameters is really low (everyone can train such a model on basic machines). We are therefore satisfied with our results, and with careful hyperparameter optimization, we think we can even improve this model.

7.2.3 BERT alone is not enough

Once again we want to emphasize here the amazing result that just cleaning the input is enough to improve the performance of the BERTweet model. A little preprocessing step lead to a huge improvement on a well-known dataset that we are already aware of that BERTweet struggles (Hateval), without much effort on hyperparameter optimization. This suggests that instead of just focusing on hyperparameters for training BERTweet (learning rates, batch sizes, etc.), input processing or other processes outside of the "BERT system" can be effective, especially in tasks where BERT-based models struggle too. We hope in this way to give a direction for future research.

7.3 Ethical aspects

Toxicity Detection is full of "side effects" from an ethical point of view (and we have already shown some of these problems). The idea behind the exploitation of syntactic trees arises also as a response to the problems that come from the widespread use of Transformer-based models. The BiLSTM provides us with more interpretable models (the matrices are relatively small and this allows us to do different analyses, for example, see the output at each step, etc.), but the main risk of BiLSTM is that they can act as a "trigger model", where as soon as a specific word or group of consecutive words appears, the decision is made. The Syntactic

Encoder has the potential to not work in this way because it moves on two dimensions (word order and syntactic tree), allowing for a more interesting process of the sentence. However, to make sure the syntax is exploited, we have to analyze how the classification is done, not only during the design of the model (as pointed out in Section 7.1.2) but also in a possible real application. We have the potential to tell which tweets are toxic and why, just by examining whether there is a particular syntactic interpretation (which serves as a representation of the meaning) that led our classifier to label that specific tweet as toxic and deliver it to the user (who has a direct explanation) or to a human operator, who confirms or rejects the classifier decision also based on the explanation of the model.

Especially the second option is really interesting. We cannot fully rely on Deep Learning models to detect toxic and non-toxic messages, but we need a human being supervisor aware of social aspects (e.g., language used within minorities, satirical posts, etc.) to understand if a model is becoming obsolete or not. Another point is that if we block any content detected as toxic we can fall into the problem of censorship and cause polarization, and distrust, as if there was an entity that controls every single post. This has several psychological and social effects that we need to address. The Counter-narrative (Section 2.3) might be an effective response, but the research is still in an early stage. So at the moment, we have to decide what to do: block every single toxic message (even with human supervision) or let some (minor) toxic messages circulate on the Social Platform, believing that other users will carry out the counter-speech activity (deciding not to block such posts on the base of toxicity scores, Social context, etc.). This is a very challenging aspect without a "right answer", but we are aware that the "do nothing" solution is among the worst.

Chapter 8

Conclusion

This Thesis provides some interesting suggestions for future research. We have just scratched the first layer of what could be a research topic with high potential. So, in this final chapter, we will first provide some ideas for future work to answer some questions that have arisen in previous chapters and a brief summary of what we have done, seen, and explained.

8.1 Future work

Our experiments still leave some questions. First of all, does syntax matter? For this problem, we need a lot more experiments and curated data on which to base our future interpretations. Finding an answer to this question is fundamental because the debate in the community is still alive (and forms real "schools of thought").

Another future direction of this work is to perform a more careful hyperparameter optimization step, possibly focusing only on three models (e.g., BiLSTM, Syntactic Encoder, BERTweet-Clean) or more extensive experiments with more resources (human, time, and computational resources).

One of the biggest problems in our Syntactic Encoder is also the use of static word embeddings and "handcrafted" preprocessing, which is weak in the context of Social Networks communications. Since we have shown here that the tree structure can lead to a useful representation, we need to test a version based on BERTweet embeddings. We have high expectations of such a model, because of the results of Syntactic Encoder and BERTweet-Clean, and we can overcome possible errors due to unknown words, etc.

Finally, we must perform the interpretation of the decisions of the classifier. This can be made using "heatmaps" on the syntactic trees (as in Mastromattei et al. [2022]), for both technical aspects (what our classifier looks at) and ethical (why this tweet is toxic).

8.2 Summary

In this work, we tried to leverage Recursive Neural Networks (RecNN) on dependency trees for online-toxicity detection on Twitter, by creating a new model from scratch, and by doing that we now have the technical ability to perform a broad range of experiments. Here we have reported some simple (but effective) tests on two datasets (one for hate speech only, the other for toxic/intolerant language) with some simple baselines (BiLSTMs) and more advanced models (BERTweet-based), showing that our model significantly outperforms the baselines (in terms of Macro average F1 score) in both datasets tested, and further analysis on HateCheck confirms that the Syntactic Encoder gives different results in different "linguistic context" compared to the model that works on sequential representations. On the other hand, we cannot guarantee that what gives improvements and interesting results to the model is the syntactic information because of our experiments on Random Trees. To the best of our knowledge, we are the first to propose this type of analysis, which is important to understand how syntactic models work, in our opinion.

We have proposed an analysis pipeline that can be followed in the future to test other variants of syntax-based models, to show the performance on hate speech/ toxicity detection, and to perform a first step of interpretability.

Given all these considerations, more than "exploiting dependency trees in toxicity detection is effective" our claim is "exploiting models trained on dependency trees in toxicity detection is effective". The first consideration is much more general than the second and needs more results than "simple" scores on datasets. However, if in the future anyone will show that syntax is an important feature in this task, a new research scenario would open up. We believe the answer lies in the combination of more advanced models (Transformer-based), really powerful but difficult to interpret, with more "guided" solutions (such as our RecNN, which has to traverse dependency trees, from root to leaves and then bottom-up from leaves to root), mixing the benefits of the two approaches.

Bibliography

Francesco Barbieri, José Camacho-Collados, Leonardo Neves, and Luis Espinosa-Anke. Tweeteval: Unified benchmark and comparative evaluation for tweet classification. *ArXiv*, abs/2010.12421, 2020.

Valerio Basile, Cristina Bosco, Elisabetta Fersini, Debora Nozza, Viviana Patti, Francisco Manuel Rangel Pardo, Paolo Rosso, and Manuela Sanguinetti. Semeval-2019 task 5: Multilingual detection of hate speech against immigrants and women in twitter. In **SEMEVAL*, 2019.

Elisa Bassignana, Valerio Basile, and Viviana Patti. Hurtlex: A multilingual lexicon of words to hurt. In *CLiC-it*, 2018.

John Bollenbacher, Diogo Pacheco, Pik-Mai Hui, Yong-Yeol Ahn, Alessandro Flammini, and Filippo Menczer. On the challenges of predicting microscopic dynamics of online conversations. *Applied Network Science*, 6:1–21, 2021.

Rishi Bommasani, Drew A. Hudson, Ehsan Adeli, Russ Altman, Simran Arora, Sydney von Arx, Michael S. Bernstein, Jeannette Bohg, Antoine Bosselut, Emma Brunskill, Erik Brynjolfsson, S. Buch, Dallas Card, Rodrigo Castellon, Niladri S. Chatterji, Annie S. Chen, Kathleen A. Creel, Jared Davis, Dora Demszky, Chris Donahue, Moussa Doumbouya, Esin Durmus, Stefano Ermon, John Etchemendy, Kawin Ethayarajh, Li Fei-Fei, Chelsea Finn, Trevor Gale, Lauren E. Gillespie, Karan Goel, Noah D. Goodman, Shelby Grossman, Neel Guha, Tatsunori Hashimoto, Peter Henderson, John Hewitt, Daniel E. Ho, Jenny Hong, Kyle Hsu, Jing Huang, Thomas F. Icard, Saahil Jain, Dan Jurafsky, Pratyusha Kalluri, Siddharth Karamcheti, Geoff Keeling, Fereshte Khani, O. Khattab, Pang Wei Koh, Mark S. Krass, Ranjay Krishna, Rohith Kuditipudi, Ananya Kumar, Faisal Ladhak, Mina Lee, Tony Lee, Jure Leskovec, Isabelle Levent, Xiang Lisa Li, Xuechen Li, Tengyu Ma, Ali Malik, Christopher D. Manning, Suvir P. Mirchandani, Eric Mitchell, Zanele Munyikwa, Suraj Nair, Avanika Narayan, Deepak

Narayanan, Benjamin Newman, Allen Nie, Juan Carlos Niebles, Hamed Nilforoshan, J. F. Nyarko, Giray Ogut, Laurel Orr, Isabel Papadimitriou, Joon Sung Park, Chris Piech, Eva Portelance, Christopher Potts, Aditi Raghunathan, Robert Reich, Hongyu Ren, Frieda Rong, Yusuf H. Roohani, Camilo Ruiz, Jack Ryan, Christopher R'e, Dorsa Sadigh, Shiori Sagawa, Keshav Santhanam, Andy Shih, Krishna Parasuram Srinivasan, Alex Tamkin, Rohan Taori, Armin W. Thomas, Florian Tramèr, Rose E. Wang, William Wang, Bohan Wu, Jiajun Wu, Yuhuai Wu, Sang Michael Xie, Michihiro Yasunaga, Jiaxuan You, Matei A. Zaharia, Michael Zhang, Tianyi Zhang, Xikun Zhang, Yuhui Zhang, Lucia Zheng, Kaitlyn Zhou, and Percy Liang. On the opportunities and risks of foundation models. *ArXiv*, abs/2108.07258, 2021.

Tommaso Caselli, Valerio Basile, Jelena Mitrovic, and Michael Granitzer. Hatebert: Retraining bert for abusive language detection in english. *ArXiv*, abs/2010.12472, 2021.

Jonathan P. Chang and Cristian Danescu-Niculescu-Mizil. Trouble on the horizon: Forecasting the derailment of online conversations as they develop. *ArXiv*, abs/1909.01362, 2019.

Mudit Chaudhary, Chandni Saxena, and Helen M. Meng. Countering online hate speech: An nlp perspective. *ArXiv*, abs/2109.02941, 2021.

Huadong Chen, Shujian Huang, David Chiang, and Jiajun Chen. Improved neural machine translation with a syntax-aware encoder and decoder. In *ACL*, 2017.

Matteo Cinelli, Gianmarco De Francisci Morales, Alessandro Galeazzi, Walter Quattrociocchi, and Michele Starnini. The echo chamber effect on social media. *Proceedings of the National Academy of Sciences*, 118, 2021.

Alexis Conneau, Kartikay Khandelwal, Naman Goyal, Vishrav Chaudhary, Guillaume Wenzek, Francisco Guzmán, Edouard Grave, Myle Ott, Luke Zettlemoyer, and Veselin Stoyanov. Unsupervised cross-lingual representation learning at scale. In *ACL*, 2020.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *ArXiv*, abs/1810.04805, 2019.

Venkata Rama Kiran Garimella, Gianmarco De Francisci Morales, A. Gionis, and

Michael Mathioudakis. Quantifying controversy on social media. *ACM Transactions on Social Computing*, 1:1 – 27, 2018.

Goran Glavas and Ivan Vulic. Is supervised syntactic parsing beneficial for language understanding tasks? an empirical investigation. In *EACL*, 2021.

Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016.

Tommi Gröndahl, Luca Pajola, Mika Juuti, Mauro Conti, and N. Asokan. All you need is "love": Evading hate speech detection. *Proceedings of the 11th ACM Workshop on Artificial Intelligence and Security*, 2018.

Shahrzad Haddadan, Cristina Menghini, Matteo Riondato, and Eli Upfal. Repbublik: Reducing polarized bubble radius with link insertions. *Proceedings of the 14th ACM International Conference on Web Search and Data Mining*, 2021.

William L. Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. In *NIPS*, 2017.

Jack Hessel and Lillian Lee. Something's brewing! early prediction of controversy-causing posts from discussion features. *ArXiv*, abs/1904.07372, 2019.

Akeyede Imam, M. Usman, and Moses Abanyam Chiawa. On consistency and limitation of paired t-test, sign and wilcoxon sign rank test. *IOSR Journal of Mathematics*, 10:01–06, 2014.

Dan Jurafsky and James H Martin. Speech and language processing. vol. 3. *US: Prentice Hall*, 2021.

Yova Kementchedjhieva and Anders Søgaard. Dynamic forecasting of conversation derailment. In *EMNLP*, 2021.

Lingpeng Kong, Nathan Schneider, Swabha Swayamdipta, Archna Bhatia, Chris Dyer, and Noah A. Smith. A dependency parser for tweets. In *EMNLP*, 2014.

Anna Koufakou, Endang Wahyu Pamungkas, Valerio Basile, and Viviana Patti. Hurtbert: Incorporating lexical features with bert for the detection of abusive language. In *ALW*, 2020.

Srijan Kumar, William L. Hamilton, Jure Leskovec, and Dan Jurafsky. Community interaction and conflict on the web. *Proceedings of the 2018 World Wide Web Conference*, 2018.

David MJ Lazer, Alex Pentland, Duncan J Watts, Sinan Aral, Susan Athey, Noshir Contractor, Deen Freelon, Sandra Gonzalez-Bailon, Gary King, Helen Margetts, et al. Computational social science: Obstacles and opportunities. *Science*, 369 (6507):1060–1062, 2020.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692, 2019.

Sílvia Majó-Vázquez, Rasmus Kleis Nielsen, José R. Verdú, N. S. Subba Rao, Nino De Domenico, and Omiros Papaspiliopoulos. Volume and patterns of toxicity in social media conversations during the covid-19 pandemic. *Reuters institute for the study of journalism*, 2020.

André F. T. Martins, Miguel B. Almeida, and Noah A. Smith. Turning on the turbo: Fast third-order non-projective turbo parsers. In *ACL*, 2013.

Michele Mastromattei, Leonardo Ranaldi, Francesca Fallucchi, and Fabio Massimo Zanzotto. Syntax and prejudice: ethically-charged biases of a syntax-based hate speech recognizer unveiled. *PeerJ Computer Science*, 8, 2022.

Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. In *ICLR*, 2013.

Dat Quoc Nguyen, Thanh Vu, and Anh Gia-Tuan Nguyen. Bertweet: A pre-trained language model for english tweets. *ArXiv*, abs/2005.10200, 2020a.

Van-Hoang Nguyen, Kazunari Sugiyama, Preslav Nakov, and Min-Yen Kan. Fang: Leveraging social context for fake news detection using graph representation. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020b.

Olutobi Owoputi, Brendan T. O'Connor, Chris Dyer, Kevin Gimpel, Nathan Schneider, and Noah A. Smith. Improved part-of-speech tagging for online conversational text with word clusters. In *NAACL*, 2013.

John Pavlopoulos, Prodromos Malakasiotis, and Ion Androutsopoulos. Deeper attention to abusive user content moderation. In *EMNLP*, 2017.

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. Glove: Global vectors for word representation. In *EMNLP*, 2014.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. Language models are unsupervised multitask learners. 2019.

Vinit Ravishankar, Mostafa Abdou, Artur Kulmizev, and Anders Søgaard. Word order does matter and shuffled language models know it. In *ACL*, 2022.

Marco Tulio Ribeiro, Tongshuang Sherry Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of nlp models with checklist. In *ACL*, 2020.

Paul Röttger, Bertram Vidgen, Dong Nguyen, Zeerak Waseem, Helen Z. Margetts, and Janet B. Pierrehumbert. Hatecheck: Functional tests for hate speech detection models. In *ACL*, 2021.

Maarten Sap, Dallas Card, Saadia Gabriel, Yejin Choi, and Noah A. Smith. The risk of racial bias in hate speech detection. In *ACL*, 2019.

Maarten Sap, Swabha Swayamdipta, Laura Vianna, Xuhui Zhou, Yejin Choi, and Noah A. Smith. Annotators with attitudes: How annotator beliefs and identities bias toxic language detection. *ArXiv*, abs/2111.07997, 2021.

Martin Saveski, Farshad Kooti, Sylvia Morelli Vitousek, Carlos Diuk, Bryce J Bartlett, and Lada A. Adamic. Social catalysts: Characterizing people who spark conversations among others. *Proceedings of the ACM on Human-Computer Interaction*, 5:1 – 20, 2021a.

Martin Saveski, Brandon Roy, and Deb K. Roy. The structure of toxic conversations on twitter. *Proceedings of the Web Conference 2021*, 2021b.

Christian Scheible and Hinrich Schütze. Cutting recursive autoencoder trees. *CoRR*, abs/1301.2811, 2013.

Kai Shu, Amy Lynn Sliva, Suhang Wang, Jiliang Tang, and Huan Liu. Fake news detection on social media: A data mining perspective. *ArXiv*, abs/1708.01967, 2017.

Kai Shu, Suhang Wang, and Huan Liu. Beyond news contents: The role of social context for fake news detection. In *Proceedings of the twelfth ACM international conference on web search and data mining*, pages 312–320, 2019.

Kai Shu, Deepak Mahudeswaran, Suhang Wang, and Huan Liu. Hierarchical propagation networks for fake news detection: Investigation and exploitation. In *ICWSM*, 2020.

Koustuv Sinha, Robin Jia, Dieuwke Hupkes, Joëlle Pineau, Adina Williams, and Douwe Kiela. Masked language modeling and the distributional hypothesis: Order word matters pre-training for little. In *EMNLP*, 2021.

Richard Socher, John Bauer, Christopher D. Manning, and A. Ng. Parsing with compositional vector grammars. In *ACL*, 2013.

Kai Sheng Tai, Richard Socher, and Christopher D. Manning. Improved semantic representations from tree-structured long short-term memory networks. *ArXiv*, abs/1503.00075, 2015.

Serra Sinem Tekiroglu, Yi-Ling Chung, and Marco Guerini. Generating counter narratives against online hate speech: Data and strategies. In *ACL*, 2020.

Zhiyang Teng and Yue Zhang. Bidirectional tree-structured lstm with head lexicalization. *ArXiv*, abs/1611.06788, 2016.

James Thorne, Andreas Vlachos, Christos Christodoulopoulos, and Arpit Mittal. Fever: a large-scale dataset for fact extraction and verification. In *NAACL*, 2018.

Alice Tontodimamma, Eugenia Nissi, Annalina Sarra, and Lara Fontanella. Thirty years of research into hate speech: topics of interest and their evolution. *Scientometrics*, 126:157–179, 2021.

Ashish Vaswani, Noam M. Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *ArXiv*, abs/1706.03762, 2017.

Bertie Vidgen, Tristan Thrush, Zeerak Waseem, and Douwe Kiela. Learning from the worst: Dynamically generated datasets to improve online hate detection. *ArXiv*, abs/2012.15761, 2021.

Elena Voita. NLP Course For You, Sep 2020. URL `https://lena-voita.github.io/nlp_course.html`.

Pia Weissenhorn, Yuekun Yao, L. Donatelli, and Alexander Koller. Compositional generalization requires compositional parsers. *ArXiv*, abs/2202.11937, 2022.

Adina Williams, Andrew Drozdov, and Samuel R. Bowman. Do latent tree learning models identify meaningful structure in sentences? *Transactions of the Association for Computational Linguistics*, 6:253–267, 2018.

Wenjie Yin and Arkaitz Zubiaga. Towards generalisable hate speech detection: a review on obstacles and solutions. *PeerJ Computer Science*, 7, 2021.

Fabio Massimo Zanzotto, Andrea Santilli, Leonardo Ranaldi, Dario Onorati, Pier Mattia Tommasino, and Francesca Fallucchi. Kermit: Complementing transformer architectures with encoders of explicit syntactic interpretations. In *EMNLP*, 2020.

Justine Zhang, Cristian Danescu-Niculescu-Mizil, Christina Sauper, and Sean J. Taylor. Characterizing online public discussions through patterns of participant interactions. *Proceedings of the ACM on Human-Computer Interaction*, 2:1 – 27, 2018.

Xuhui Zhou, Maarten Sap, Swabha Swayamdipta, Noah A. Smith, and Yejin Choi. Challenges in automated debiasing for toxic language detection. In *EACL*, 2021.

Appendix A

Code implementation

A.1 Dataset

We attach the code to create the dataset for the Syntactic Encoder. Several functions are needed to do padding procedures and manage the data for parallelization purposes. In input, we give GloVe embeddings' vocabulary, sequence of words (divided in trees), POS tags (corresponding to the words), the maximum dimension of a dependency tree (in terms of nodes), the maximum number of trees for a single tweet, the original tweets (for visualization purpose) and the labels. We need to explicitly define the order for visiting the trees and keep track of the parent of each node. An important trick is to, given a node n, represent in "reverse order" n's children in the visiting order, in this way we can exploit the same "visit order" vectors both for the top-down procedure and bottom-up one.

```
import torch
from torch.utils.data import Dataset
import utils_proc

class TweetStructure(Dataset):

    def __init__(self, glove, words, poss, parents, max_len_tree, max_roots, tweets, labels):
        #input to TreeLSTM
        visit_order, parent_visit_order = utils_proc.visit_tree(parents) #take vectors for order of visiting
        #preprocess step, padding: same number of trees for each tweet, same number of nodes for each tree
        self.words, self.poss, self.visit_order, self.parent_visit_order, self.pad_mask_trees, self.
             pad_glove_phrase = \
            utils_proc.preprocess(words, poss, visit_order, parent_visit_order, max_len_tree, max_roots)
        w_emb = glove #take glove dictionary
        pad = torch.Tensor.zero_(torch.Tensor(2, 50))
        #add two dummy vectors to glove, because of padding indeces (-1)
        self.glove = torch.cat((w_emb, pad), dim=0)
        #keep list of tweets, for further analysis
        self.tweets = list()
        for t in tweets:
            self.tweets.append(preprocess(t))
        #labels
```

```
        self.labels = labels

    def __len__(self):
        return len(self.tweets)

    def __getitem__(self, idx):
        #take tweet
        tweet = str(self.tweets[idx])
        #initialize embeddings
        emb = []
        #keep list of words with no glove embedding associated. We decided to replace it with the avg of
            embeddings
        unk = list()
        #for each tree
        for tree in self.words[idx]:
            emb_tree = []
            #initialize avg value
            avg = self.glove[0] * 0
            count = 0
            #for each word inside the tree
            for word_idx in range(len(tree)):
                #append the embedding from glove. Thanks to two dummy dimensions added previously, if a word
                    has no GloVe embeddings it takes a Zero Tensor
                emb_tree.append(self.glove[tree[word_idx]])
                #collect sum over the phrase
                avg += self.glove[tree[word_idx]]
                #if index == -1 (no embedding associated), take trace of its index
                if tree[word_idx] == -1:
                    unk.append(word_idx)
                else:
                    count +=1 #otherwise count the value
            for w in unk: #replace embeddings of unknown words with the avg of other embeddings
                emb_tree[w] = avg/count
            #"Stack everything"
            emb_tree = torch.stack(emb_tree)
            emb.append(emb_tree)
        emb = torch.stack(emb)

        return{
            'tweet_text': tweet, #original tweet
            'emb': emb, #embeddings
            'poss': torch.Tensor(self.poss[idx]), #POS tags
            'pad_glove_phrase': torch.Tensor(self.pad_glove_phrase[idx]), #"mask" for recognizing pad values
                from not pad values (like attention mask in BERT)
            'visit_order': torch.Tensor(self.visit_order[idx]), #extract visit order
            'parent_visit_order': torch.Tensor(self.parent_visit_order[idx]), #parent of node visited
            'pad_mask_trees': torch.Tensor(self.pad_mask_trees[idx]),   #pad mask for trees
            'labels': torch.Tensor([self.labels[idx]])  #labels
        }
```

A.2 Syntactic Encoder

We report here the code for the Syntactic Encoder implementation. The elements in input are the ones shown in the previous Section.

```
import torch as torch
import torch.nn as nn
import torch.nn.functional as F

class TweetToxicityClassifier(nn.Module):

    def __init__(self,
                 size,
                 num_classes,
                 vocab,
                 device,
                 ):
```

```
        super(TweetToxicityClassifier, self).__init__()

        #TreeLSTM structure
        self.size = size #size of input embeddings
        self.device = device    #cuda or cpu

        #BiLSTM parameters run at the very beginning
        self.forward_bilstm = torch.nn.LSTMCell(size+25, size, bias=False).to(self.device)
        self.backward_bilstm = torch.nn.LSTMCell(size+25, size, bias=False).to(self.device)

        #LSTM from root to leaves
        self.cell_topdown = torch.nn.LSTMCell(2*size, 2*size, bias=False).to(self.device)

        #LSTM over children of the same parent
        self.cell_bottomup = torch.nn.LSTMCell(4*size, 2*size, bias=False).to(self.device)

        # "move up" parameters, which take last vectors of the children sequence to the parent
        self.move_up = nn.Linear(2*size, 2*size, bias=False).to(self.device)
        self.move_up_init = nn.Linear(2*size, 2*size, bias=False).to(self.device)
        self.act_move_up = nn.Tanh()

        #final LSTM, over different tree embeddings of the same tweet
        self.act_root = nn.ReLU()
        self.root_to_sent = nn.Linear(2*size, 2*size, bias=False)
        self.sentence_lstm = torch.nn.LSTMCell(2*size, 2*size, bias=False).to(self.device)

        #for POS tagging, 25 one-hot vectors [0,24] + a zero tensor, as padding value
        self.one_hot_pos = torch.cat((F.one_hot(torch.Tensor(range(25)).long(), num_classes=25), torch.Tensor.
              zero_(torch.Tensor(1, 25)))).to(self.device)

        #final linear transformation, before softmax (our MLP)
        self.drop = nn.Dropout(p=0.1)
        self.act_mlp = nn.ReLU()
        self.act_mlp2 = nn.Tanh()

        self.linear1 = nn.Linear(2 * size, 50)
        self.linear2 = nn.Linear(50, 100)
        self.linear3 = nn.Linear(100, 2)

        self.softmax = nn.Softmax(dim=1).to(self.device)



    def forward_syntax(self, batch_size, n_trees, n_tokens,  word, pos, visit_order, parent_visit_order,
          pad_mask_trees, pad_phrase):
        #initialize tensor to store final representation
        syntax_vector = torch.Tensor.zero_(torch.Tensor(batch_size, 2 * self.size)).to(self.device)

        #from [tweet, tweet_trees, node] to [trees, node], for each type of data
        word = word.reshape(batch_size * n_trees, n_tokens, self.size)
        pos = pos.reshape(batch_size * n_trees, n_tokens)
        visit_order = visit_order.reshape(batch_size * n_trees, n_tokens)
        parent_visit_order = parent_visit_order.reshape(batch_size * n_trees, n_tokens)
        pad_phrase = pad_phrase.reshape(batch_size*n_trees, n_tokens)

        #tensor to access to different items
        std = torch.Tensor(range(len(word))).long().to(self.device)

        #initialize BiLSTM vectors, where to store "processed" vectors
        #two more tensor at the end because of padding (-2 and -1 are padding values, for root and parent of
              pad nodes)
        e_vect_forw = torch.Tensor.zero_(torch.Tensor(len(word), len(word[0]) + 2, self.size)).to(
            self.device)
        e_vect_back = torch.Tensor.zero_(torch.Tensor(len(word), len(word[0]) + 2, self.size)).to(
            self.device)

        # HERE I PUT THE BiLSTM PROCEDURE
        #initialize cell memory and h vector
        h = torch.Tensor.zero_(torch.Tensor(len(word), self.size)).to(self.device)
        c = torch.Tensor.zero_(torch.Tensor(len(word), self.size)).to(self.device)

        #FORWARD LSTM
        for i in range(len(word[0])):
```

```
        input = torch.cat((word[:, i, :], self.one_hot_pos[pos[:, i].long()]), dim=1) #concat [word_embed,
            pos_embed]
        h, c = self.forward_bilstm(input, (h, c))
        p = pad_phrase[:,i].reshape(batch_size * n_trees, 1)
        # if it is padding value, drop to zero h and c
        h = h * p.expand(-1, h.size()[1])
        c = c * p.expand(-1, c.size()[1])
        #store h value in the relative position
        e_vect_forw[:, i] = h

    #initialize cell memory and h vector
    h = torch.Tensor.zero_(torch.Tensor(len(word), self.size)).to(self.device)
    c = torch.Tensor.zero_(torch.Tensor(len(word), self.size)).to(self.device)

    #BACKWARD LSTM
    for i in reversed(range(len(word[0]))):
        input = torch.cat((word[:, i, :], self.one_hot_pos[pos[:, i].long()]), dim=1)
        h, c = self.backward_bilstm(input, (h, c))
        p = pad_phrase[:,i].reshape(batch_size * n_trees, 1)
        h = h * p.expand(-1, h.size()[1])
        c = c * p.expand(-1, c.size()[1])
        e_vect_back[:, i] = h

    #concat FORWARD RESULT and BACKWARD RESULT
    e_vect = torch.cat((e_vect_forw, e_vect_back), dim=2).to(self.device)

    # TOP-DOWN-FILTERING
    # resulting representation for each word from TOP-DOWN filtering
    # 2 dummy vectors, useful for PAD elements and "parent of the root" and parent of pad nodes.
    s_vect = torch.Tensor.zero_(torch.Tensor(len(word), len(word[0]) + 2, 2 * self.size)).to(
        self.device)

    # tensor to save memory-cells in the RecNN for each word from TOP-DOWN filtering
    c_vect = torch.Tensor.zero_(torch.Tensor(len(word), len(word[0]) + 2, 2 * self.size)).to(self.device)

    #follow visit ordering fixed in the dataset, previous state is stored in the parent position (for both
          s and c)
    for i in range(len(word[0])):
        vo = visit_order[:, i].long()
        pvo = parent_visit_order[:, i].long()
        s_vect[std, vo], c_vect[std, vo] = self.cell_topdown(e_vect[std, vo], (s_vect[std, pvo], c_vect[
            std, pvo]))

    #initialize vectors for BOTTOM-UP procedure
    h_vect = torch.Tensor.zero_(torch.Tensor(len(word), len(word[0]) + 2, 2 * self.size)).to(
        self.device)
    c_vect = torch.Tensor.zero_(torch.Tensor(len(word), len(word[0]) + 2, 2 * self.size)).to(self.device)
    pad = torch.Tensor.zero_(torch.Tensor(len(word), 2, 2 * self.size)).to(self.device)

    #first initialization: take vector from TOP-DOWN procedure and pass all to the "move up" network: in
         this way, leaves are initialized
    x_init_vect = self.act_move_up(self.move_up_init(torch.cat((s_vect.clone(), pad), dim=1)))
    x_vect = x_init_vect.clone()

    # BOTTOM-UP PROCEDURE
    #visit in "reverse" order
    for i in reversed(range(len(word[0]) - 1)):
        vo = visit_order[:, i + 1].long()
        pvo = parent_visit_order[:, i + 1].long()
        #take h and c from previous child in children chain (store in father position), take x from the
            child in exam
        h_vect[std, pvo], c_vect[std, pvo] = self.cell_bottomup(torch.cat((s_vect[std, pvo], x_vect[std,
            vo]), dim=1),
            (h_vect[std, pvo], c_vect[std, pvo]))
        #update x every time
        x_vect[std, pvo] = self.act_move_up(self.move_up(h_vect[std, pvo]))

    # OUTPUT: x vector assigned to the #batch_size roots
    x = x_vect[std, visit_order[:, 0].long()]
    # transform from [trees, vector] to [tweets, tweet_trees, vector]
    x = x.reshape(batch_size, n_trees, 2 * self.size)
    #Final LSTM!
    c = torch.Tensor.zero_(torch.Tensor(x.size()[0], 2 * self.size)).to(self.device)
```

```
        for i in range(x.size()[1]):
            syntax_vector, c = self.sentence_lstm(self.act_root(self.root_to_sent(x[:, i, :])), (syntax_vector
                , c))
            syntax_vector = syntax_vector * pad_mask_trees[:, i].expand(x.size()[0], 2 * self.size)
            c = c * pad_mask_trees[:, i].expand(x.size()[0], 2 * self.size)
        return syntax_vector


    def forward(self, word, pos, visit_order, parent_visit_order, pad_mask_trees, pad_phrase):
        # EXTRACT TREE EMBEDDING
        # index for each element of the batch
        #size of the single BATCH
        batch_size = word.size()[0]
        #number of trees for each tweet
        n_trees = word.size()[1]
        #number of tokens for each tree
        n_tokens = word.size()[2]
        #extract tweet "syntactic" embedding and give it in input to final MLP
        class_vector = self.forward_syntax(batch_size, n_trees, n_tokens,  word, pos, visit_order,
            parent_visit_order, pad_mask_trees, pad_phrase)
        output1 = self.drop(self.act_mlp(self.linear1(class_vector)))
        output2 = self.drop(self.act_mlp(self.linear2(output1)))
        logits = self.act_mlp2(self.linear3(output2))

        return self.softmax(logits)
```