Phonetic normalization as a means to improve toxicity detection

Mémoire

Charles Poitras

Maîtrise en informatique - avec mémoire
Maître ès sciences (M. Sc.)

Québec, Canada

Phonetic normalization as a means to improve toxicity detection

Mémoire

Charles Poitras

Sous la direction de:

Richard Khoury, directeur de recherche

Résumé

À travers le temps et en présence des avancements de la technologie, l'utilisation de cette technologie afin de créer et de maintenir des communautés en ligne est devenue une occurrence journalière. Avec l'augmentation de l'utilisation de ces technologies, une tendance négative peut aussi se faire identifier; il y a une quantité croissante d'utilisateurs ayant des objectifs négatifs qui créent du contenu illicite ou nuisible à ces communautés. Afin de protéger ces communautés, il devient donc nécessaire de modérer les communications des communautés. Bien qu'il serait possible d'engager une équipe de modérateurs, cette équipe devrait constamment grandir afin de pouvoir modérer l'entièreté du contenu. Afin de résoudre ce problème, plusieurs se tournent vers des techniques de modération automatique. Deux exemples de techniques sont les "whitelists" et les "blacklists". Malheureusement, les utilisateurs néfastes peuvent facilement contourner ces techniques à l'aide de techniques subversives. Une des techniques populaires est l'utilisation de substitution où un utilisateur remplace un mot par un équivalent phonétique, ou une combinaison visuellement semblable au mot original. À travers ce mémoire, nous offrons une nouvelle technique de normalisation faisant usage de la phonétique à l'intérieur d'un normalisateur de texte. Ce normalisateur recrée la prononciation et infère le mot réel à partir de cette normalisation, l'objectif étant de retirer les signes de subversion. Une fois normalisé, un message peut ensuite être passé aux systèmes de classification.

Abstract

Over time, the presence of online communities and the use of electronic means of communication have and keep becoming more prevalent. With this increase, the presence of users making use of those means to spread and create harmful, or sometimes known as toxic, content has also increased. In order to protect those communities, the need for moderation becomes a critical matter. While it could be possible to hire a team of moderators, this team would have to be ever-growing, and as such, most turn to automatic means of detection as a step in their moderation process. Examples of such automatic means would be the use of methods such as blacklists and whitelists, but those methods can easily be subverted by harmful users. A common subversion technique is the substitution of a complete word by a phonetically similar word, or combination of letters that resembles the intended word. This thesis aims to offer a novel approach to moderation specifically targeting phonetic substitutions by creating a normalizer capable of identifying how a word should be read and inferring the obfuscated word, nullifying the effects of subversion. Once normalized phonetically, the messages are then sent to existing means of classification for automatic moderation.

Contents

List of Tables

List of Figures

Thanks

I would like to extend my thanks to my research director, Richard Khoury for the instrumental help offered throughout this research and for the opportunity to take part in it. Thanks also go out to the team at Two Hat Security for offering a place to do research and use this research on real-world examples. Finally, I would like to thank Marc-André Larochelle for the suggestions he offered to guide my research.

Introduction

With the ever-growing presence of online communications, social media and other interconnected services, it is an undeniable fact that these tools have become an important part of daily life. With that come many boons, such as an increased reach in who we can communicate with. Unfortunately, that very boon is also a curse in the way that is allows undesirables to get into contact with us and others. While getting rid of those people might be as simple as hitting the block button, some vulnerable people might not even be aware of that it's an option, or not realize that the person they are in contact with is toxic. Among those vulnerable people we often find children of impressionable ages. A popular method of protecting those individuals is to create communities specially targeted towards them. This, however, also means that these new communities are prime targets for toxic individuals and measures must be put in place in order to stifle their nefarious actions. This is a necessary thing to do if we want the protected communities to thrive and be safe from minor things like offensive language, as well as major and harmful actions like grooming.

Many methods currently exist to enforce policies and protect communities. Examples of those methods include, but are not limited to: blacklists, whitelists and rule-based text analysis similar to expert systems. These can be powerful tools in the fight against online harm, but unfortunately they are all saddled with an important flaw: the large amount of work required to maintain and update them. Languages evolve to better fit the needs of the population using them and better represent the reality of the times. This fact is also true for online communications where this evolution is greatly sped up. Be it to shorten words and sentences, to visually change the words or to invent words representing new phenomena, online users evolve their common language faster than most people can keep up; entire websites have sprung up to help catalogue online lingo.

Another major part of the ever-evolving landscape that needs to be taken into account is that harmful users can try to bypass existing rules by slightly altering the words they use. This strategy is called *subversion*, as the harmful user is said to be subverting the rules of the community. These alterations can take many forms, from dropping certain letters, to multiplying them and replacing them by characters that are outside of the English alphabet, there are countless ways harmful users can try and bypass filters. A simple way to prevent

abuse of this sort could be to simply disallow words that are not part of a dictionary, but this approach is flawed as normal users making typos or using innocuous alterations to words, such as "2morrow" instead of "tomorrow", might find themselves unable to communicate and leave the community. This is an especially flagrant flaw when it comes to younger or less educated users that simply don't know, or don't care about, how every word is written. Since the goal is to maintain a safe and thriving community, this might lead to an early demise. One solution to this poor grammar issue could be to allow common substitutions in the filters, but this comes back to the inordinate amount of work required to maintain and update the filters.

An alternative solution to this problem is to enhance filter-based systems by giving them the ability to automatically correct out-of-vocabulary (OOV) words to their English counterparts. This process is called *normalization*. Done correctly, it would allow unrestricted healthy conversations while making it impossible to disguise harmful speech to circumvent filters.

In this thesis, I aim to create a system that will allow for the normalization of words through phonetics. The decision to use phonetics is based on the fact that modifications to words are often made so that the modified word can be read out loud and pronounced similarly to the correct word. Consequently, a two-part system capable first of figuring out how an OOV word sounds should then be able to figure out what the intended word was. Designing and testing this two-part system is the main contribution of this thesis.

Two sets of experiments will be used to validate this work. The first will be normalization experiments, to see if the phonetic normalization system can recover the correct intended words. The second will be harm detection experiments, to see if phonetic normalization can help a classifier solve the subversive harm challenge.

The goal of the research is to, through the use of a novel normalizer making use of character-level analysis and phonetics, offer a way to help reduce the effects of subversion in the context of online communications.

The results of our experimentation and research are presented in three different sections and five different chapters. Chapter 1 covers the related works we found throughout our research, and makes up the first section of this thesis. The second section covers the methodology and reasoning used in the creation of our normalizer in Chapter 2, along with the normalization experiments and results in Chapter 3. Lastly, the third section studies our use-case, the resilience our normalizer offers against the problem of subversion. This is presented in Chapter 4, along with experimental results in Chapter5.

Chapter 1

Related Works

1.1 Online Harm and Subversion

1.1.1 Harmful Content

Harmful content, sometimes also known as online toxicity or toxic content, is broadly defined as content that can bring online harm. According to the Fair Play Alliance's *Framework Disruption and Harms in Online Gaming* [19], harmful actions in video games can be summarized as disruptive behaviours that stray from the intended goal of the online game. If we broaden this definition to cover all online communications, harmful content is anything that would go against the concept of a healthy conversation.

However, as pointed out in [22], such a definition is extremely broad, and what is considered healthy by some might be harmful to others for reasons such as culture, age, lived experiences or belief systems.

According to a 2020 survey done by the ADL [7], 81% of individuals surveyed reported having been exposed to some form of harassment. According to [19], this is only one of many harmful behaviours. Harmful behaviours relevant to online conversations include hate, extremism, sharing of inappropriate content, criminal or predatory conduct, dangerous speech, abuse, antisocial actions, purposeful disruption, aggravation and harassment.

Due to the large variation in how online harm can be presented, the many forms it can take, and the high level of subjectiveness that can be present in both how it is represented and how it is received, we believe the best way to describe harmful behaviours would be to define it as any action taken in order to voluntarily disrupt someone's state in a negative manner, or to take actions that could result in such consequences. Those behaviours are not necessarily limited to a single interaction and can be spread over large periods of time, such as in the case of cyberbullying.

Given this definition of harmful behaviour, harmful content is anything created in the process

of executing harmful behaviours, which can result in online harm.

1.1.2 Subversion

The dictionary definition of subversion is the systematic attempt to overthrow or undermine a system by persons working from within. In the context of online harm detection, subversive users are individuals trying to circumvent the rules of a community through the use of tactics aimed at the flaws in the systems.

Attempts at subversion can take many forms and they constantly evolve along with the systems they try to subvert. Some users might resort to using “1337 sp33k” (leet speek) while others may add odd punctuation or spacing in their words as listed in [26]. Those techniques are fairly simple and can be prevented by the use of filtering techniques like whitelists, but even a system as simple as a whitelist can be broken in imaginative manners. Two simple examples would be the twisting of definitions by describing anatomical parts as "trouser snakes," or the use of phonetic alterations to create words, such as “as whole.”

Subversion evolves so rapidly that more complex community management systems such as Community Sift [1] offer large amounts of customizations to allow for the rapid iteration in the system rules to catch and prevent subversive actions.

Subversion does come with one large issue, however, and that is the intentionality that is involved in it. We do not believe it is possible to infer intent by simply looking at a message. For this reason, detecting subversion is extremely difficult. This does not signify that it is impossible to detect subversive elements, even if that task is difficult due to the rapid evolution of subversion tactics that harmful users make use of. The simplest way to accomplish this task would be to consider anything that isn’t a word or a common typo as potentially subversive and act upon this information, but even this approach could not catch everything cases like “trouser snake” would simply be outside of that scope.

1.2 Character Based Phonetic Normalization

While there have been a few attempts at normalizing text using phonetics, we found no system that was designed to identify phonetics and then recreate an English word using the International Phonetic Alphabet (IPA).

1.2.1 Phonetic Representation

There are multiple different ways to represent the phonetic content of a word, including both different levels of pronunciation details and different representations of the sounds. The three

[1] `https://www.twohat.com/community-sift/`

main representations are alphabetic representations, iconic representations and analphabetic representations.

Alphabetic Systems

Alphabetic systems are systems designed to use either a language's own alphabet or a new alphabet in order to represent pronunciations in a way that can be read and understood by someone familiar with the system. While this is not a feature unique to this type of system, it is still an important part.

Most alphabetic systems use an algorithm to create the phonetic form of words. An example of such an algorithm would be the one within the American Soundex system [9], which is designed to phonetize surnames in order to help analyze census data. The way this is done is by taking a last name and turning it into a four-character code comprised of a single letter and three digits. The characters contained within this code are obtained by taking the first letter of the name and, by following a few selection rules, converting consonants into to a number and selecting the first three numbers. As can be infered from this process, quite a few words will have identical codes. We call those repeated encoding results, collisions. Along with the American Soundex, being itself a variation of the Russel Soundex [27], we can also find other variations of the core principles in the Daitch-Mokotoff Soundex [6] and Cologne phonetics [25] offering other similar encodings, both with varied amounts of collisions and aiming to solve different or broader issues.

Other algorithms such as Metaphone[4], New York State Identification and Intelligence System (NYSIIS) [8], Match Rating Approach (MRA) [5] and Caverphone [1] also exist and generally offer fewer collisions within the encodings. While their algorithms are slightly more complex than the one for the Soundex system, they allow for greater flexibility since there are fewer collisions between the phoneticized words. Some of them, such as the metaphone algorithm, are already being used in some spell-checking software to help identify which word was actually intended by comparing the phonetic encoding with a database of other encodings.

Since, in many languages, an alphabet letter can represent multiple sounds, alphabetic systems often use a non-ambiguous phonetic alphabets, such as the International Phonetic Alphabet (IPA). This alphabet was introduced as a way to accurately represent how a word sounds. This was done by creating a large alphabet containing many different characters each representing a different phoneme. Its accuracy has allowed it to become the main way to represent pronunciations within many dictionaries. While it is already very accurate, there are also some variations such as Ext-IPA [10] or VoQS [11].

Examples of encodings with the various algorithms, including some collisions, are included in the table 1.1. While the last three examples were carefully selected to show examples of collisions, it demonstrates that no system is perfect. The case of "carry" and "Kira" shows

	Soundex	Metaphone	NYSIIS	MRA	Caverphone	IPA
Elephant	E415	ELFNT	ELAFAD	ELPHNT	ALFNT11111	/ɛl ə fənt/
Alphabet	A411	ALFBT	ALFBT	ALPHBT	AFPT111111	/ˈælfəˌbɛt/
Kira	K600	KR	CAR	KR	KRA1111111	/ˈkɪərə/
Carry	C600	KR	CARY	CRY	KRA1111111	/ˈkærɪ/
Carrey	C600	KR	CARY	CRY	KRA1111111	/ˈkærɪ/

Table 1.1: Encodings of various words under different phonetic systems. Soundex, Metaphone, New York State Immunization Information System (NYSIIS), Match Rating algorithm (MRA), Caverphone and International Phonetic Alphabet (IPA)

flaws in some of the simpler systems and an example of where the IPA prevails when it comes to differentiating the two pronunciations. In the case of "carry" and "Carrey", there are collisions across the board, even in IPA form.

Iconic and analphabetic systems

While using an alphabetic phonetic representation is by far the most common approach, two alternatives do exist.

Iconic systems represent a word's pronunciation using some sort of graphical representation of the sounds. An example of an iconic system would be the Visible Speech system[21] which aims to represent phonemes with pictograms representing a side view of a mouth pronouncing them. This makes it an interesting and useful tool when learning to pronounce words, particularly for deaf people.

Analphabetic systems are probably the most uncommon way to represent phonetics as they can become extremely complex. An example of such a system would be the system designed by Keneth Pike [24]. This system aims to decompose each sound into all of its components, ranging from the way the air flows in the lungs and mouth to the phonetic functionallity of the sound. Due to high level of complexity of those systems, they will be ignored for this research.

1.2.2 Phonetic Normalization

While there have been many novel systems designed to approach the task of normalizing text, such as the work done in [29] where the soundex system was used, or the work done in [18] where a custom system of phonetic signatures was built, we believe the closest normalization system to what we propose in this thesis is the one created by [28].

Their normalizer uses the `g2p-seq2seq` [2] network, a transformer-based neural network that generates phonemes based on the CMUDict[3] dictionary. This dictionary uses a phoneme set

[2] `https://github.com/cmusphinx/g2p-seq2seq`
[3] `https://github.com/cmusphinx/cmudict`

based on the ARPAbet[20] symbol set. The authors, using the phonemes, then build a set of potential words for each word present which is then used to reconstruct a sentence based on phonetic similarity and next-word probability to rebuild the most likely intended sentence based on how it sounds.

Their system differs from the system we propose in the way the messages are reconstructed in two different aspects. First, our proposed solution is made up of two different sequence-to-sequence models. One for the original phoneticization, and one to recreate the word. The second difference is their system uses next-word probability to guess the intended message, whereas we normalize words in a vacuum.

1.2.3 Character Based normalization

In [26] we find a system similar to what we propose in the way they use character based normalization. Their system, broadly aimed at most types of word-level subversion, uses a `RoBERTa` model to do character-level analysis and recreate a word that should be contained in the English dictionary, offering a system capable of directly normalizing a word that potentially contains subversive elements. Their system is also capable of doing sentence-level analysis by grouping multiple sets of tokens together and treating them as a single word.

When compared to our system, this character-based normalizer differs mainly in the fact that it does not aim to identify the phonetics of a word as a part of the normalization process.

In each of [28] and [26] we find a high similarity to the goal we are trying to achieve, even moreso if we combine both of their work together. However, their research was done in parallel to ours, and by the time we became aware of their work, ours was either already ongoing or in its finalization phase. As such, no input from their work is found in ours.

Chapter 2

Text Normalization

In this chapter, we will present the work we did on text normalization. Among the topics that will be discussed will be the reason behind the use of phonetics and the International Phonetic Alphabet (IPA), the datasets and how they are built, and the various networks that we trained. The next chapter will analyze the results of the experiments we will set up here.

2.1 Methodology

The underlying intuition behind our work is that, while online users can generate nearly endless spelling variations of words, either accidentally (e.g. typos) or deliberately (to write faster, add a personal flavour to their message, etc.), they do so in a way that creates words that would sound nearly identically to their correct counterparts if they were read out loud. For instance, misspelled words like “luv”, “any1”, “togedar”, can still be pronounced correctly.

Following this idea, the text normalization system we propose is composed of two different deep neural networks working in tandem to generate phonetic versions for Out-of-Vocabulary (OOV) words, and use the generated pronunciation to discover In-Vocabulary (IV) words that sound similar or identical to what was written in the message.

2.1.1 Phonetic translation

The underlying idea guiding the development of our system is to analyze the way an OOV word is written on a character level and generate a phonetic representation that will then be used to build the IV word that was intended by the author. Our approach is similar to what is done in machine translation. For that challenge, one takes a word, an entire sentence, or a set of sentences, in a source language and feeds them through a network to obtain the equivalent content in a target language. In our case we have three languages. The first one is Internet English, the unedited version of English people use online. The second language is the phonetic representation of the same, which uses a different alphabet altogether. And

the third language is proper English. We will thus have two translation tasks, from Internet English to phonetics, and from phonetics to English.

For our work, we start by translating an individual OOV, which we will treat as a sentence composed of characters. Our intuition is that each character can translate to a pronunciation from a set of possible pronunciations it can take, and picking the correct one depends on the context (the surrounding characters), in much the same way as a word can translate to one of several possible words in a target language based on the context of surrounding words. In turn, the phonetic translation of the OOV word is taken as a sentence of characters and each is translated into the most likely English letter given the context of the surrounding characters. This translation task is what we hope the networks can learn and use to normalize words. An additional optional final step could be to put the output of the English translation through a spellchecker system to make sure the word recovered is real and written correctly. However, as our research focuses on the phonetic normalization network, this additional step is outside the scope of our work.

2.1.2 Limitations

The underlying assumptions of our system are that there is a 1-to-1 correspondence of OOV and IV words, and that the OOV words are phonetically similar to the IV words they correspond to. As a result, our proposed approach will not be able to handle forms of OOV word spellings found online that violate these assumptions.

To begin, acronyms cannot be handled by our system. For example, writing "LOL" instead of "laugh out loud" is not phonetically similar at all, and thus violates both our assumptions. Likewise, while the acronym "IOU" is phonetically similar to the phrase "I owe you" it represents, our network is designed to output individual words and would not be able to recover the phrase.

Likewise, OOV words created by splitting an IV word into multiple words using spaces cannot be handled by our methodology. Our system will simply see each portion as a separate word and try to recover an IV word for it.

While our method may be able to handle missing individual letters, it cannot account for entire missing syllables in words. For example, it would not be able to normalize "bday" to "birthday" as it has no way to recover the deleted phonetic information.

Finally, our system is designed to normalize to the English language, which means it cannot handle URLs, usernames, hashtags, and other non-English strings often found in messages. It also cannot handle messages in other languages, although it would be possible to retrain our system to handle any language that uses a phonographic alphabet (logographic alphabets violate our basic assumptions). Our system cannot handle messages that mix multiple

languages. Fixing that issue would be a non-trivial problem, as different languages will pronounce the same letters differently. The simplest solution to this problem may be to add a pre-processing step to pinpoint the span of each language in the multilingual message.

2.2 Normalization Network

2.2.1 Network Libraries

In this research, we experiment with two different deep neural architectures. The first one is the `fairseq` [1] architecture by Facebook, and the second one is the `transformers` [2] architecture offered by Huggingface.

The first library, `fairseq`, offers a command-line API that allows users to directly feed sentences to it. In our case, the sentences are the various words used to train the networks for normalization. The networks of this library were designed with translation in mind, so no modifications were required on our part.

We initially opted to use the `fairseq` library due to the ease that existes in using the embeddings generated by the `fasttext` library. After a while, due in part by the rise in popularity of the `transformers` library, and the desire to test out more networks, we added a second library.

The second library, `transformers`, offers a set of neural networks that overlaps with the ones offered in `fairseq`, but it is mostly aimed at classification and question-answering and offers no direct way to do sequence to sequence (`seq2seq`) operations. In order to do those operations, we modified the networks by adding an output layer that generates character sequences. In addition to this alteration to the network itself, we implemented a `tokenizer` to tokenize each word's letters and characters by themselves.

In our early testing stages, we found that the `RoBERTa` models edged out the other models by a slight margin for the same amount of epochs. Consequently, every model trained using this library was based on the `RoBERTa` architecture.

The `RoBERTa` [23] architecture is a neural network architecture based on the `BERT` [16] architecture, which is itself based on the basic transformer architecture [30]. The main changes are the tuning of hyperparameters, removing the next-sentence prediction objective and the dynamic change of the sequence masking patterns of the `BERT` architecture.

[1] `https://github.com/pytorch/fairseq`
[2] `https://github.com/huggingface/transformers`

2.2.2 General Architecture

In order to implement our proposed normalization system, we have to properly deconstruct the two different translation problems. This was done by using two different networks, one for the Internet English to Phonetic task, or the English-to-IPA network, and a second one for the Phonetic to Normal English task, or the IPA-to-English network. Both of these networks work in tandem to normalize the messages. The general architecture we settled on, along with the broad steps necessary for normalization are shown in Figure 2.1. The steps are as follows:

1. Split a message into multiple words
    - For each word in the message:
        a) Tokenize the word into characters
        b) Feed the characters into the English-to-IPA network to generate the equivalent IPA word
        c) Tokenize the predicted IPA word into characters
        d) Feed the characters into the IPA-to-English network to generate the equivalent English word
2. Combine the words to form the normalized message

When passing words to a neural network using the `fairseq` library, other than requiring each word to be split character-wise, no additional steps are required, and no alterations need to be made to the network architecture. However, when using the `transformers` library, none of the architectures offered include a sequence output layer. Consequently, we add this layer to the various base architectures we train and test. This layer had to accept the output vector from the base architecture and create a prediction matrix describing the most likely character for each position in a word. Once that matrix is created, we select the character with the highest weight as the predicted character. The resulting process is represented by Figure 2.2.

2.3 Datasets

In order to conduct our experiments, we create two training datasets and obtain three test datasets.

2.3.1 Clean and Condensed Datasets

The main dataset we create we call the "clean dataset". This dataset is comprised of English-IPA word pairs scraped from TheFreeDictionary website[3]. This dictionary includes multiple

[3] `https://www.thefreedictionary.com/`

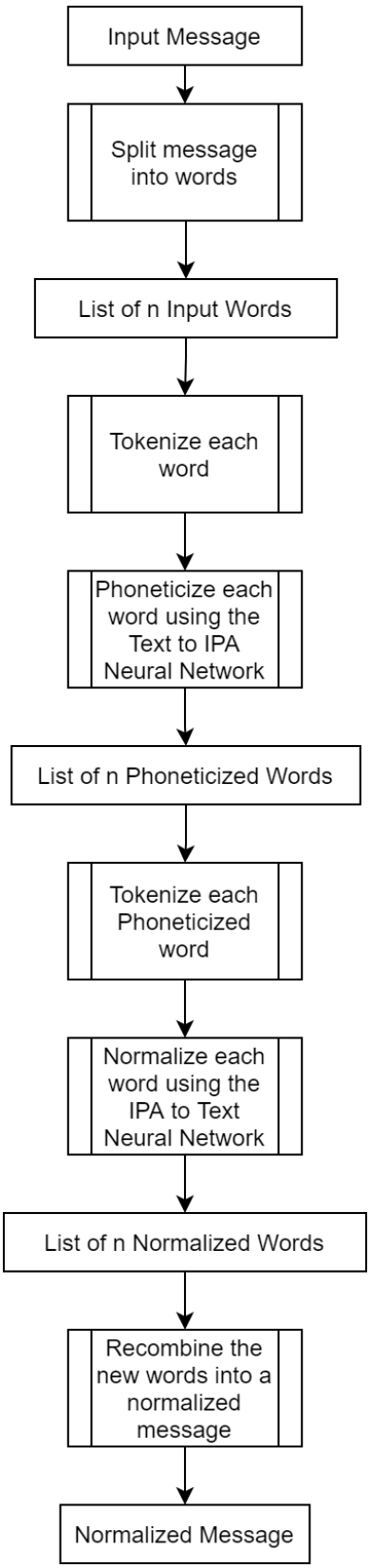

Figure 2.1: Normalization process for any message

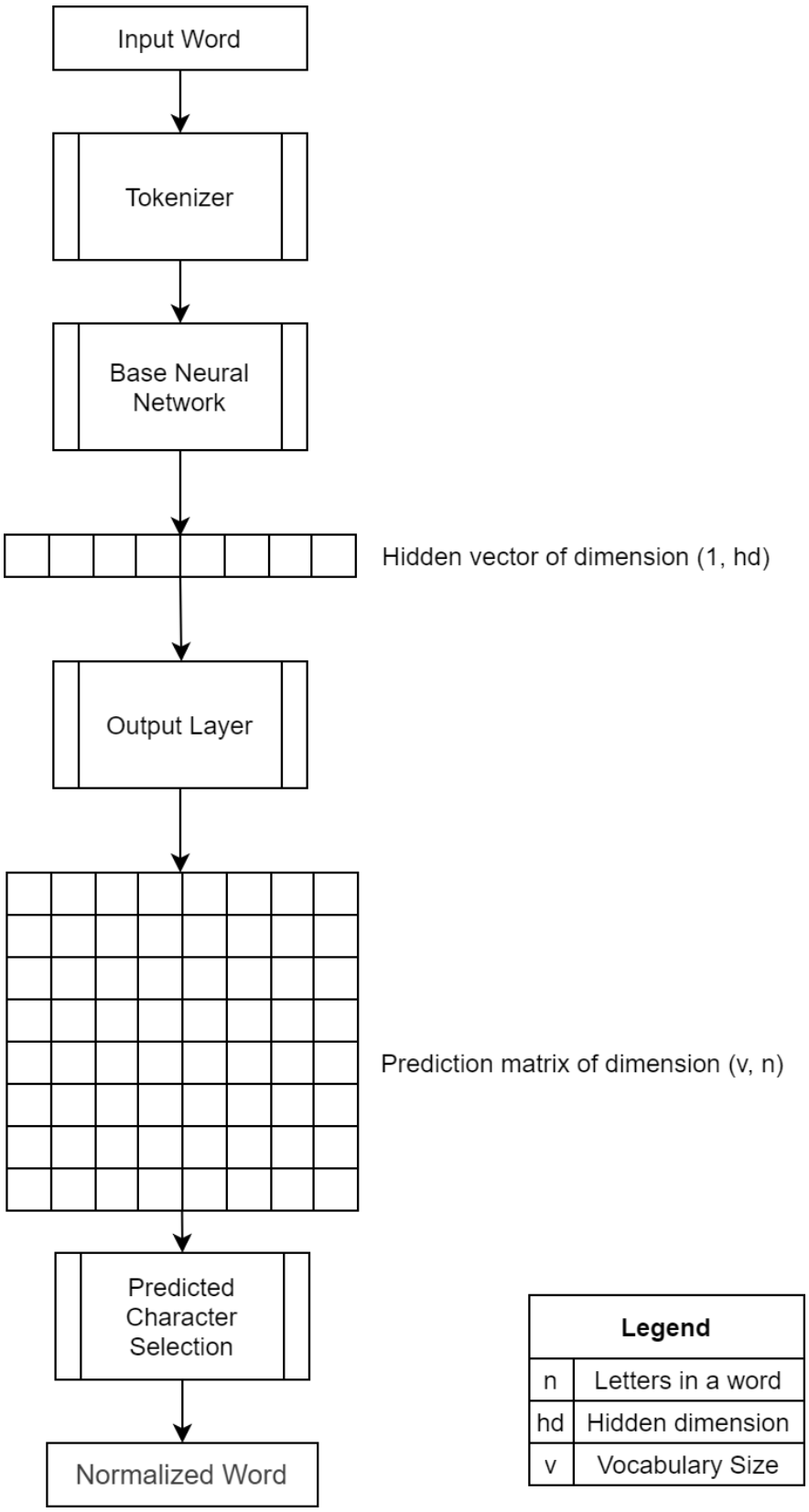

Figure 2.2: Data flow in a `transformers` library network

pronunciations for each word when variations exist. Examples of multiple pronunciations are the various ways to say "potato" (/pə-tˈt/ and /pəˈteɪtəu/), and "tomato" (/tə-mˈt/ and /təˈmɑːtəu/) and other regionalisms. In cases where there are multiple ways to write a word (e.g. "color" vs. "colour"), they are not merged and are treated as separate words. The fact that many words possess multiple pronunciations is an issue when trying to train a neural network with the goal of creating a unique output for each word it is given. This is due to the fact that during the backpropagation step, if there are multiple conflicting pronunciations, the weights will be "pulled" in conflicting directions. This effect will be compounded for each word with multiple pronunciations, preventing the network from converging to the unique output we desire. In order to prevent this, we preprocess the clean dataset into what we call the "condensed dataset". This is a condensed version of the clean dataset that contains a single pronunciation for each word. To select the best pronunciation, we experiment with four different methods which are as follows:

- Select a random English-IPA pair for each word.
- Use a preliminary version of our Phonetic-to-English network to recover the word for each IPA string, and select the IPA with the smallest Levenshtein distance between the correct word and the generated word.
- Sort the IPAs for each word in a pseudo-alphabetical order, and select the first IPA for each word. By "pseudo-alphabetical" we mean that the words are sorted in spelling order, as they would be in alphabetical order, but using IPA characters instead of the English alphabet.
- Use the most common pronunciation of each syllable.

The first method, the random selection, is the least biased method and, given a large enough sample size, allows for the greatest variation in grapheme to phoneme pairings.

The second method presents an additional challenge, namely the initial selection of English-IPA pairs to train the Phonetic-to-English model. While it is true that we could use the neural networks offered in the `g2p-seq2seq`[4] repository used by [28], there is a large overlap in the training datasets and a bias towards certain English-IPA pairs would still be present. For this reason, we settle on using the same set of random pairs used in the first method in hopes that the grapheme to phoneme variations will allow for the best generalization. Once trained, this preliminary network is used to convert each possible IPA string of a word to English, and the best IPA string was selected as the one that generates the English word with the lowest Levenshtein distance to the real word. For example, if the pronunciations for

[4] `https://github.com/cmusphinx/g2p-seq2seq`

“tomato”, /tə-mˈt/ and /təˈmɑːtəu/, predicted“"tomayt” and “tomahtoe” respectively, the first pronunciation would be included in the condensed dataset.

The third method, contrary to the first two, favours the selection of similar IPAs for similar words but introduces the most bias in the grapheme to phoneme selection. The problem this method tries to solve is that there is a large variation in the number of pronunciation for each syllable. This might slow down, or prevent, the neural networks from converging during training. By purposefully biasing the pronunciations we select by sorting them pseudo-alphabetically and keeping the first one, we create a simpler condensed dataset containing fewer variations for syllables occurring at the beginning of words. This sorting creates a bias because it will force cases where there are multiple similar ways to write IPAs to favour specific beginnings in the IPAs. For example, the word “glue” has two pronunciations starting with a different sound for the “g”, /glu:/ and /glu/, and in pseudo-alphabetical order the first one will be selected. The same decision will be made for other words beginning with the syllable “glu”, leading to uniformity in the dataset and simplifying the learning process. Other examples of this kind of variation in the spelling of the IPAs can be seen in Table 2.1, which contains a few words beginning in “abb” that have multiple IPAs and multiple ways to pronounce that syllable. As can be observed, biasing the IPAs by sorting pseudo-alphabetically does push towards a reduction in the variety of first characters, but it does not remove it completely as there are cases where specific pronunciations do not exist, or a rare pronunciation comes first in pseudo-alphabetical order and takes precedence over the more common pronunciation. In order to sort the IPAs the built-in methods provided in Python 3 on a machine with English as the default language was used.

The fourth method consists in picking the most common pronunciation for each syllable, based on observed frequency in TheFreeDictionary, and using it to create the IPA for each word. However, not every word in the clean dataset has syllable demarcations, complicating the implementation of this solution. Moreover, there is a large variation in how syllables are pronounced via their stress or emphasis, which would in turn create a lot of variation in our condensed dataset. Consequently, we opt not to explore this approach for the construction of the condensed dataset.

2.3.2 Augmented Dataset

Since our goal is to normalize incorrectly-written online text, it is important to have a dataset of words containing characters that are outside of the English alphabet, along with typos and other mistakes. In order to generate this dataset, we apply a set of substitutions to our clean dataset. A subset of the substitutions can be seen in the Table 2.2, and the complete list of 885 substitutions is made available as a Github Gist[5]. As can be observed in the table, substitutions are applied both to individual characters and to character groups. In cases like

[5]Substitutions: `https://gist.github.com/chpoit/e5cb454bbd38ad7fe05cb8636dd569f4`

Word	Existing IPAs	IPAs without stress	Selected IPAs
abbado	/əbɑdəu/	/əbɑdəu/	/əbɑdəu/
abbas	/ˈæbəs/	/æbəs/	/æbəs/
abbasid	/əˈbæsɪd/ /æbˈə-sɪdˈ/ /ˈæbəˌsɪd/ /ə-bæsˈɪdˈ/ /ˈæb ə sɪd/ / əˈbæs ɪd/	/æbəsɪd/ /əbæsɪd/	/æbəsɪd/
abbe	/ˈæbɪ/ /ˈæbeɪ/ /æ-beɪˈ/ /æˈbeɪ/ /ˈæb eɪ/ /ˈaːbə/	/æbɪ/ /abə/	/abə/
abbot	/ˈæbət/ /æbˈətˈæb/ /ət/	/æbət/	/æbət/
abbott	/ˈæbət/ /ˈæb ət/ /æbˈət/	/æbət/	/æbət/

Table 2.1: Some words beginning with "abb" and various pronunciations

Initial characters	Substitution
a	4, 🅰, 🅐, 🄰
cause	cauz, cos, cuz, kuz
N	||
ate	8, 🅰🆃🅴
ait	8
to, two	2

Table 2.2: Potential substitutions for word segments and characters

the substitution of "8" for "🅰🆃🅴", we have a phonetic substitution, but in other cases like the use of "||" instead of "N", the change is visual. Cases like substitutions for "a" are both phonetic and visual. Some visual cases were added to the substitutions to represent common subversion techniques.

In addition to these substitutions, we also doubled and removed letters at random to represent common typos.

The algorithm to build the corpus of augmented words is the following:

1. For every substitution in the substitution list, try every possible substitution on each word in the original list of words

    a) If a substitution can be made, apply it as many times as possible in a word. Add the new version of the word to a temporary list of augmented words.

2. Create an intermediate list containing the temporary list of augmented words.

3. For every element in the temporary list and the original list do the following:

    a) Iterate over the letters in the words, skipping the first and last letters, and add a copy of the word to the intermediate list without that letter.

    b) Iterate over the letters in the words and add a version of the word containing two copies of that letter to the intermediate list.

4. Create a final list containing every original word

5. For every altered word in the intermediate list:

    a) If the altered word is in the original list of words, throw it out

    b) If the altered word in not in the original list of words, add it to the final list.

6. Keep every unique entry in the final list.

Applying these steps builds a very large dataset of augmented words with many possible alterations. The order of these steps was chosen to allow for typos within augmented words. When dropping letters, the decision not to drop the first and last letters was made to avoid the addition of too many invalid pronunciations, which we define as the case where a transformed word becomes a different real word which should have a different pronunciation, but due to our corpus construction will still be paired to the original word's IPA string. An example of this would be the transformation of "factual" into "actual" or "completed" into "complete". For the same reason, we also remove every altered word that turns into another one, such as "rifle" becoming "rife" or "rile". We can note, however, that we do not filter for combinations of alterations resulting in an invalid pronunciation, such as for example the case where an augmented word such as "r1fle" has the "f" or "l" removed to become "r1le" or "r1fe". So, while we limit the spread of invalid pronunciations, they are not completely absent from our dataset.

Furthermore, while this dataset represents some of the common substitutions found in online text, it remains an artificial construction. To build a real dataset of online text would require us to collect text from a real online source such as Twitter, filter out correct English words and non-analyzable text (usernames, URLs, acronyms, etc.), and to label the correct IV English word for each remaining OOV word. While the first steps are easy to do, the labelling step represents a non-trivial workload, which is why we opted for generating a synthetic dataset instead.

2.3.3 Test Datasets

Along with our two training datasets, we also use three different testing datasets. The first one is the one used by [18] to test their algorithm, the second one is a dataset provided by EMNLP [6] and the last one is a set of tweets and their normalized form available from [2]. These datasets contain various mistakes in the words, including some mistakes that the networks are not designed to fix, such as the previously mentioned case of acronyms like "LOL", "lmao" and "idk". Those datasets, along with how they are pre-processed, are presented in section 3.2.1 of chapter 3.

2.4 Data Processing

2.4.1 Alphabets

Considering the steps needed to normalize the messages, we see a total of three different alphabets that will be in use through the normalization process. These alphabets are as follow:

- Input English Alphabet: This alphabet is the internet-English alphabet that contains every unique character found in the input datasets.
- Phonetic Alphabet: This alphabet contains every character found within the IPAs.
- Output English Alphabet: This alphabet contains every unique character in the dataset that the network needs to predict.

The first alphabet is the alphabet used as an input for the first neural network, the English to IPA network. Two versions of this alphabet exist. The first version, the Basic English Alphabet, is very similar to the output alphabet and contains only the original characters found in the non-augmented datasets. The second version, the Augmented English alphabet, contains both the characters in the Basic English Alphabet and every character introduced via the previously mentioned substitutions of the augmented dataset. The decision to have two versions of the alphabet was made to better separate the experiments with and without the augmentations in the input dataset and to reduce the number of parameters in the neural networks. The Basic English Alphabet contains 112 characters compared to the 848 found in the Augmented English Alphabet.

The second alphabet, the Phonetic Alphabet, is the alphabet containing the IPA characters found within the various English-IPA pairs. It is used both as the output of the first neural network and as the input of the second one, the IPA to English network. This alphabet

[6] A copy of the dataset is made available here: `https://gist.github.com/chpoit/4c12d4d3edced66b6f946ca3f1747e0f`

does not contain every single character within the International Phonetic Alphabet. This is because the missing IPA characters were not present within the IPAs we obtained in our dataset. Those sounds are not represented in our dataset either because they are too rare in the English language, or because they do not exist within it. In total, 56 characters are used.

The last alphabet, the Output English Alphabet, is an alphabet which contains the set of characters that makes up clean English words. This alphabet is built using the characters from the IV words contained in the clean dataset and the test datasets. This means that it is almost identical to the Basic English Alphabet. However, when compared to the Augmented English Alphabet, this alphabet is indeed a reduced set of characters. The reason behind wanting a smaller set of characters is simply because normal English does not contain a plethora of odd characters, and since we want to normalize words in a form that is as close to normal English as possible, we want the Output English Alphabet to be as close to the normal alphabet as possible. This alphabet contains the 26 letters of the alphabet along with some diacritics, as well as numerals and some other characters which are found in our test datasets. The final character count is 112 (note that they are not the same set as the 112 characters of the Basic English Alphabet).

The three different alphabets are made available in a Github Gist. [7]

2.4.2 Data Formatting and Tokenization

In order to feed the data to the `fairseq` library, each word was split into a set of letters and fed to the various neural networks through the provided command-line API. Tokenization for this library is unaltered from the default behaviour. The process for the transformer library was more complicated. In order to be able to feed the words to the embedding layer, they had to be tokenized. As mentioned previously, we wrote a tokenizer to accomplish this task. This was done because the tokenizers provided by the library tokenize words in a sentence by extracting elements of the words and sentences. As such, they are incompatible with our design as we want the networks to learn relations between characters, and not the segments of characters and sentences. The result of this tokenization gives us a vector of indices representing every character of a word. The same tokenization was done for the IPAs. Writing a custom tokenizer allows us to keep a certain consistency between each experiment by reusing the same index table between each alphabet. This custom tokenizer also allows us to perform a special tokenization which will be explained in Section 3.3.3. As was done in the default tokenizers, we also add tokens representing the beginning and the end of each word to allow the neural network to infer that some parts of the word vectors are of no importance. In most cases, we let the network learn the embeddings at the same time as it is training.

[7] Alphabets: `https://gist.github.com/chpoit/6c7e7c0325366f8ef258eca794200d3a`

2.5 Conclusion

In this chapter, we presented our use of a dual neural network architecture in order to normalize OOV words. Those two neural networks accomplish two separate tasks. The first task, the phoneticization of an OOV word using the International Phonetic Alphabet, is accomplished by the first neural network, the Text-to-IPA neural network. The second task, accomplished by the IPA-to-Text neural network, recreates an IV English word using the phoneticized form generated by the first network. Combined together, those two networks allow for the normalization of OOV words used for the broader task of normalizing messages comprised of both IV and OOV words. Using the principles and processes laid out in this chapter, the next chapter will go over the experiments that we performed to create and validate our normalization system.

Chapter 3

Text Normalization Results

This chapter will give a detailed presentation of our experiments using the processes and architectures described in the previous chapter, and will present and analyze in detail the results obtained in these experiments.

3.1 Evaluation of the Generated Words and IPAs

In order to evaluate the quality of the words and the IPAs we opt to use the Levenshtein distance to compare the original word or IPA to what was generated by our system. In addition, when comparing models, we count how many words are generated without errors. Based on those two metrics, we can select the models that gives the most accurate predictions. Every operation used to calculate the Levenshtein distance has a cost of 1, meaning that a substitution is only treated as one operation.

In order to not bias results, we compute the averages of the Levenshtein distances by excluding perfect results. So for example, if a system has 100 entries with errors and 100 without, the average would only be computed on the 100 entries with errors. In order to evaluate the number of errors in IPAs, we base our results on the IPAs we previously scraped from the web and use these IPAs as the expected results. To calculate the errors in the IV English words, we use the matching words contained in the respective dataset. For example, if we had initially fed the word “tomato” to the neural networks and we had received “tomto” as the output, it would be compared to the original word “tomato”.

We are aware that in some cases, the matching word for a specific input conflicts with what someone would expect. An example of such a conflict is found within the EMNLP dataset where an expected normalization would have us normalize the word “untalented” into “talented.” This expected normalization, as we call it, is the way the dataset expects the word to be normalized to remove errors contained within. It may be the case that, within the context of how the dataset was built this normalization is correct; however our system does not take

context into account and, looking only at the word, this normalization is wrong. Nonetheless, we opt not to go over every entry to make it what we felt it should be.

3.2 Normalization Experiments

We perform five different sets of normalization experiments. The first two are the Text-to-IPA and the IPA-to-Text experiments, to test their respective networks independently. The next three experiments explore variations of our system. They are the IPA-to-IPA experiments, the Text+IPA-to-Text experiments, and the transfer learning experiments.

We will train multiple architectures from the `fairseq` and `transformers` libraries, which we list below. Unless specified, and other than for vocabulary and vocabulary size, every experiment is run using the default parameters for each architecture.

fairseq The networks trained via `fairseq` are:

- Convolutional [17]
  - `fconv_wmt_en_de`
- LightConv [32]
  - `lightconv_wmt_en_de`
  - `lightconv_wmt_en_fr_big`
  - `lightconv_wmt_en_de_big`
- Transformer [30]
  - `transformer_vaswani_wmt_en_de_big`
  - `transformer_wmt_en_de`
  - `transformer_wmt_en_de_big_t2t`

While we did train the `lightconv_wmt_en_fr_big` neural network, we do not include it in the results as it is a carbon copy of the `lightconv_wmt_en_de_big` neural network, and both the parameters and results were identical.

Specific data on the makeup of the seven different neural networks are available in Section B.1 of Annex B.

transformers As mentioned previously, we decided to use the `RoBERTa` architecture for most of our experiments. The experiments we ran while using the `transformer` library and that architecture are:

- Train the network under the default parameters (768d embeddings)
- Train with pre-trained 2d embeddings
- Train with pre-trained 100d embeddings
- Train the networks with the augmented dataset

An important point to note is that the amount of attention heads for the networks has to be a factor of the embedding size. As such, instead of the default 12 attention heads used for the 768d embeddings, 2 heads and 10 heads were used respectively for the 2d and 100d embeddings. The reason those numbers were selected was simply because they were the highest number of heads that fit the above requirements, while still being under the default 12 heads.

Both the 2d and 100d pre-trained embeddings are embeddings trained using `fastText`'s [13] Word Vector system. They both represent each character in our alphabets using vectors with a length of 2 or 100. The neural networks are modified to use those pre-trained embeddings instead of creating embeddings as it learns from the datasets.

3.2.1 Preprocessing for the Test Datasets

As mentioned in the previous chapter, some of the words in the test datasets contain characters that are not found in the normal English alphabet. In addition to that, some of the words were not part of the English-IPA pairs that were originally obtained. Due to these two facts, it is not possible to directly obtain proper IPAs for those entries in the test dataset. To resolve this issue, we opt to not take those entries into account when calculating the metrics for the Text-to-IPA experiments.

In addition to this, some of the expected outputs were capitalized in the original test sets. Since our networks do not take capitalization into account, those entries were changed to lowercase.

Another change was made to the contents of the datasets by removing the entries that were not words. This includes URLs, username mentions, hashtags, and various lone characters such as punctuation signs. As mentioned previously in Section 3.1, some entries contain conflicting expected normalizations, but those entries were kept in the datasets.

Table 3.1 contains the statistics representing which proportion of each dataset was used or ignored.

	Tweet Dataset	EMNLP Dataset	[18] Dataset
Total words	10203	41181	3593
Words without matching phonetics	3825	5109	619
OOV words	667	0	1
Words used for Text-to-IPA	5711	36072	2973
Words used for IPA-to-Text	9536	41181	3592

Table 3.1: Dataset statistics and information on elements used to evaluate the experiments

3.3 Results

3.3.1 Text to IPA

In this section, we present the results obtained with our trained Text-to-IPA neural networks. The tests we run in this section are used to evaluate the performance of the first half of the normalizer, converting the initial OOV English, or dirty inputs, into IPA intermediate forms.

Since there are differences between the two libraries we experiment with, the results obtained are split into different subsections. However, the testing methodology does not vary.

Over the course of our research, improvements were made to the datasets and the IPA extraction algorithm to improve the quality of the training data and to remove erroneous entries. As such, not every experiment presented here was done using the same version of the dataset. Generally speaking, the `fairseq` experiments were done with older versions of the datasets while experiments done with the `transformers` library used the improved versions of the datasets.

As is standard in machine learning, we split our training dataset into training and validation portions. This split is random and reset at each experiment, but remains fixed within each experiment at an 80/20 ratio.

The results for all the experiments are broken down into four statistics: the number of words correctly converted to their IPA form, the number of IPAs generated with mistakes, the ratio of correct predictions to the total number of words, and the average edit distance on IPAs with incorrect predictions.

fairseq

The following three tables, tables 3.2, 3.3 and 3.4, contain the results obtained when generating IPAs using the different test datasets when using the `fairseq` library.

`fairseq` Network	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
fconv_wmt_en_de	565	5146	5.54	2.02
lightconv_wmt_en_de	**1846**	**3865**	**18.09**	**1.97**
lightconv_wmt_en_de_big	544	5167	5.33	2.12
transformer_vaswani_wmt_en_de_big	406	5305	3.98	7.77
transformer_wmt_en_de	427	5284	4.19	2.45
transformer_wmt_en_de_big_t2t	510	5201	5.00	2.04

Table 3.2: Results on the Tweet dataset using the various neural networks

`fairseq` Network	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
fconv_wmt_en_de	917	35155	2.23	3.35
lightconv_wmt_en_de	**3476**	**32596**	**8.44**	**2.62**
lightconv_wmt_en_de_big	1004	35068	2.44	3.34
transformer_vaswani_wmt_en_de_big	984	35088	2.39	4.22
transformer_wmt_en_de	975	35097	2.37	3.59
transformer_wmt_en_de_big_t2t	948	35124	2.30	3.33

Table 3.3: Results on EMNLP dataset using the various neural networks

`fairseq` Network	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
fconv_wmt_en_de	79	2894	2.20	3.17
lightconv_wmt_en_de	**301**	**2672**	**8.38**	**2.67**
lightconv_wmt_en_de_big	97	2876	2.70	3.12
transformer_vaswani_wmt_en_de_big	83	2890	2.31	10.34
transformer_wmt_en_de	88	2885	2.45	4.51
transformer_wmt_en_de_big_t2t	86	2887	2.39	3.23

Table 3.4: Results on [18] dataset using the various neural networks

transformers

The following three tables, tables 3.5, 3.6 and 3.7 contain the results obtained when generating IPAs using the different test datasets when using the `transformers` library.

Experiment	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
Pretrained 100d embeddings	2659	3052	26.06	**1.68**
Pretrained 2d embeddings	0	5711	0.00	N/A
Augmented dataset	498	5213	4.88	2.52
Default Parameters	**3020**	**2691**	**29.60**	1.73

Table 3.5: Results on Tweet dataset using for the `transformers` experiments

Combined results for the Text-to-IPA experiments

When looking at both sets of results, Tables 3.2, 3.3 and 3.4 for the `fairseq` results and Tables 3.5, 3.6 and 3.7 for the `transformers` results, we note that one network in each library outperforms the others in every dataset. They are `lightconv_wmt_en_de` for the `fairseq` library and the `RoBERTa` transformer using the default parameters for the `transformers` library. We claim that fact based on our use of the Lehvenstein distance used to evalute the predictions. Since our goal is to find the best neural network capable of creating pronunciations, we select the ones with the lowest amount of errors and the smallest difference between what was predicted and what was expected to be generated, based on our testing dataset.

Experiment	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
Pretrained 100d embeddings	3484	32588	8.46	2.49
Pretrained 2d embeddings	0	36072	0.00	N/A
Augmented dataset	1705	34367	4.14	3.12
Default Parameters	**4058**	**32014**	**9.85**	**2.43**

Table 3.6: Results on EMNLP dataset using for the `transformers` experiments

Experiment	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
Pretrained 100d embeddings	301	2672	8.38	**2.55**
Pretrained 2d embeddings	0	2973	0.00	N/A
Augmented dataset	205	2768	5.71	2.95
Default Parameters	**357**	**2616**	**9.94**	2.56

Table 3.7: Results on [18] dataset for the `transformers` experiments

When we dig deeper into these results, we find a correlation between the length of an input word and the average edit distance. We can observe this correlation in the aggregate results given in Figures 3.1 to 3.6. The first three figures, 3.1, 3.2 and 3.3 present the relationship between the average edit distance between the generated IPAs an the expected result in relation to the amount of letters present in the input for the `fairseq` models. The remaining three, 3.4, 3.5 and 3.6, contain the same results for the `transformers` models.

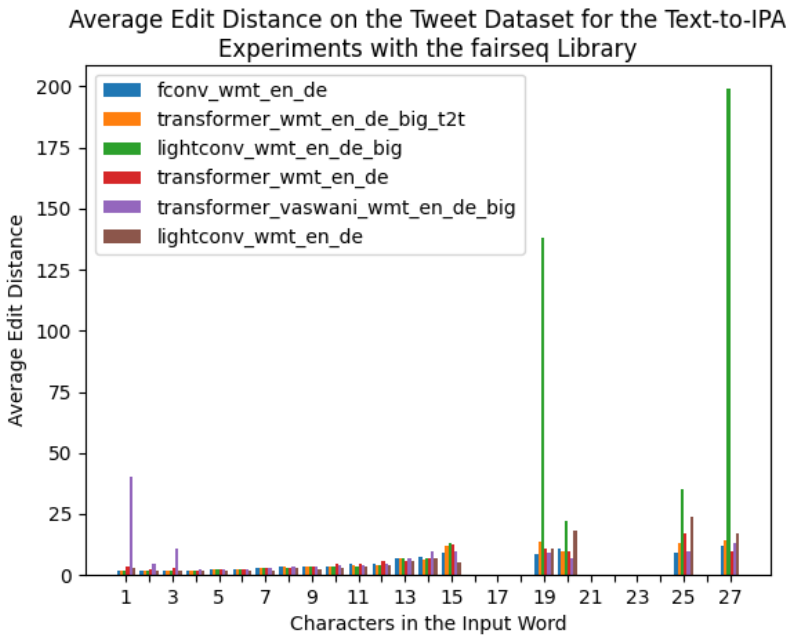

Figure 3.1: Average edit distance based on the input word length for the Text-to-IPA experiments with `fairseq` on the Tweet Dataset

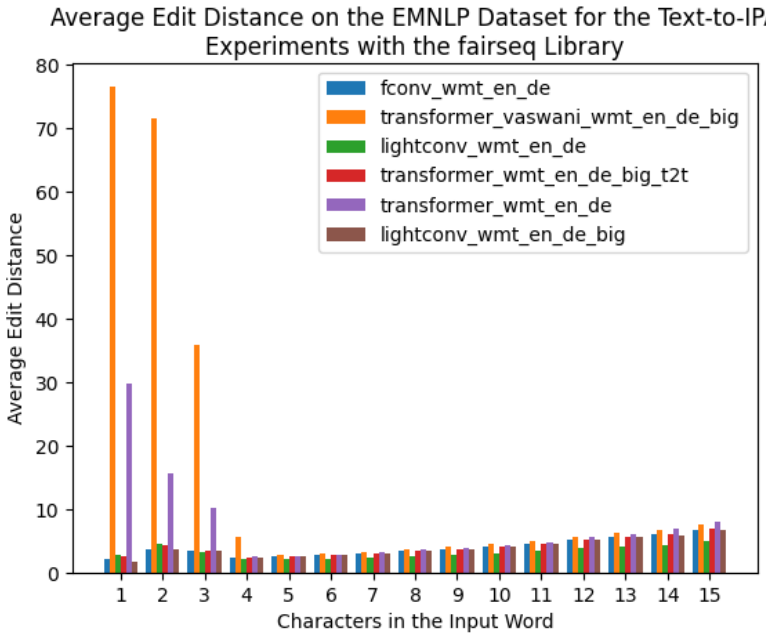

Figure 3.2: Average edit distance based on the input word length for the Text-to-IPA experiments with `fairseq` on the EMNLP Dataset

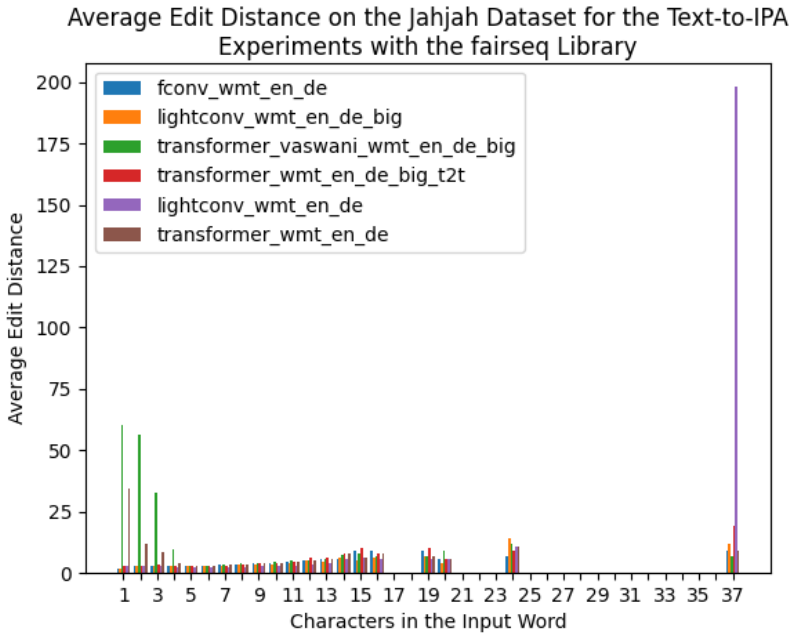

Figure 3.3: Average edit distance based on the input word length for the Text-to-IPA experiments with `fairseq` on the [18] Dataset

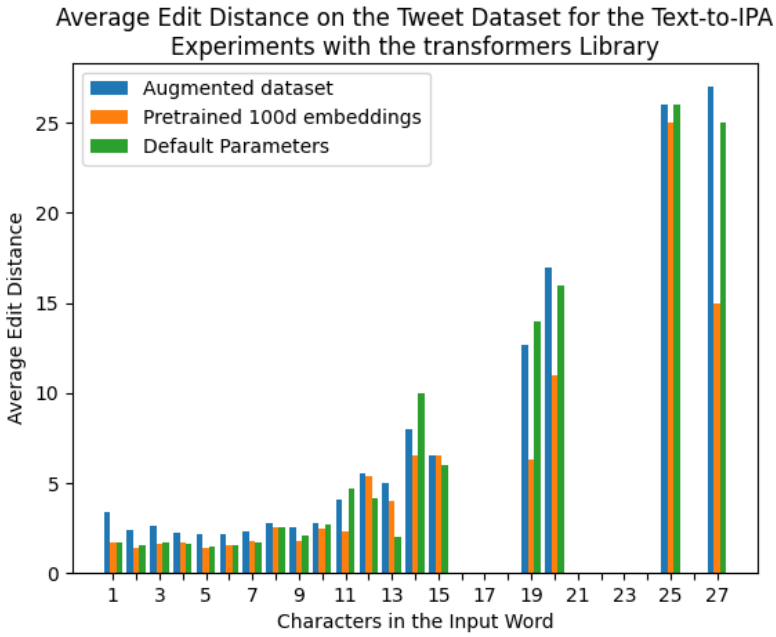

Figure 3.4: Average edit distance based on the input IPA length for the Text-to-IPA experiments with `fairseq` on the Tweet Dataset

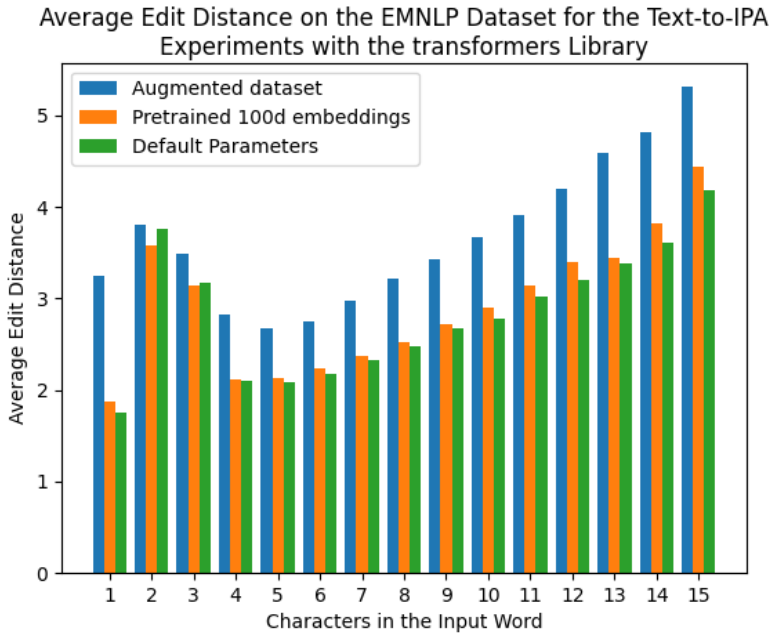

Figure 3.5: Average edit distance based on the input IPA length for the Text-to-IPA experiments with `fairseq` on the EMNLP Dataset

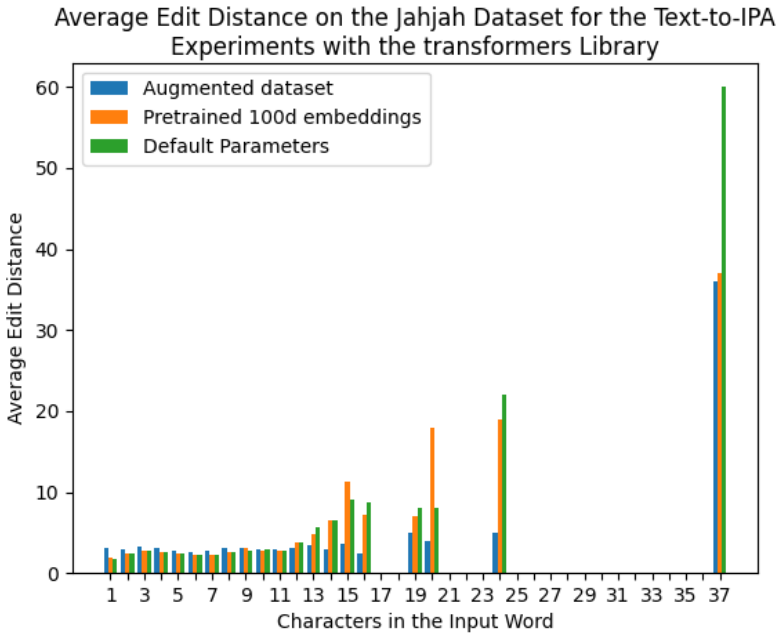

Figure 3.6: Average edit distance based on the input IPA length for the Text-to-IPA experiments with `fairseq` on the [18] Dataset

The increase in the amount of errors contained in the predictions is especially obvious in the EMNLP dataset, as seen in figures 3.5 and 3.2, where the increase in the number of errors is mostly linear for words between 4 and 9 characters, which is where the large majority of words sit, but the average number of errors increases at a non-linear rate outside of that range. Similar observations can also be made when looking at the results of the other dataset. We focus on this expected linearity as we would expect the amount of errors in a prediction to scale with the amount of letters in a word; a word with only two characters should not contain over 20 errors. The same conclusion can be pulled for the longer words, and we believe that this increase is linked to the small makeup of words of those lengths present in the training datasets. The exact proportion of words in relation to their length is available in section A.1 of annex A.

In figures 3.1, 3.2, 3.3, and 3.6 we can observe some odd outliers in the results, namely words with more errors than the amount of letters they contain. This is caused by the absence of predicted start and end of the words, meaning that should the neural network generate an IPA containing a large amount of superfluous data after the end of the intended end of a word, there is no way for it to indicate that anything past a specific character should be ignored. This leads to some of short character IPAs to be generated with a large amount of noise. The same effect can be seen for the words with longer IPAs. This, once more, is a consequence from the underrepresentation of the shorter or longer words in the training dataset. This means that, in theory, the upper bound on the number of errors in a generated word would

be 220 characters, since this is the maximum number of characters allowed in a word.

Improving Text to IPA results

During our experiments, we noticed a fairly high amount of noise within the generated IPAs. In order to try to correct those IPAs, we decided to introduce a third network in the middle of our normalization architecture. We intended it to be an IPA-to-IPA network, which would correct erroneous outputs from the Text-to-IPA network before inputting them to the IPA-to-Text network.

We implemented this network using the RoBERTa architecture and trained it using pairs of incorrectly-generated IPAs and their correct targets. Unfortunately, the network failed to learn to correct the IPAs, and only ever produced IPAs worse than those given as an input. For this reason, we quickly rejected this idea.

3.3.2 IPA to Text

This section presents and analyzes the results obtained when adding the second half of our system, the neural networks capable of reconstructing an English words from the IPA generated by the Text-to-IPA networks.

Similarly to the Text-to-IPA results, the results in this section will be separated by library.

An important point to note for the results presented here is that the IPA used as a source for the inputs of the IPA-to-Text neural networks are the IPAs generated by the best performing Text-to-IPA neural network, our `RoBERTa` transformer using the Default parameters. This is done for four main reasons. The first reason is to simulate real-world use of the system, where we would naturally use the best-performing version of each component. The second reason is to allow for words that were not originally scraped from online dictionaries to have a simulated pronunciation when normalizing them back into English. The third reason is that doing it this way allows us to best simulate an end-to-end use of our normalizer and allows us to evaluate the system as a whole at the same time as we test our IPA-to-Text component. Finally, we believe this is an important point to mention because it is a discrepancy from how the IPA-to-Text neural networks were trained, namely, using only clean IPAs scraped from online dictionaries. This gives us our final reason, avoiding the need to retrain our IPA-to-Text neural networks whenever we make a slight change to our Text-to-IPA networks, or whenever we wish to test a new variation of those networks.

Since the IPAs used for testing are generated, the concern with entries without proper IPAs raised in the Text-to-IPA section can be ignored and every entry was used in measuring the different metrics.

As in the previous section, the experiments were done on different versions of the datasets

over time. The experiments presented in this section are paired with the corresponding experiment presented in the Text-to-IPA section, apart from the `fairseq` neural networks where only the best performing network, `lightconv_wmt_en_de`, was reused, as we deemed the other underperforming networks to not be worth retraining.

Similarly to the Text to IPA results, all results are presented using four statistics: the number of IPAs correctly converted back to English, the number of words with mistakes, the ratio of correct predictions to the total number of words, and the average edit distance of the incorrect predictions.

fairseq

The following table, Table 3.8, contains the results obtained when generating words using the `fairseq` library's `lightconv_wmt_en_de` network. This network is the one found to give the best results in our previous experiments in Tables 3.2, 3.3 and 3.4. Shortly after setting up this experiment, we made the decision to switch to the `transformers` library instead. Consequently, we only trained a single `fairseq` network to completion. The results from the training are presented in the following table, table 3.8.

Dataset	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
Tweet Dataset	2383	7153	23.36	2.44
EMNLP Dataset	5239	35942	12.72	2.47
[18] Dataset	404	3188	11.24	2.79

Table 3.8: Results obtained when training the lightconv_wmt_en_de network on the IPA-to-Text task

Unfortunately, the results themselves are not good. While the results obtained with the IPA-to-Text neural network are slightly better than those obtained on the equivalent Text-to-IPA neural network, they do not achieve our goal of recreating English words from their phonetics with a sufficient accuracy.

transformers

The following three tables, tables 3.9, 3.10 and 3.11, contain the results obtained when generating a word from an IPA using the `transformers` library.

Since only the Text-to-IPA neural networks can use the Augmented dataset as input and no IPA is modified in augmenting the dataset, there is no need to train an IPA-to-Text network on the augmented dataset, as it would give the same results as the default `RoBERTa` model. As such, only the default `RoBERTa` network and those using the pre-trained embeddings are trained for this task.

When observing the results, we once again see that the default network and the 100d network have similar results and both perform better on different aspects, sometimes offering a lower

Experiment	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
Pretrained 100d embeddings	3596	5940	35.25	**2.20**
Pretrained 2d embeddings	0	9536	0.00	N/A
Default Parameters	**4046**	**5490**	**39.66**	2.35

Table 3.9: Results on Tweet dataset using the `transformers` experiments

Experiment	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
Pretrained 100d embeddings	**6034**	**35147**	**14.65**	2.35
Pretrained 2d embeddings	0	41181	0.00	N/A
Default Parameters	5999	35182	14.57	**2.30**

Table 3.10: Results on EMNLP dataset using the `transformers` experiments

Experiment	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
Pretrained 100d embeddings	**456**	**3136**	**12.69**	2.66
Pretrained 2d embeddings	0	3592	0.00	N/A
Default Parameters	445	3147	12.39	**2.61**

Table 3.11: Results on [18] dataset the `transformers` experiments

overall accuracy in favor of a lower average distance or the opposite. Once again however, in a similar fashion to the Text-to-IPA results in Section 3.3.1, this type of insight does not allow us to identify the best neural network.

In a similar fashion to the Text-to-IPA results, we explore the reasons for those low results. However, unlike with the Text-to-IPA networks, we cannot find a correlation between the number of errors per input length and the proportion of IPAs with that length making up the training dataset. This lack of correlation can be seen in figures 3.7, 3.8 and 3.9 for the `fairseq` experiments, and figures 3.10, 3.11 and 3.12 for the `transformers` experiments. We believe that the low performance is rather due to the lack of variety in output words compared to input IPA strings found in the test sets. Indeed, multiple input IPA strings, resulting from different misspellings and alterations of a word, all correspond to the same output word. This makes training the network to recognize the pronunciation of words very difficult. In a similar fashion to the Text-to-IPA results, the exact proportion of the word length in the datasets can be found in Section A.2 of Annex A.

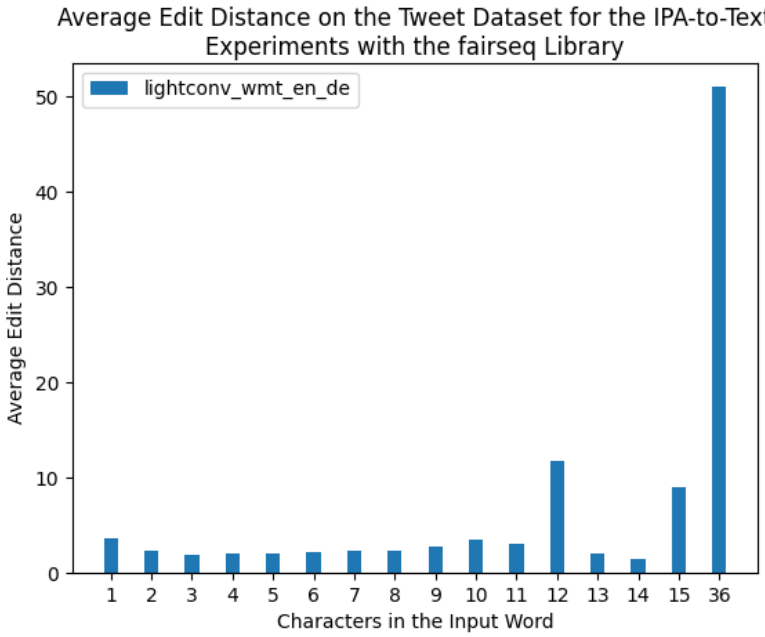

Figure 3.7: Average edit distance based on the input IPA length for the IPA-to-Text experiments with `fairseq` on the Tweet Dataset

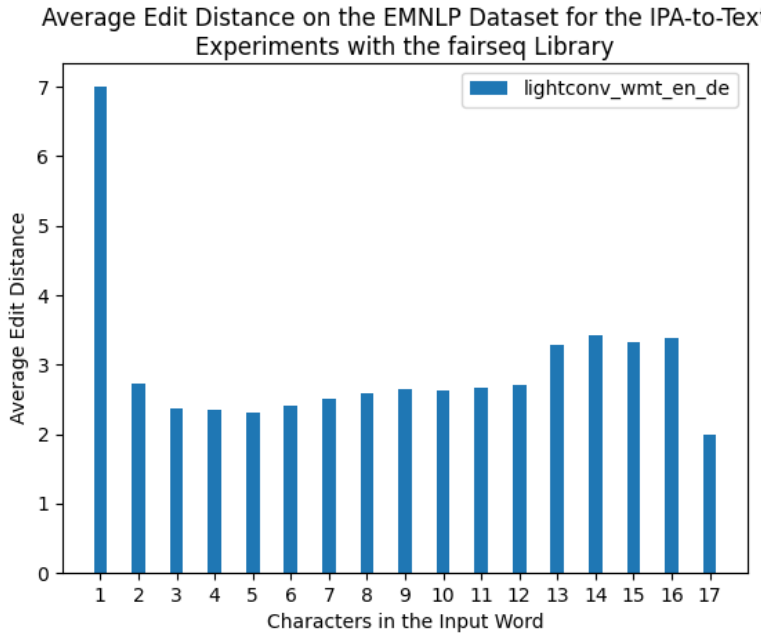

Figure 3.8: Average edit distance based on the input IPA length for the IPA-to-Text experiments with `fairseq` on the EMNLP Dataset

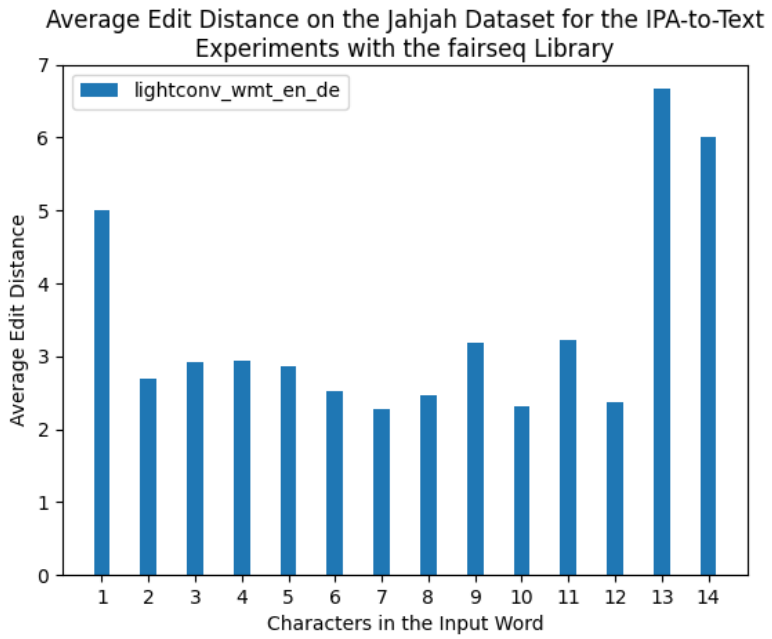

Figure 3.9: Average edit distance based on the input IPA length for the IPA-to-Text experiments with `fairseq` on the [18] Dataset

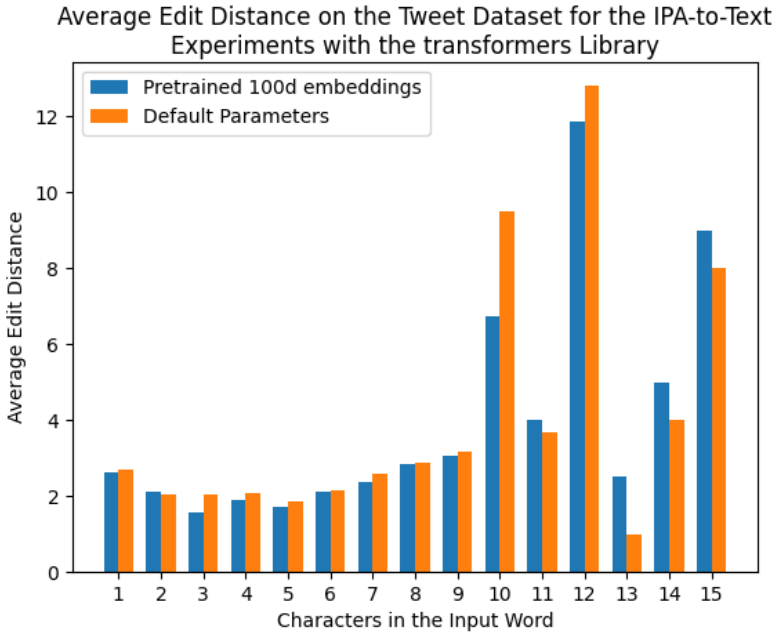

Figure 3.10: Average edit distance based on the input word length for the IPA-to-Text experiments with `fairseq` on the Tweet Dataset

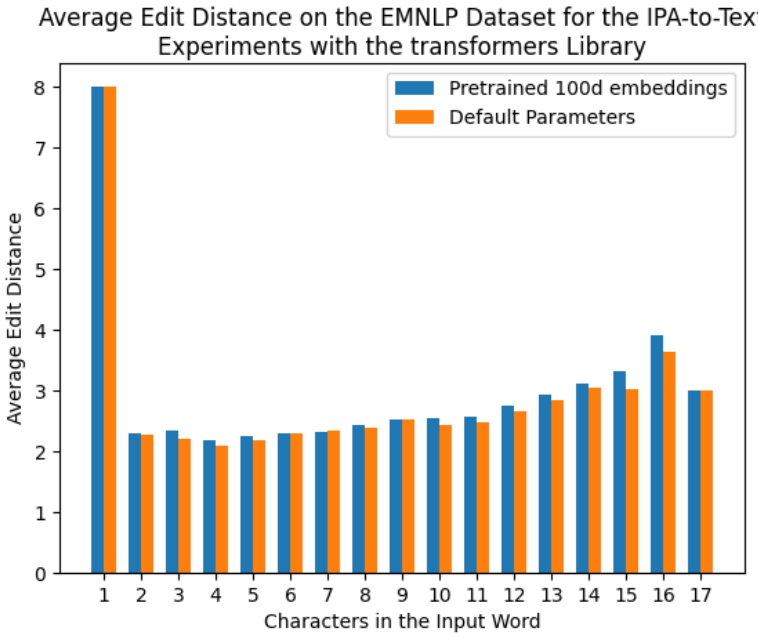

Figure 3.11: Average edit distance based on the input word length for the IPA-to-Text experiments with `fairseq` on the EMNLP Dataset

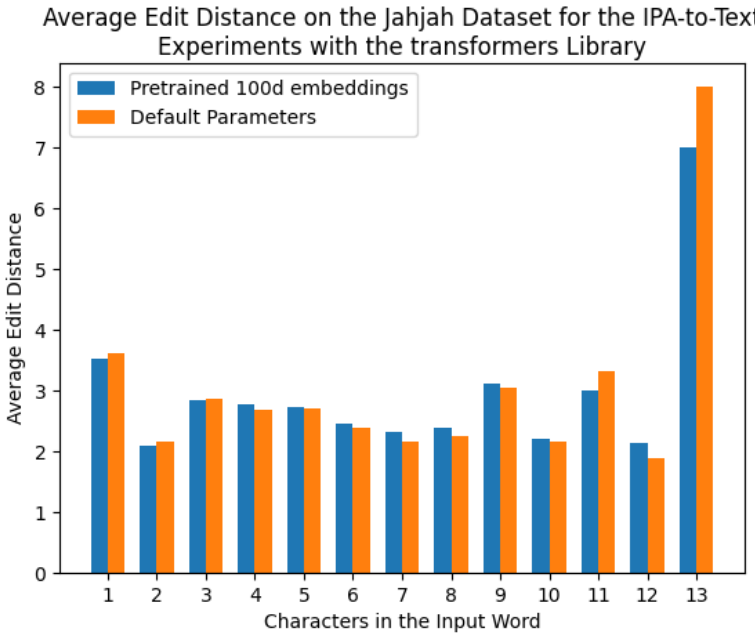

Figure 3.12: Average edit distance based on the input word length for the IPA-to-Text experiments with `fairseq` on the [18] Dataset

Not unlike the results in Section 3.3.1, we can observe words with more errors than letters in

figures 3.7, 3.8, 3.9, 3.10, 3.11 and 3.12. This is caused, once again, by the lack of a signal to mark the start and the end of a generated word.

In addition, at least a portion of the errors of our system are due to the odd and impossible expected normalizations found in the test dataset, which we discussed previously. Without the context of the message they were found in, which we do not have access to and do not use in our system, some of the normalizations in the datasets are impossible to resolve correctly.

Finally, it is important to highlight the fact that the generated IPAs used as input are quite noisy and incorrect. This naturally makes it difficult for the IPA-to-text network to correctly recognize the corresponding English words. The fact the network was trained using correct IPA strings only compounds this problem.

The results are still significant when it comes to selecting the best performing IPA-to-Text network, which, once again, is the `RoBERTa` transformer with default parameters. This network outperforming every other network both in average edit distance and correct predictions.

3.3.3 Text + IPA to Text

In order to obtain better results when converting from phonetics to English, we consider using both the input word and generated IPAs as input. This was done by combining the vectors of the word and the IPA and feeding this combined vector to the neural network. In order to implement this, the tokenizer was modified to create a combined index that represents the paired characters to create a new set of tokens. An example of the combination can be seen in Figure 3.13. This network was trained using the Augmented dataset.

This approach initially seemed fruitful, as the network trained well and gave positive results on the validation dataset, reaching an average edit distance of 0.0054 within the first epoch. However, performance degraded on new test data, as can be seen in Table 3.12. We believe that this is due to overfitting caused by including in the input an encoded form of the output we desire. Taking the example presented in Figure 3.13, we see that the different letter pairings all give a specific tokenized value once merged. Our goal was to allow the neural network to learn those pairings and identify frequent English-IPA pairs that would then be normalized back into proper English. What we found during testing is that the network learned to make predictions based on the English characters alone, ignoring the IPA information entirely. We believe the issue is that, instead of learning the normalization task, the network learned to decode the merged tokens and recover the English tokens they encoded. This in turn made it good at recognizing words it trained on, but incapable of handling unseen words, even those very similar to words it trained on. The exceedingly high accuracy of the Tweet dataset when compared to the others, or any result obtained in this chapter, is explained by the buildup of the dataset, where the great majority of the content is expected to be normalized as the unormalized input itself. This, in part, confirms our theory that there is an overreliance on

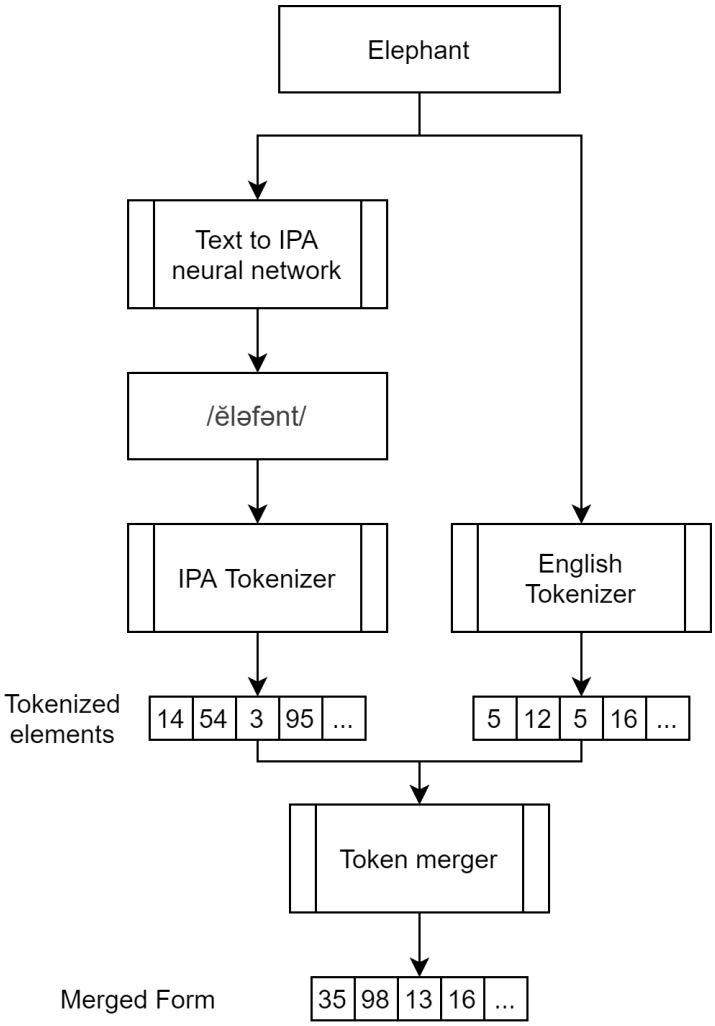

Figure 3.13: Tokenization of English-IPA Character Pairs for Normalization

the encoded input values.

Experiment	Correct Predictions	Incorrect Predictions	Correct Prediction Ratio %	Average Edit Distance
Tweet Dataset	6683	3519	66.51	0.66
EMNLP Dataset	78	41103	0.19	1.62
[18] Dataset	2	3591	0.055	2.13

Table 3.12: Results on EMNLP dataset using for the `transformers` experiments

We also considered different variations of this architecture. One alternative would have used the two tokenized vectors as two separate inputs to the system. However, given the overfitting issues observed with the merged token vector and the fact splitting the vectors would only make it easier for the network to overfit to the English characters, that idea was dismissed. Another design relied on increasing the number of variations and misspellings in the training vocabulary to improve the network. However, as we observed no difference in the network's

treatment of correct and mistaken words - it relied on the English tokens in both cases - we also discarded that idea.

However, we should note that the fact this network was unable to normalized unseen words using text information, while the IPA-to-text network was capable of doing it, demonstrates that there is valuable information in the IPAs and validates the fundamental hypothesis of this research.

3.3.4 Transfer Learning

As was mentioned before, in addition to typos and grammatical errors, online texts can often use characters and symbols outside of the regular alphabet. Given a network that would be able to generate phonetics for normal clean words, like the ones in the training corpus of the Text-to-IPA experiments, it should be possible to alter the models and use them for transfer learning by using the augmented dataset.

To do so, we took the best-performing models from the Text-to-IPA experiments and froze them, put in a new embedding layer and a new output layer, and retrained the output layer.

Unfortunately, after over a week of training, the network failed to converge and the experiment was considered a failure.

The most likely issue is the increased size of the input vocabulary, which went from 100 characters in the previous version to over 900 characters in this experiment. This massively-increased input size, combined with the frozen layers, seems to have made it impossible for the network to rebuild the known associations between characters, since the weights tied between the embedding layer and the first layer were erased when the first layer and the original embedding layer were removed.

Future work may explore variations of this idea, for example by tweaking which layers are frozen in order to facilitate retraining. However, this exploration was judged beyond the score of our research.

Since the augmented dataset is built from a list of substitutions, it may be possible to build initial embeddings for the substituted characters that were originally outside of the original un-augmented alphabet. For example, should the letter "a" be replaced by the character "α", we could rebuild an initial embeddings dataset by copying the embedding the neural network had originally created for the letter "a" and using it as the original embedding of the character "α", and doing the same for each of the newly added characters. Doing so could help jumpstart the transfer process by allowing the neural network to keep altering the embedding layer to obtain better vectors for the new characters. This would not be a perfect solution in cases where a single character is replaced by multiple characters, or in cases where a known character is substituted by another known character, but it may resolve some of the

issues causing our experiment to fail due to a lack of learning, should it not cause a large amount of overfitting if the neural network fails to alter the embeddings enough.

3.4 Conclusion

In this chapter we studied the results of the normalization experiments we conducted with the systems we presented in the previous chapter. Our results show that the networks trained with the `transformers` library edged out the other networks. Even though the performance was not as good as we had hoped, it still demonstrates that there is merit to our hypothesis that phonetic information can be used to understand and correct poorly-written text. In the next chapter, this normalization system will be applied to a real-world challenge, namely harmful speech detection.

Chapter 4

Harmfulness Detection

As we covered in the Introduction of this thesis, misspelled words online do not only originate from honest errors or attempts at personalizing messages. Some ill-intentioned online users misspell harmful words and messages to disguise them and circumvent harm filters; a process known as subversion [14]. One of the objectives of our work is to determine if the use of phonetic normalization can help counter this attack.

In order to ascertain whether or not our normalization system can be used to help improve harmful speech detection systems. By correcting misspellings in online messages, in particular those due to subversive harm, our system could improve the quality of the inputs given to the systems charged to classify the messages into different types and levels of harmfulness. In this chapter, we will cover the experiments we designed as well as the harm classifiers we created to verify the use of the normalization system. Throughout both this chapter and chapter 5, the use of the terms `classifier` and `harm classifier` will be used interchangeably to represent harmful speech detection systems as a whole.

4.1 Methodology

As we presented in chapters 2 and 3, our best performing normalizer was the one made up of two `RoBERTa` neural networks. In order to properly test how this normalization system can benefit a harm detection system, we devised two sets of tests. The first is the *classifier input test*, which will measure the benefit of normalization when different types of input are given to a harmfulness classifier. The second set of tests is the *subversion test*, and it evaluates the effect of the normalization on harm classification given different levels of subversion in the messages before normalization.

We believe that combining those two sets of tests will allow us to properly identify the conditions in which our normalization system will perform best.

4.1.1 Dataset

The dataset we opt to use in order to train our classifiers is the dataset provided for the Kaggle competition created by Jigsaw [3] which was aimed specifically at harm detection.

This dataset is split into two subsets, a training and a testing dataset, respectively containing 159,571 and 153,164 entries. Due to the way the test set labels are set up, only 63,978 test entries are actually usable to validate the training and compare the networks. This loss of around 90,000 test entries is because those entries do not have proper labels, all of them being set with a value of "-1". It is explained on the competition page that those entries were simply not used for scoring, and as such, no proper labels were provided. The entries in both subsets are made up of comments and their labels, or classes.

Both datasets are labelled with up to six classes representing different types of harm. Those labels are `toxic`, `severe_toxic`, `obscene`, `threat`, `insult` and `identity_hate`. To those labels, we add a seventh, `has_toxicity` which represents the presence of potentially harmful content. This label was added to identify comments which contain harmful elements, but are not inherently harmful. An example of such a comment would be "OH DEAR OH DEAR, HOW FUKIN SAD. I WILL REGRET THIS FOR THE REST OF MY LIFE!!" where the original commenter used the word "FUKIN" which led the comment to be classified as `obscene`, but not `toxic`. On the surface, such a label is not extremely useful as it can be inferred from the other classes. However, we see it as a way to measure wether the normalizer is able to properly normalize messages or if it is creating data that the classifiers cannot analyze properly. It is also used to help us identify cases where the existence of harm is not lost through normalization, but where the specific type of harm would be.

4.1.2 Training datasets

We have a total of four types of inputs we used to train the different neural networks. They are the original dataset provided in the Kaggle Competition, along with the normalized, phonetic, and combined datasets derived from it.

In order to build the three derived datasets, we compile a list of every unique word found in the original dataset. In order to find unique words, every comment is tokenized with [12]'s `TweetTokenizer`, as mentioned in Section 4.1.6.

We create our phonetic input dataset by replacing each word in a message with its phonetic equivalent, as determined by the text-to-IPA portion of our normalizer. Then we created our normalized input dataset by replacing each phonetic word in the phonetic input message with its normalized equivalent, as determined by the IPA-to-text portion of our normalizer. Punctuation signs, numbers, and other special characters in the original message were left unchanged in both transformed versions. Finally, the fourth dataset is obtained by pairing

together both the phonetic and normalized messages.

A limitation of our normalizer is that it can only handle words up to a maximum length of 62 characters. Consequently, longer words were excluded from the process described above and left "as is" in the transformed messages. These longer words are jumbles of characters and long URLs, so they should not affect our experiments.

4.1.3 Classifier Input Types

In order to evaluate the effectiveness of normalization on classification results, we designed four different groups of neural networks, each one trained on one type of input. Each network will thus provide a different insight:

- Unmodified input: Classifiers trained using this input type will be used as our baseline for each experiment.
- Normalized input: The use of normalized data as an input for a classifier is meant to represent systems that were built using our normalizer as a pre-processing step.
- Phonetic input: In chapter 2 we described how we generate an alternate normalized form of words, their IPAs. We will train neural networks using these normalized IPAs as input instead of the normalized words. These classifiers will allow us to test if it would be better to only use phonetic version of the words, and whether the conversion of phonetics back to English is beneficial for harm detection.
- Both inputs: Finally, we decided to train classifiers using both normalized and phonetic forms of the words as input, to see if it might improve classification results.

Taken together, the results of the experiments using these four different input types will help us properly evaluate the usefulness of our normalizer for this task. Comparing the results of the experiments with the first two types of inputs, unmodified and normalized, will allow us to evaluate the effectiveness of the normalizer, and the additional two types will allow us to evaluate whether additional useful information can be gleamed from the pronunciation of a word.

In total, we train eight neural networks for these experiments, namely four using the `Bi-LSTM` architecture and four using the `RoBERTa` architecture.

4.1.4 Normalization Test Datasets

In order to test how much the normalization improves the performance of harm classification, we devise a series of alterations to apply on the original test dataset to simulate the use

of subversive harm. In total, we have fifteen dataset versions, five unormalized ones, five phoneticized and five normalized ones. These datasets are as follows:

- Original test set: This is the original unaltered test set. It contains some elements that could already be seen as subversive, such as the removal of some letters. This dataset gives us our baseline performance evaluation. It contains various levels of harmful content and subversive elements which were not altered or removed.
- Substitution test set: This dataset is built by applying letter substitutions to the words in a message. More details are given in the subsection below.
- Letter removal test set: This dataset is created by randomly removing one letter from each word found in a message.
- Letter duplication test set: Similarly to the previous dataset, this dataset is created by randomly duplicating one letter from each word found in a message.
- Combined subversion dataset: This dataset is built by taking the substitution test set and randomly adding or removing a random letter from each word.

Each of these five datasets was then run through our normalization system to create a normalized dataset and a phoneticized dataset. Examples of the altered comments in the new datasets of some comments can be found in table 4.1.

It is important to note that the three versions of the datasets, meaning the unnormalized version, the phoneticized version and the normalized versions are different from the five types of datasets presented here. Each dataset has three different versions of itself.

Due to how we seeded and used randomness during dataset creation, removal and duplication of letters occurs for the same characters in both datasets. A similar behaviour is seen in the combined dataset. However, the letter affected by the removal or duplication diverges due to the added randomness in selecting the transformation, and the potential presence of a substitution in a word.

As can also be gleaned from the data in table 4.1, we do not specifically target words which are harmful, or which could be part of a harmful conversation. We opted not to target words that are harmful for two reasons. The first being that the context surrounding a word is often very important in determining harm, and the second reason being that, since some of the messages contained in the dataset were not all written in proper english, we believed that adding subversion throughout the entire messages would be a better test of our normalizer since we built it to be agnostic of the context it is used in, both by design, and as a consequence of it normalizing messages word by word.

Dataset	Message
Original	Thank you for understanding. I think very highly of you and would not revert without discussion.
Substitution	Thank you for understanding. I think very highly of you and wld not revert without discussion.
Letter Removal	Thak yu fr undersanding. I tink vry hghly of yu ad wold nt revet wihout discussin.
Letter Duplication	Thannk yoou foor understtanding. I thhink veery highly of yoou annd wouuld noot reverrt witthout discussioon.
Combined	Thannk yoou fr understaning. I thiink vry highhly of yu ad wd nt rvert wthout discussioon.
Original Phoneticized	næk ju fɕ ɑndərstændɪ. aɪ nɪk vɛri haɪli ɑf ju ænd wudd nɑt rɪvɜrt wɪnhut dɪskɑfən.
Letter Removal Phoneticized	næk ju fr ɑndərsændɪ. aɪ tɪk vri hflɪ ɑf ju æd would nnt rɪvɛt wɪhaut dɪskɑsɪm.
Letter Duplication Phoneticized	nækk ju fɕrr ɑndərstændɪ. aɪ nɪk vɥɪ haɪgi ɑf ju ænd wud d nut rɪvɜrt wɪthaut dɪskɑfən.
Substitution Phoneticized	næk ju fɕ ɑndərstændɪ. aɪ nɪk vɛri haɪli ɑf ju ænd wul nɑt rɪvɜrt wɪnhuu dɪskɑfən.
Combined Phoneticized	nækk ju fr ɑndərsteɪɪ. aɪ nɪɪk vri haɪlii ɑf ju æd udd nnt rvɜrt hʉhaut dɪskɑfən.
Original Normalized	thank you for understanding. iye think very hilhey off you annl woodeed not revert withhoot discussion.
Letter Removal Normalized	thack you fre undersanding. eye tink vrie hshley off you ad wold kent revet wihhout discussin.
Letter Duplication Normalized	thankk you foure undherstanding. eye thinnk veery hyggi off you andd wooed d nootka revert whithoutt discussion.
Substitution Normalized	thank you for understanding. eye think very higey off you andd wool not revert withhooo discussion.
Combined Normalized	thankk you fre understating. eye thienk vrie hilhl off you ad uddled kent rvert hahehott discussion.

Table 4.1: Examples of alterations and normalization on the comment with id `0001ea8717f6de06`[1]

The normalized versions of the datasets were created by using the same process we explained in Section 4.1.2.

Creating the datasets

In order to create the substitution test set, special consideration has to be taken in order to avoid exponential growth in the size of the dataset. Since we use the same substitution

[1]We opted to only include the first subversive transformation of the comments for both the Substitution and Combined Subversion datasets to not overfill the table.

list we described in Section 2.3.2, we would end up with an absurdly large and meaningless dataset if we applied every possible substitution. Instead, we define a substitution group as a list of substitution pairs for an original group of letters, and we define a substitution pair as the mapping between a letter group and the letter group with which it is getting replaced. An example of a substitution pair would be the pairing between “ack” and “ak”, and a substitution group would be the group made up of the four substitutions found for the letter group “ack”, which are presented in Table 4.2.

Letter Group	Substitution
ack	ak
ack	ck
ack	k
ack	ac

Table 4.2: Substitution group for the letter group ack"

To resolve this issue, we opt to randomly select one substitution group present in each message and create new versions of the message with every substitution we have for that group. Each new version will apply one of the substitutions, but it will be applied to all instances of the group found in the message. We will thus end up with as many new messages as there are substitutions for the group.

A second reason for creating fewer substitutions in this dataset, compared to what we did for the augmented dataset in Section 2.3.2, is that the best-performing normalizer we trained was only trained on English words found in a dictionary. Consequently, an overwhelming number of substitutions would hinder its performance. We considered the idea of creating multiple datasets with varying amounts of substitutions, but we opted against it for the same reason.

A similar concern exists for the letter removal and letter duplication datasets. To avoid creating large datasets, we opt to only apply one transformation at random per word in each comment, giving new datasets of the same size as the original test set. Out of concern for the integrity of the words, the first and last letters were always left untouched.

The final dataset, the combined subversion dataset, is created by combining these approaches. First the substitution rule is applied to the message to create new messages. Then we randomly pick a character to duplicate or remove in each word of the new message. The same limitations used for the substitutions and the character duplication and removal are also applied here.

4.1.5 Harm Detection Systems

For our experiments, we will train two different harm detection network architectures. The reasons are to validate that our results are not somehow tied to one specific network architecture, and that our normalizer can benefit multiple different harm detection systems.

The first network is based on the most popular kernel offered on the same Kaggle competition that the dataset is from[1]. This network is composed of two Bi-LSTM layers followed by two linear layers on which a ReLU function is applied, and a final linear layer used for classification.

The second network is a `RoBERTa`-based classifier using the `transformers` library. We opt to use it because the `RoBERTa` network gave the best results in our previous experiments in Chapter 3.

In both cases, the neural network was trained from scratch with no pretraining for each type of input.

4.1.6 Tokenization

Tokenization is an important part of preprocessing text for its use in natural language processing. In order to properly run our experiments, we used three different tokenizer algorithms. The first one, `TweetTokenizer` from [12], was used when creating the various datasets. The other two, the `Keras` tokenizer from [15] and the `huggingface` tokenizer[2] from [31], were used respectively by the `Bi-LSTM` and `RoBERTa` neural networks to process the neural network inputs.

We opt to use `TweetTokenizer` for the task of tokenizing the dataset because it was designed to tokenize tweets, a form of online communication akin to the short messages found in our datast. It also tokenizes the messages into words, unlike other tokenizers that may split suffixes and prefixes from the root of a word. This type of tokenization allows us to get words that are similar to the data we originally trained the normalizer with. Once the words are obtained from the messages, we phoneticize them to obtain the phoneticized list of words. Due to how our normalizer works, some phoneticized words have spaces in them. While our normalizer is only trained to deal with one word at a time, we opt to treat those space-separated phoneticized words as multiple words. A list of unique phoneticized words was built from the generated phonetic words, and each of these unique phonetic words was transformed back to English using our normalization system.

While we can use the `TweetTokenizer` to build the datasets, neither of the networks we use as harm detectors originally used this tokenizer. In the case of the `Bi-LSTM` networks, we need to train the `Keras` tokenizer on the dataset whenever we run an experiment. Even if it was retrained every time, this tokenizer was constant for each dataset. This gives us three distinct tokenizers, one for the original dataset, one for the phoneticized dataset, and one for the normalized dataset. The `RoBERTa` neural network, on the other hand, uses its own tokenizer, the `huggingface` tokenizer, which was pre-trained on a dataset different from ours.

[1] `https://www.kaggle.com/thousandvoices/simple-lstm`
[2] `https://github.com/huggingface/tokenizers`

We opt to use each network's original tokenizer to stay as close as possible to the original design and functioning of the neural networks.

Typical behavior for a tokenizer that encounters an unknown word is to either mark that word as unknown or to remove it. The `Keras` tokenizer removes unknown tokens. However, the `huggingface` tokenizer does not mark unknown tokens as such, but instead attempts to split them into multiple known sub-tokens. For example, should the token "beginninging" be seen, it would get split into both "beginning" and "ing", if it is seen as a valid way to split the token. If it is not, the token is marked as unknown.

We do, however, in the case of the `RoBERTa` neural networks, make a change to how the `huggingface` tokenizer is used. Instead of using the pretrained tokenizer, we train a new tokenizer for each of input type. This means retraining the English tokenizer, in addition to the training the normalized and phoneticized tokenizers. This was done to have an equivalent tokenizer for each type of dataset. We deem this to be a necessary step because the default English tokenizer provided in the `tokenizer` library might be aware of additional vocabulary. This additional vocabulary leads to a tokenizer containing more unique tokens than what is contained in our original dataset. Combined with the token-splitting behavior, the original tokenizer might have biased the results of the subversion tests in favor of the neural networks making use of it when compared to the phonetic, normalized or hybrid models. To avoid any issues, we decided that training a new tokenizer using only our original dataset is the best solution to avoid bias.

In addition to this tokenization, every comment is left-padded up to a certain length. In the case of the Bi-LSTM, the comments are padded up to 220 tokens, the original value used in the Kaggle kernel. The `RoBERTa` neural net is padded up to 150 tokens. This is due to a technical limitation, namely the lack of memory in our GPUs. We should note that the average and median number of tokens per message in our datasets are 90.6 and 50 respectively, so the difference between a maximum input length of 150 or 220 tokens has little impact in practice.

4.2 Running the Tests

Putting it all together, the next chapter will present results using our 5 different datasets and their three different versions different used in the training and testing of eight neural network variations, covering two different neural network architectures and the performance of our normalizer accross those two.

In order to avoid spending excessive time training eight neural networks, we did not train them until they converged to a local maximum, but instead stopped the training after 100 epochs. As the graphs found in Section 5.3.1 present, this number of epochs was likely excessive.

Unless specified otherwise, all networks were trained for that number of epochs, to keep the comparison fair.

4.3 Conclusion

In this chapter, we presented our test cases to evaluate the usefulness of our normalizer on the task of subversive harm detection. We designed a variety of tests that will allow us to evaluate the quality of harm detection using the normalized messages, and tests that evaluate which normalized data type is most useful. We believe we have created a complete test suite to validate the performance of our normalizer for our purpose of subvertive harm detection. The next chapter will, using the tests laid out in this chapter, present the results obtained and offer our conclusions based on those results.

Chapter 5

Harmfulness Detection Results

5.1 Introduction

This chapter presents the results of the experiments described in the previous chapter. It will also present our conclusion as to whether or not our phonetic normalizer can help improve the results obtained by a harmful message classifier, and answer the broader question of the influence of phonetic normalization on an increased amount of subversive elements in messages.

5.2 Network Architecture and Data Processing

This section covers the pre-processing of the data and the alterations we made to the default architectures in order to use the different input types.

As mentionned in Section 4.1.6, we use different tokenizers for each of the architecture type. In the case of the `Bi-LSTM` models, we use the `Keras` tokenizer, and we use the `huggingface` tokenizer for the `RoBERTa` models.

5.2.1 Architectural Alterations

In order for the harm detection networks to predict 7 classes per message, as defined in Section 4.1.1, we have to modify their architecture.

In the case of the `Bi-LSTM`, the change is to allow auxiliary outputs for the network. Doing so also allows us to use a Binary Cross-Entropy with Logits loss function, or `BCEWithLogits` loss, to train the neural network. The auxiliary outputs are additional outputs that can be used to gain or predict additional data without affecting the primary output. In our case, the auxiliary output is a single linear layer parallel to our primary output making predictions on the 6 non-primary labels. It is used to allow for independent training of the final output layers. The use of the auxiliary output allows for the use of a Binary Cross-Entropy loss

because we now need to solve a multi-label problem. It would also have been possible to alter the primary output to predict on the 7 classes, but we opted to separate them as our goal was to focus on the broader presence of toxicity and use the additional data to inform our analysis.

The changes we make to the `RoBERTa` network are a little more complicated. We use version 2.6.0 of the `transformers` library, which has no network implementation that supports multi-class or multi-label classification. Consequently, we modified the `forward` methods of the classes to use a `BCEWithLogits` loss instead of the `MSE`, or `CrossEntropy` loss functions. The effective result of this alteration is the same as if a newer version of the library had been used, but we deemed it a better solution than to try and update the library and all of the subdependencies, such as `pytorch` and `CUDA`.

Hybrid Inputs

As we mentioned in Section 4.1.1, we also train neural networks that use both the phoneticized and normalized inputs. The networks that use this input have been updated to receive two different embedding layers, one for each type of input, which are then concatenated together as one larger input embedding. This concatenation was done at the token (word) level. This means that while the tokens may not perfectly line up, either because one token is normalized to multiple tokens or because of how the different tokenizers split the message, the general "sense" of the comments should not be lost when sent to the `Bi-LSTM` or `attention` layers. An example of the process is presented in Figure 5.1. We did not measure the impact of the token misalignment nor take any measures to alleviate its effects. Given that the misalignment can come from either the normalization or tokenization process, or both, we feel that adding functionalities to track it down and correct it would greatly complexify the system.

This concatenation does cause an issue, namely the doubling of the length of the embedding vectors. In the case of the `Bi-LSTM` this meant an increase in the length of the embedding from 300 to 600. To resolve this issue, we shrink the embedding vector back down to 300 in the first `Bi-LSTM` layer, bringing us back to padded vectors of length 220 containing word embeddings with a length of 300. This does not represent a significant increase in the overall number of parameters when compared to the `Bi-LSTM` harm detector that do not use hybrid inputs. In the case of the `RoBERTa` model, however, since the default embedding size is 768, the final embedding size would have been 1536. In that case, we instead generate embeddings with a size of 384 by configuring the embedding layer to have 384 parameters, and then combining them back into a single, larger embedding with the original 768 parameters. We opt to do this because the doubling of the embedding size would have either forced us to rewrite the model to allow for the reduction of the hidden dimension within the attention layers, or use 1536 as our hidden dimension throughout the model, which we saw as way too large of a departure for the 768 used for the other `RoBERTa` models, due to the large number of interim

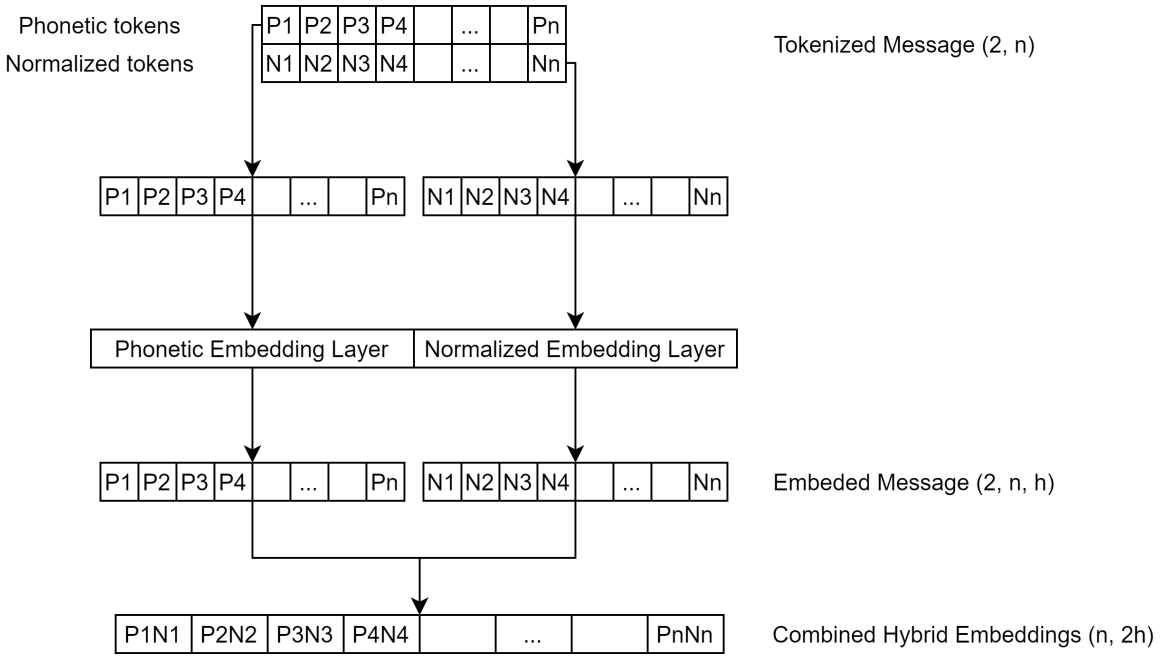

Figure 5.1: Concatenating embeddings for the hybrid inputs

layers when compared to the `Bi-LSTM`, which is only made up of five layers, compared to the 12 layers and 12 attention heads of the `RoBERTa` models.

It would have been possible to use other setups to handle the combined input types besides the ones described above, such as by having a different stack of `LSTM` or `attention` layers and combining the results before classification, or by using a linear layer to reduce the size of the embeddings. However, our approach is appropriate for our goal of showing the effectiveness of our normalization approach for the task of harm detection. Alternatives could be studied in follow-up research, to compare their strengths and weaknesses.

5.3 Results

In this section, we cover the results of both the Classifier Input test and the Normalization Performance tests.

5.3.1 Classifier Input Tests

This section covers the results we obtain when training the two different harm detection neural network architectures with the four different input types. Before going over the results, we will quickly cover how we selected the checkpoints, or epochs, used to obtain those results. Those same checkpoints are also used in the subversion tests. While a checkpoint generally refers to a saved snapshot of a model's weights at a point in time, and an epoch refers to an

iteration of training of the neural network, they are essentially the same in our use case and both words will be used interchangeably.

Neural Network Checkpoint Selection

In order to select the checkpoints that will be used for both tests for each network when trained with each dataset, we use the average loss we obtain on the validation set used during training. The loss for both types of neural networks for every input type can be found in Figure 5.2 for the `Bi-LSTM` neural networks and in Figure 5.3 for the `RoBERTa` neural networks.

As can be observed in Figure 5.2, it is clear that the `Bi-LSTM` network tends to plateau between the 10th and the 15th epoch. In this figure, we only present the first 50 epochs. We did train the network for 100 epochs; however the remaining 50 epochs continue the same plateau and presenting them makes the figure less legible. A similar pattern was observed when looking at the training loss. As can be seen from the same figure, the `Bi-LSTM` classifier trained using only the original training dataset outperforms all three of our alternative input types with respect to the validation loss. Out of our three normalized input types, the best performing type for this class of classifier is the hybrid input.

The exact numerical value of the loss at the best epoch is presented in Table 5.1. Accordingly, the checkpoint that was selected for each neural network is the one with the best loss. This was done for two reasons. The first reason is to avoid performing excessive training that could lead to overfitting. The second reason is that, while we are comparing all the classifiers, each one is actually independent of the others and its performance should be evaluated separately.

Input Type	Best Epoch	Loss at Best Epoch
Default	7	0.06059
Phonetic	11	0.06229
Normalized	11	0.06364
Hybrid	9	0.06160

Table 5.1: Loss at Best Epoch for the `Bi-LSTM` Neural Networks

The validation results for the `RoBERTa` neural network, shown in Figure 5.3, offer a much less clear picture than those for the `Bi-LSTM` neural network. We see that the phonetic inputs performed better than the others. We also note the odd behavior of the loss, which fluctuates instead of converging, and which we will come back to further down. Similarly to the `Bi-LSTM` neural networks, these networks were only trained for 100 epochs, but the results past the 50th epoch follow a similar pattern to what is presented in the previously mentionned figure, and as such only the first 50 epoch are shown. The best loss and associated epoch can be found in Table 5.1. For the first three networks, the epoch selected for the experiments was the one with the best loss. For the fourth network with hybrid inputs, we find that the best loss of 0.16863 at epoch 98 only narrowly beats a few of the earlier epochs by a value of

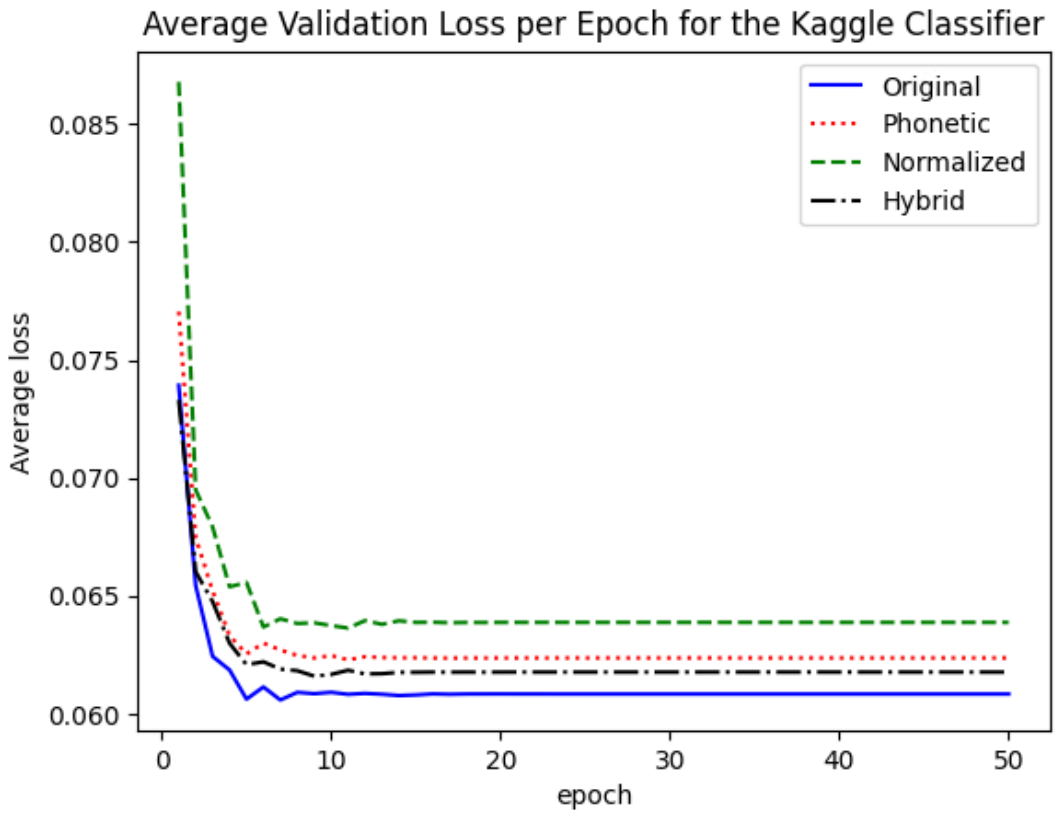

Figure 5.2: Average Loss per Epoch on the Validation set for the `Bi-LSTM` neural networks

0.00001-0.00003, and these earlier epochs have a lower risk of overfitting on the training set. Consequently, the epoch we select is epoch 31 with a loss of 0.16865.

Input Type	Best Epoch	Loss at Best Epoch
Default	2	0.11634
Phonetic	2	0.06618
Normalized	25	0.16153
Hybrid	98	0.16863

Table 5.2: Loss at Best Epoch for the `RoBERTa` Neural Networks

When looking at the validation loss graph of Figure 5.3 and the best loss values of Table 5.4, we see that the behavior of the network differs from what we would normally see when observing this metric. Normally, a neural network would gradually converges to a local minimum, as the `Bi-LSTM` did. We link this odd behavior to the fact that the `RoBERTa` network is designed to train using massive datasets with millions of different entries, while we used a relateively small dataset comprised of slightly less than 160k entries. Furthermore, we performed no hyper-

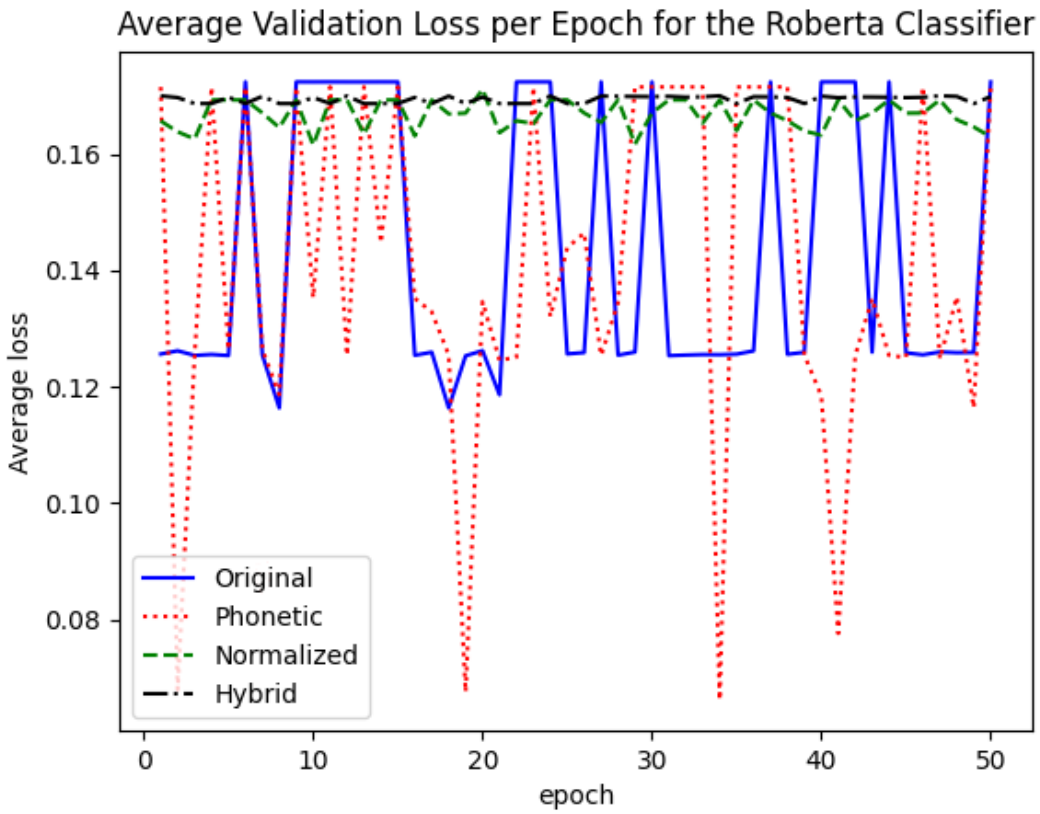

Figure 5.3: Average Loss per Epoch on the Validation set for the `RoBERTa` neural networks

parameter tuning. We believe this led to an almost immediate overfitting on the dataset. This overfitting behavior could be alleviated by reducing our `learning rate`, or by increasing the decay in the `loss scheduler`. However, since our goal is to create a harm classifier to test and evaluate our normalization system with, and not to create the best possible harm classifier, we decided that such hyper-parameter tuning of the classifiers was out of scope.

Looking at the training loss for all of those networks also shows that it stays mostly constant past the first epoch, varying by a few hundredths around the best epoch value, with the only exception being the Phonetic input neural network that sees its training loss go up to 0.16 by the fifth epoch. This means that even though the neural network is able to learn (as demonstrated by its results in table 5.4), it fails to converge, as shown by the combination of the best epoch loss and the behavior of the validation loss in figure 5.3.

Prediction Accuracy Results

This section presents and discusses the accuracy of the predictions made by the classifiers on each of their respective test sets. Since our goal in this first experiment is to evaluate the performance of the classifiers on different input types, and not different levels or types of text alterations or subversion, only the Original test set is used, in the form adapted for each network. This means that results for the Default networks will use the unaltered dataset, the Phonetic networks will use the phoneticized form of the Original test set, and so on.

The accuracy of the `Bi-LSTM` networks is presented in Table 5.3, while the results obtained with the `RoBERTa` networks are presented in Table 5.4.

Input Type	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate	All Valid
Default	91.26	91.91	99.14	95.45	99.67	95.22	98.89	86.20
Phonetic	91.72	92.16	**99.25**	**95.49**	99.67	**95.32**	98.89	86.89
Normalized	**91.81**	**92.27**	99.19	95.47	99.67	95.16	98.89	**87.06**
Hybrid	90.97	91.52	99.16	95.28	99.67	95.04	98.89	85.93

Table 5.3: Accuracy for all labels for the `Bi-LSTM` neural networks

Input Type	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate	All Valid
Default	91.59	91.81	99.43	95.30	99.67	95.19	98.89	**90.68**
Phonetic	**92.50**	**92.53**	99.43	**95.93**	99.67	**95.55**	98.89	88.70
Normalized	90.24	90.48	99.43	94.23	99.67	94.64	98.89	90.24
Hybrid	90.24	90.48	99.43	94.23	99.67	94.64	98.89	90.24

Table 5.4: Accuracy for all labels for the `RoBERTa` neural networks

As we can observe from both result sets, the networks all have extremely similar results, and sometimes identical results as in the case of the identity_hate and threat labels, implying that the "markers", the elements that help identify labels, are distinctive enough for those labels that they are unaffected by the normalization process. Likewise, in the case of the `RoBERTa` network, the normalized and hybrid inputs have identical results, which implies that the phonetic part of the hybrid input was essentially ignored by the harm classifier.

On the other hand, the results show that the `Bi-LSTM` harm classifier achieves better results overall using phonetic or normalized input, and the `RoBERTa` classifier achieves its best results using the phonetic input. This indicates that our phonetic normalizer does slightly improve harm detection given normal online messages as input.

5.3.2 Subversion Tests

Next, we study the effects of our phonetic normalization on the performance of harm classification in the presence of subversion. The results are separated according to the four different types of subversion datasets we have. For each subversion dataset, we further present test

results using each of our two harm detection neural networks, and each network was trained using five different training corpora. Each neural network trained for the classifier input tests in Section 5.3.1 use their respective test sets. The baseline for each subversion dataset and neural network classifier is the test using the un-normalized dataset, the entry named "Default (baseline)." Each of the tables presented in the rest of the section is presented in the same manner. For each modified dataset, a total of five results are presented as follows:

- Default (baseline): Using the neural network trained on the unmodified dataset, this network classifies the un-normalized version of the subversive dataset.
- Default (Normalized): Using the neural network trained on the unmodified dataset, this network classifies the normalized version of the subversive dataset.
- Phonetic: Using the neural network trained on the phonetic version of the unmodified dataset, this network classifies the phoneticized version of the subversive dataset.
- Normalized: Using the neural network trained on the normalized version of the unmodified dataset, this network classifies the normalized version of the subversive dataset.
- Hybrid: Using the neural network trained on the hybrid inputs, this network classifies the hybrid form of the dataset made up of both the phoneticized and normalized for version of the subversive dataset.

Each of these setups thus represents a different scenario. The `Default (baseline)` version of the neural network represents a normal classifier working with no normalization, and is the baseline we will compare our systems to to demonstrate the impact of normalization. The `Default (normalized)` setup represents the use of our normalizer as input to a harm classifier that has not been retrained, and thus checks if our normalized text can be used as plain text by a system. The `Normalized`, `Phonetic` and `Hybrid` results represent our normalizer being used in different contexts and paired with harm classifiers trained on the same data to account for changes in input.

Duplication Dataset

Neural Network	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate
Default (baseline)	87.56	87.95	**99.42**	94.13	99.67	**94.52**	98.89
Default (normalized)	74.72	77.24	99.22	93.38	99.67	92.19	98.89
Phonetic	**88.30**	**88.88**	99.37	**94.84**	99.67	93.76	98.89
Normalized	85.84	86.68	99.35	93.70	99.67	93.30	98.89
Hybrid	83.70	85.03	99.31	94.36	99.67	94.06	98.89

Table 5.5: Accuracy for all labels for the `Bi-LSTM` neural networks on the Duplication Dataset

Neural Network	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate
Default (baseline)	90.60	90.84	99.43	94.55	99.67	94.83	98.89
Default (normalized)	**91.30**	**91.53**	99.43	95.10	99.67	95.09	98.89
Phonetic	90.54	90.67	99.43	**95.34**	99.67	**95.17**	98.89
Normalized	90.24	90.48	99.43	94.23	99.67	94.64	98.89
Hybrid	90.24	90.48	99.43	94.23	99.67	94.64	98.89

Table 5.6: Accuracy for all labels for the `RoBERTa` neural networks on the Duplication Dataset

Removal Dataset

Neural Network	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate
Default (baseline)	88.05	88.76	**99.42**	94.12	99.67	**94.55**	98.89
Default (normalized)	67.88	71.27	99.37	93.04	99.67	91.91	98.89
Phonetic	88.30	88.87	99.41	**94.77**	99.67	94.32	98.89
Normalized	**88.53**	**89.04**	99.39	94.20	99.67	94.15	98.89
Hybrid	87.89	88.69	99.40	94.55	99.67	94.54	98.89

Table 5.7: Accuracy for all labels for the `Bi-LSTM` neural networks on the Removal Dataset

Neural Network	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate
Default (baseline)	90.51	90.75	99.43	94.48	99.67	94.79	98.89
Default (normalized)	**90.83**	**91.06**	99.43	94.75	99.67	94.85	98.89
Phonetic	90.14	90.34	99.43	**95.11**	99.67	**94.99**	98.89
Normalized	90.24	90.48	99.43	94.23	99.67	94.64	98.89
Hybrid	90.24	90.48	99.43	94.23	99.67	94.64	98.89

Table 5.8: Accuracy for all labels for the `RoBERTa` neural networks on the Removal Dataset

Substitution Dataset

Neural Network	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate
Default (baseline)	90.02	**90.57**	99.09	**94.71**	99.62	**94.60**	98.70
Default (normalized)	86.04	86.95	98.97	93.67	99.62	93.33	98.70
Phonetic	89.78	90.19	**99.17**	94.56	99.62	94.14	98.70
Normalized	**90.03**	90.42	99.11	94.10	99.62	93.84	98.70
Hybrid	89.77	90.29	99.08	94.59	99.62	94.27	98.70

Table 5.9: Accuracy for all labels for the `Bi-LSTM` neural networks on the Substitution Dataset

Neural Network	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate
Default (baseline)	90.36	90.62	99.35	94.53	99.62	94.62	98.70
Default (normalized)	90.49	90.75	99.35	94.67	99.62	94.54	98.70
Phonetic	**91.31**	**91.41**	99.35	**95.22**	99.62	**94.93**	98.70
Normalized	89.06	89.33	99.35	93.49	99.62	94.00	98.70
Hybrid	89.06	89.33	99.35	93.49	99.62	94.00	98.70

Table 5.10: Accuracy for all labels for the `RoBERTa` neural networks on the Substitution Dataset

Combined Dataset

Neural Network	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate
Default (baseline)	86.37	86.86	**99.34**	93.39	99.62	**93.88**	98.70
Default (normalized)	70.98	73.43	99.23	92.59	99.62	91.58	98.70
Phonetic	**86.94**	**87.52**	99.32	**94.18**	99.62	93.32	98.70
Normalized	86.03	86.72	99.28	93.13	99.62	93.04	98.70
Hybrid	85.14	86.12	99.29	93.59	99.62	93.56	98.70

Table 5.11: Accuracy for all labels for the `Bi-LSTM` neural networks on the Combined Dataset

Neural Network	has_toxicity	toxic	severe_toxic	obscene	threat	insult	identity_hate
Default (baseline)	89.32	89.59	99.35	93.72	99.62	94.13	98.70
Default (normalized)	**89.88**	**90.14**	99.35	94.17	99.62	94.36	98.70
Phonetic	89.40	89.61	99.35	**94.49**	99.62	**94.49**	98.70
Normalized	89.06	89.33	99.35	93.49	99.62	94.00	98.70
Hybrid	89.06	89.33	99.35	93.49	99.62	94.00	98.70

Table 5.12: Accuracy for all labels for the `RoBERTa` neural networks on the Combined Dataset

Analysis

We begin this analysis by looking at the difference in the accuracy between the original, subversion-less, datasets found in Tables 5.3 and 5.4 and the accuracy of the various subversion setups presented in Tables 5.5 to 5.12.

The first result we can observe is the impact of subversion on harm detection. By comparing the results of Tables 5.3 and 5.4 to the `Default (baseline)` results, we can confirm that the addition of subversive elements in messages reduces the overall performance of harm classifiers. This is shown in the fact the detection accuracy of almost all the predictions, apart from the three under-represented labels, `severe_toxic`, `threat` and `identity_hate` goes down by a few percentage points. The behavior seen with the under-represented labels, a very minor change downwards or an unchanged accuracy, is explained by overfitting due to the relatively small proportion those labels represent in the training dataset. An additional odd behavior in the case of the `severe_toxic` label on both the duplication and removal datasets, when using the `Bi-LSTM` classifiers, in the form of a small increase in the accuracy can be observed.

This behavior can be explained by the simple fact that both of these transformations create or remove noise in the messages. This slight increase is reflected in the combined dataset, but to a lesser degree, which we can attribute to the presence of substitutions.

Next, we consider the effect of normalizing the input before harm classification, by comparing the `Default (baseline)` and `Default (normalized)` results. In this comparison, we notice a striking difference between the `Bi-LSTM` and `RoBERTa` classifiers. The latter benefits from our normalization, and even achieves its best results on the `has_toxicity` and `toxic` labels with that setup. On the other hand, the `Bi-LSTM` has the opposite behaviour and achieves some of its worst performances with that setup, including a staggering 21% drop in accuracy on the removal set for the `has_toxicity` label in Table 5.7. This indicates that our normalizer does have an effect on the input text, but for that effect to be positive or negative depends on the harm detection architecture being used afterwards. The transformed messages cannot be used blindly as input for any system.

This observation is confirmed by our next experiment, the comparison using the normalized input and the harm detection classifier re-trained on normalized data. In this case, we compare the `Default (baseline)` and `Default (normalized)` results to the `Default (baseline)` and `Normalized` results. Once again, the result depends on the harm classifier being used. In this case, the `Bi-LSTM` improves and achieves some of its best performances. Notably, the accuracy on the removal dataset for the `has_toxicity` label in Table 5.7 goes from the worst performance when using normalized inputs and the default classifier to the best performance when using normalized inputs and the retrained classifier. The `RoBERTa` classifier again has the opposite behavior, and while its performance using the retrained classifier does not drop precipitously, it remains consistently lower than or equal to that of the default classifier with either inputs.

Finally, we can compare the performance of the retrained network using the three different forms of our normalized input, the `Normalized`, `Phonetic`, and `Hybrid` inputs. There are improvements to be seen in the results, however there is no clear benefit to one method over the other two. Instead, the benefits seem to vary from situation to situation, with some harm labels performing better with one normalization setup. For example, we can generally observe a positive relationship between the use of phonetic inputs and the detection of the `has_toxicity` label, while that input hinders the detection of `insult` when using the `Bi-LSTM` classifier. Likewise, we can observe relationships between certain types of subversion and normalization. For example, the hybrid input performs worse in the presence of removal and duplication type of subversion, which may be due to the added or missing letters messing up the alignment between the text and phonetic content of the concatenated input.

Putting all our results together, we can draw some conclusions on the use of our normalizer for subversion detection. The main conclusion, when looking at overall toxicity detection (the

has_toxicity label), is that one form or another of our normalizer always performs better than not using the normalizer. Using the RoBERTa classifiers, the Default (normalized) and the Phonetic setup both outperform the baseline when subversive elements are present. For the Bi-LSTM classifier, the best performing setup is usually the Phonetic classifier, with the only real exception being on the Substitution dataset (Table 5.9), where the Normalized setup outperforms it.

As an aside, in Tables 5.9, 5.10, 5.11 and 5.12, a slight drop in the accuracy of the labels threat and identity_hate can be seen when compared to all other results obtained on those labels in this chapter. This is caused by the way the Substitution and Combined datasets are created. As explained in Section 4.1.4, we have multiple copies of the different messages in the dataset with different substitutions. The amount of copies for each message varies based on how many pairs are present in a substitution group. As such, the drop in accuracy is caused by the fact that, for those labels, the messages with more duplication pairs are those on which the classifier ends up predicting the wrong class due to overfitting on those labels. An identical behaviour can be seen for the severe_toxic label with the RoBERTa classifiers.

While the results show that using our phonetic normalization has some benefits in our use-case, the improvement is neither as high nor as uniform as we might have hoped. It is interesting to ponder why that is the case.

As mentioned in the beginning of our research, our underlying assumption is the spelling of words is being transformed in a way that does not affect their pronunciation, such as the substitution “2day”, the repetition “yyyeeeessssss” or the removal “wat”. However, the algorithm we used to augment the dataset, described in Section 2.3.2, did not have this constraint. It could transform words into different-sounding words, for example by removing the letter “r” from the word “army” turning it into “amy”, or duplicating the “e” in “bet” turning it into “beet”. When harmful words are transformed in this way for our subversion experiments, it eliminates, not subverts nor masks, its harmful content. Unfortunately, it seems a much more precise algorithm to generate subversive content would be needed to improve our experimental procedure.

5.4 Conclusion

Considering the experimental results presented in this chapter, we can conclude that phonetic normalization does offer improvements for harm detection. Using a sophisticated enough harm detector, such as the RoBERTa model, it can even be applied as a preprocessing step without modifying or retraining the subsequent detection system. Moreover, given a real-world setting such as the one sampled in the Kaggle Jigsaw dataset, detecting harmful content in the phonetic form of messages gives better results than using the plain text messages. Finally, given an increased level of subversive harm in the messages, we also find that our normalizer

helps in detecing the disguised harmful content.

Conclusion

Throughout this thesis we presented our implementations along with the results obtained through our various experimental phases. In order to properly conclude our thesis, it is necessary to reflect on our work within the context of the goals we originally set out to achieve.

The goal of our research, and of this thesis, was to design and implement a novel text normalization technique that could infer the phonetic form of OOV words and make use of that information to discover the corresponding IV English words, and to use that system to improve harm detection in the specific context of subversive users. This would allow harmful content detection systems to perform their duties as they were originally designed to in an ever-changing online environment. In Chapters 2 and 3 we presented our phonetic normalizer, a two-part system made up of two sequence-to-sequence transformers capable of producing phonetics in the form of words made up of characters from the International Phonetic Alphabet (IPAs) from any word, in-vocabulary or out-of-vocabulary, and to convert those IPAs back into words that are contained in the English language. In Chapters 4 and 5, we used several variations of this normalizer and paired them with two different harm detectors in order to measure their effects in filtering out different forms of subversive harm.

The phonetic normalizer we produced is capable of improving the classification results when compared to a baseline harm classifier, both using normal online messages and messages we corrupted with heavy subversion. Even though the results showed only a modest improvement and were not uniform in all situations, we did show that our approach works, and that there is a fundamental benefit to using phonetic information to process online text.

Appendix A

Ratio of Word and IPA Length in Datasets

A.1 Word Length Ratio for Text-to-IPA Experiments

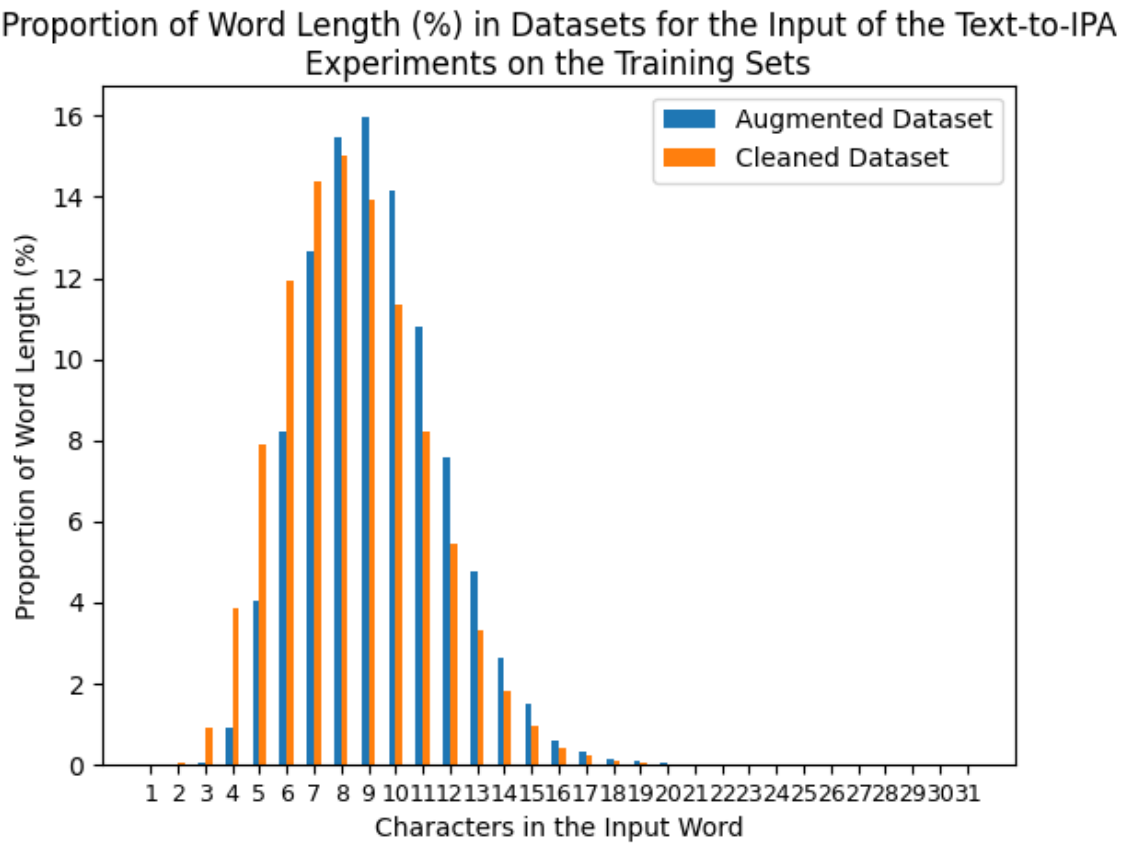

Figure A.1: Word length ratio for the training sets used in the Text-to-IPA experiments

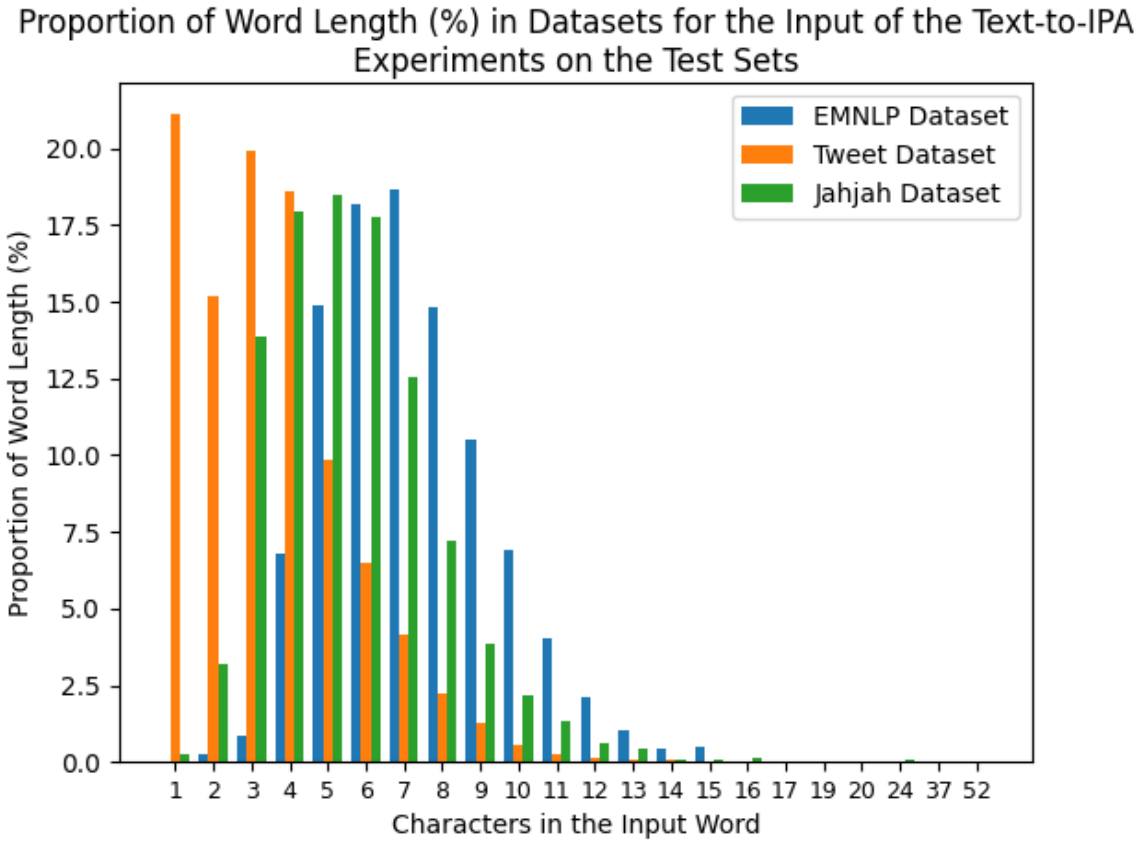

Figure A.2: IPA length ratio for the test sets used in the Text-to-IPA experiments

A.2 Word Length Ratio for IPA-to-Text Experiments

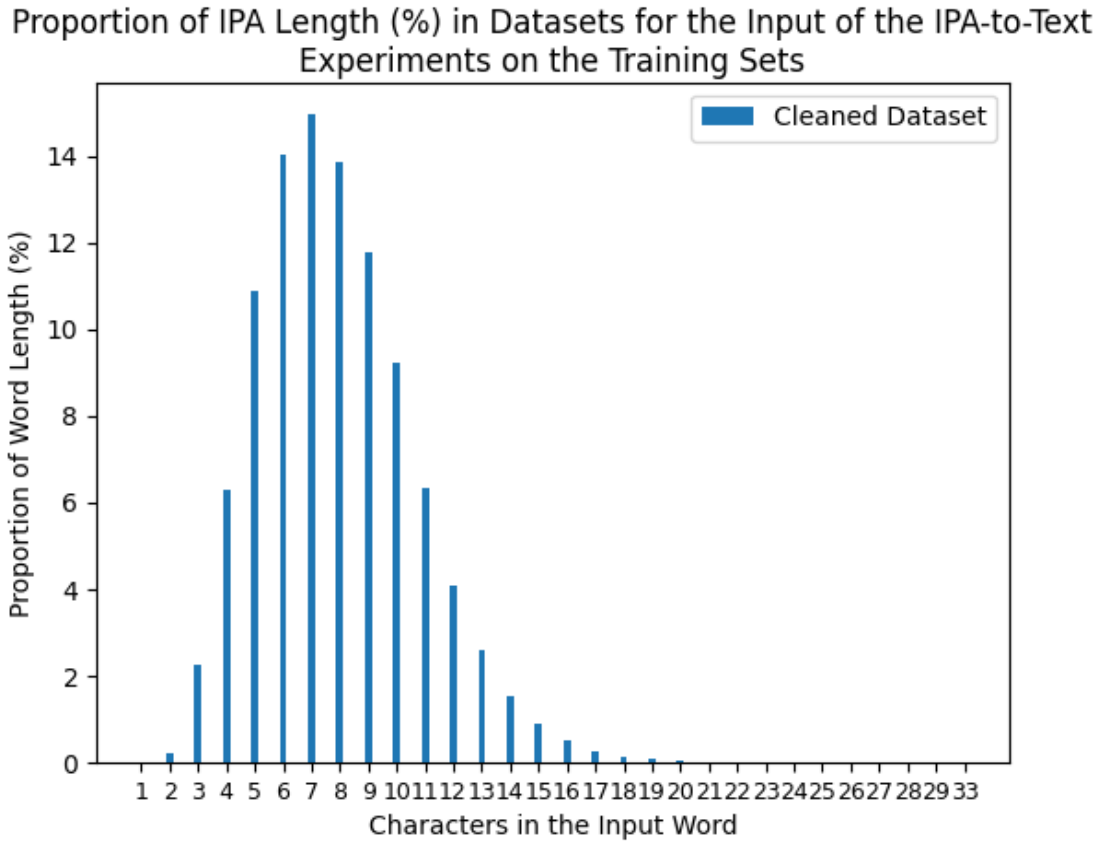

Figure A.3: IPA length ratio for the training sets used in the IPA-to-Text experiments

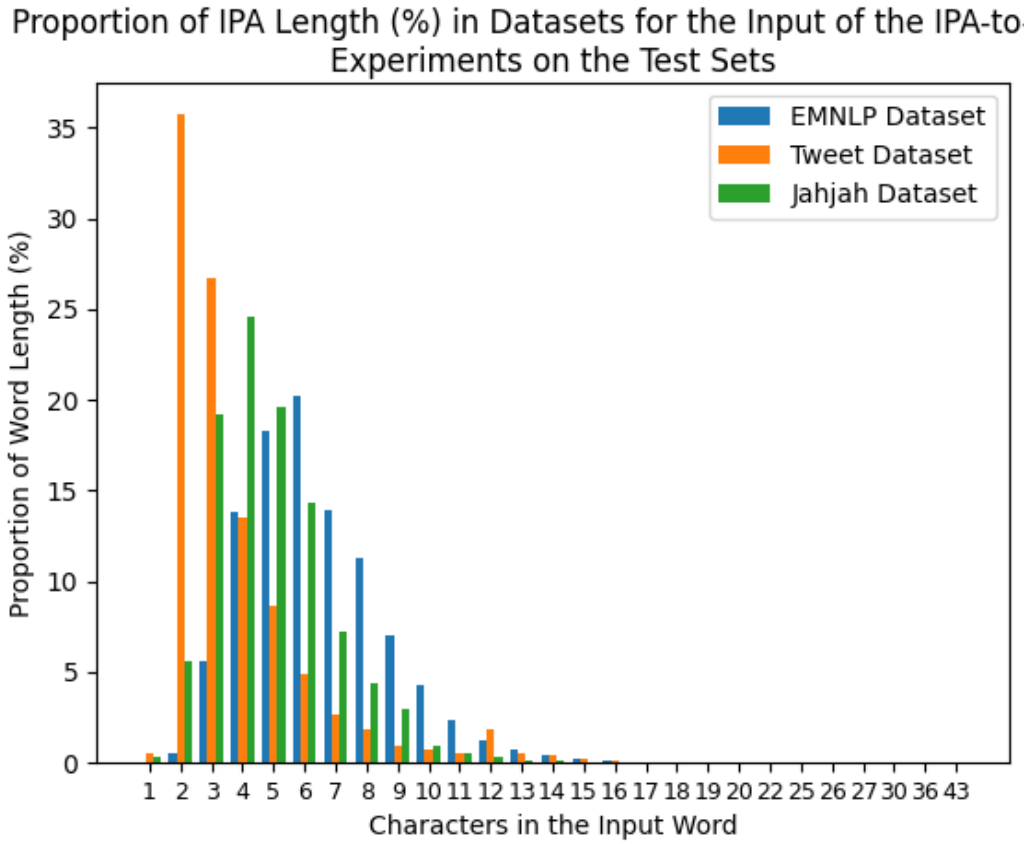

Figure A.4: Generated IPA length ratio for the test sets used in the IPA-to-Text experiments

Appendix B

Notable Architectural Data

B.1 Notable Architectural Data for the `fairseq` networks

	fconv_wmt_en_de
Encoder	Convolutional
Encoder Layers	20
Encoder Hidden Dimension	768
Decoder	Convolutional
Decoder Layers	20
Decoder Hidden Dimension	768
Hidden Layers	15
Hidden Dimensions	512 (9 Layers), 1024 (4 layers), 2048 (2 layers)
Attention	Each Decoder layer

Table B.1: Notable Architectural Data of the `fairseq` Convolutional network

	lightconv_wmt_en_de	lightconv_wmt_en_fr_big	lightconv_wmt_en_de_big
Activation Function	ReLU	ReLU	ReLU
Encoder	Convolutional	Convolutional	Convolutional
Encoder Layers	7	7	7
Encoder Hidden Dimension	512	1024	1024
Encoder FFN Dimension	2048	4096	4096
Encoder Attention Heads	8	16	16
Decoder	Convolutional	Convolutional	Convolutional
Decoder Layers	6	6	6
Decoder Hidden Dimension	512	1024	1024
Decoder FFN Dimension	2048	4096	4096
Decoder Attention Heads	8	16	16

Table B.2: Notable Architectural Data of the `fairseq` LightConv networks

	transformer_wmt_en_de	transformer_vaswani_wmt_en_de_big	transformer_wmt_en_de_big_t2t
Activation Function	ReLU	ReLU	ReLU
Encoder Layers	6	6	6
Encoder Hidden Dimension	512	1024	1024
Encoder FFN Dimension	2048	4096	4096
Encoder Attention Heads	8	16	16
Decoder Layers	6	6	6
Decoder Hidden Dimension	512	1024	1024
Decoder FFN Dimension	2048	4096	4096
Decoder Attention Heads	8	16	16

Table B.3: Notable Architectural Data of the `fairseq` Transformers

B.2 Notable Architectural Data for the `RoBERTa` network

	RoBERTa
Activation Function	GELU
Hidden Dimension	768
Hidden Layers	12
Intermediate Size	3072
Attention Heads	12

Table B.4: Notable Architectural Data of the `RoBERTa` network

Bibliography

[1] Caverphone. URL https://xlinux.nist.gov/dads/HTML/caverphone.html.

[2] Lexical Normalisation for English Tweets. URL https://noisy-text.github.io/norm-shared-task.html#resource.

[3] Toxic Comment Classification Challenge. URL https://kaggle.com/c/jigsaw-toxic-comment-classification-challenge.

[4] metaphone. URL https://xlinux.nist.gov/dads/HTML/metaphone.html.

[5] Accessing individual records from personal data files using non-unique identifiers / Gwendolyn B. Moore ... [et al.] ; prepared for the Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, D.C : Moore, Gwendolyn B : Free Download, Borrow, and Streaming : Internet Archive. URL https://archive.org/details/accessingindivid00moor.

[6] Daitch-Mokotoff Soundex System, . URL https://www.avotaynu.com/soundex.htm.

[7] Free to Play? Hate, Harassment and Positive Social Experience in Online Games 2020, . URL https://www.adl.org/free-to-play-2020.

[8] NYSIIS. URL https://xlinux.nist.gov/dads/HTML/nysiis.html.

[9] soundex. URL https://xlinux.nist.gov/dads/HTML/soundex.html.

[10] Martin J. Ball. Further to Articulatory Force and the IPA Revisions. *Journal of the International Phonetic Association*, 23(1):39–41, June 1993. ISSN 1475-3502, 0025-1003. doi: 10.1017/S0025100300004783. URL https://www.cambridge.org/core/journals/journal-of-the-international-phonetic-association/article/abs/further-to-articulatory-force-and-the-ipa-revisions/3EAFD5C249A65974F1BB94CF4E599074.

[11] Martin J. Ball, John Esling, and Craig Dickson. The VoQS System for the Transcription of Voice Quality. *Journal of the International Phonetic Association*, 25(2):71–80, December 1995. ISSN 1475-3502, 0025-1003. doi: 10.1017/S0025100300005181. URL https://www.cambridge.

org/core/journals/journal-of-the-international-phonetic-association/article/abs/voqs-system-for-the-transcription-of-voice-quality/47F4EBC4BD5E393F28BCF1F8FD7D153C.

[12] Bird, Steven, Edward Loper, and Ewan Klein. *Natural Language Processing with Python*. O'Reilly Media Inc, 2009.

[13] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *Transactions of the Association for Computational Linguistics*, 5:135–146, 2017. ISSN 2307-387X.

[14] Eloi Brassard-Gourdeau and Richard Khoury. Subversive Toxicity Detection using Sentiment Information. In *Proceedings of the Third Workshop on Abusive Language Online*, pages 1–10, Florence, Italy, August 2019. Association for Computational Linguistics. doi: 10.18653/v1/W19-3501. URL https://www.aclweb.org/anthology/W19-3501.

[15] François Chollet et al. Keras. https://keras.io, 2015.

[16] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv:1810.04805 [cs]*, May 2019. URL http://arxiv.org/abs/1810.04805. arXiv: 1810.04805.

[17] Jonas Gehring, Michael Auli, David Grangier, Denis Yarats, and Yann N. Dauphin. Convolutional Sequence to Sequence Learning. *arXiv:1705.03122 [cs]*, July 2017. URL http://arxiv.org/abs/1705.03122. arXiv: 1705.03122.

[18] Vincent Jahjah, Richard Khoury, and Luc Lamontagne. Word normalization using phonetic signatures. In *Canadian Conference on Artificial Intelligence*, pages 180–185. Springer, 2016.

[19] Kathy. Framework. URL https://fairplayalliance.org/framework/.

[20] Aldebaro Klautau. Arpabet and the timit alphabet. *URL: https://web. archive. org/web/20160603180727/http://www. laps. ufpa. br/aldebaro/papers/ak_arpabet01. pdf*, 2001.

[21] George Adams Kopp, Harriet C Green Kopp, and Angelo Angelocci. *Visible speech manual*. Wayne State University Press, 1967.

[22] Marc-André Larochelle, Éloi Brassard-Gourdeau, Zeineb Trabelsi, Richard Khoury, Sehl Mellouli, Liza Wood, and Chris Priebe. Protecting online communities from harmful behaviors. *MÉDIAS SOCIAUX*, page 149.

[23] 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:1907.11692 [cs]*, July 2019. URL `http://arxiv.org/abs/1907.11692`. arXiv: 1907.11692.

[24] Kenneth L Pike. Phonetics: A critical analysis of phonetic theory and a technic for the practical description of sounds. 1972.

[25] Hans Joachim Postel. Die kölner phonetik. ein verfahren zur identifizierung von personennamen auf der grundlage der gestaltanalyse. *IBM-Nachrichten*, 19:925–931, 1969.

[26] Tobias Renwick and Denilson Barbosa. Detection and Identification of Obfuscated Obscene Language with Character Level Transformers. *Proceedings of the Canadian Conference on Artificial Intelligence*, June 2021. URL `https://caiac.pubpub.org/pub/5uqi2h7k/release/1`.

[27] C Russell Robert. The soundex coding system. *Patent No. US1261167*, 1918.

[28] José Carlos Rosales Núñez, Djamé Seddah, and Guillaume Wisniewski. Phonetic Normalization for Machine Translation of User Generated Content. In *Proceedings of the 5th Workshop on Noisy User-generated Text (W-NUT 2019)*, pages 407–416, Hong Kong, China, November 2019. Association for Computational Linguistics. doi: 10.18653/v1/D19-5553. URL `https://www.aclweb.org/anthology/D19-5553`.

[29] Ranjan Satapathy, Claudia Guerreiro, Iti Chaturvedi, and Erik Cambria. Phonetic-based microtext normalization for twitter sentiment analysis. In *2017 IEEE International Conference on Data Mining Workshops (ICDMW)*, pages 407–413. IEEE, 2017.

[30] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention Is All You Need. *arXiv:1706.03762 [cs]*, December 2017. URL `http://arxiv.org/abs/1706.03762`. arXiv: 1706.03762.

[31] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online, October 2020. Association for Computational Linguistics. URL `https://www.aclweb.org/anthology/2020.emnlp-demos.6`.

[32] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. PAY LESS ATTENTION WITH LIGHTWEIGHT AND DYNAMIC CONVOLUTIONS. page 14, 2019.