



HAL
open science

Data augmentation and generation for natural language processing

Hugo Boulanger

► **To cite this version:**

Hugo Boulanger. Data augmentation and generation for natural language processing. Computation and Language [cs.CL]. Université Paris-Saclay, 2023. English. NNT : 2023UPASG019 . tel-04105431

HAL Id: tel-04105431

<https://theses.hal.science/tel-04105431>

Submitted on 24 May 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Data Augmentation and Generation
for Natural Language Processing.
*Augmentation et génération de données d'apprentissage
en traitement automatique des langues.*

Thèse de doctorat de l'université Paris-Saclay

École doctorale n°580 sciences et technologies de l'information
et de la communication (STIC)
Spécialité de doctorat: Informatique
Graduate School : Informatique et sciences du numérique,
Réfèrent : Faculté des sciences d'Orsay

Thèse préparée dans le Laboratoire Interdisciplinaire des Sciences du
Numérique (LISN, Université Paris-Saclay, CNRS),
sous la direction de Sophie ROSSET, directrice de recherche,
et le co-encadrement de Thomas LAVERGNE, Maître de conférences

Thèse soutenue à Paris Saclay, le 30 mars 2023, par

Hugo BOULANGER

Composition du jury

Membres du jury avec voix délibérative

Fatiha Saïs

Professeure, Université Paris Saclay

Christophe Gravier

Professeur, Université Jean Monnet

Vincent Claveau

Chargé de recherche, CNRS

Benoît Sagot

Directeur de recherche, Inria

Laure Soulier

Maîtresse de conférences, Sorbonne Université

Présidente

Rapporteur & Examineur

Rapporteur & Examineur

Examineur

Examinatrice

Acknowledgements

Working on a PhD is an intensive task, both on the energy required to produce the research and on the mental fortitude needed to not lose purpose when faced with negative results or the unexpected and abrupt disruption of life which was Covid. First, I would like to thank my supervisors Sophie Rosset and Thomas Lavergne for the trust you have given me to work on this broad and complex topic. Sophie, thank you for your patience and your broad vision of the domain. Thomas, thank you for your technical insights, and your teaching expertise.

I am grateful for the support of my family through the long educational cursus I have followed. I would like to thank my friends for the continuous support and the connection that we have kept which helped us through these difficult times.

I want to thank my colleagues and fellow PhD students for their feedback and propositions. First, I would like to thank Valentin Carpentier for the three years spent as housemate and colleague. I would like to thank Mathilde Veron for her advises, energy, and all the lunches shared. I also want to thank Juan-Manuel Coria for his insightful comments. I would like to thank all of the fellow researchers that I met at conferences and winter schools for the broader view they have given me on NLP. I hope to see you at future events.

I am grateful for the constructive feedback on this work given by Vincent Claveau and Christophe Gravier. This work has been supported by the project PSPC AIDA: 2019-PSPC-09 funded by BPI-France. This work was granted access to the HPC resources of IDRIS under the allocation 2021-AD011013018 made by GENCI. This work was granted access to the HPC resources of Saclay-IA through the Lab-IA machine.

Contents

Contents	5
Terminology	9
1 Introduction	11
1.1 Motivation	11
1.2 Research objectives	13
1.3 Outline and contributions	13
2 Text Generation	15
2.1 Introduction	15
2.2 Data representation	16
2.2.1 Tokenization	16
2.2.2 Vectorization	19
2.3 Language Modeling	22
2.3.1 Introduction	23
2.3.2 Decoding algorithms	23
2.3.3 Model architectures	25
2.3.4 Pre-trained Large Language Models	29
2.3.5 Data	33
2.3.6 Language modeling in this work	34
2.4 Vector-to-Text generation	34
2.4.1 Autoencoders	34
2.4.2 Generative Adversarial Networks	36
2.4.3 Conclusion	36
2.5 Augmentation	37
2.6 Conclusion	37
3 Methodology	39
3.1 Setting	39
3.1.1 Resource topology	39
3.1.2 Target setting	41
3.2 Evaluation methods	41
3.2.1 Evaluating generation	41
3.2.2 Sequence labeling	42
3.2.3 Evaluating sequence labeling	42
3.3 Protocol	44
3.3.1 Low-resource simulation	44
3.3.2 Evaluation	45

3.3.3	Discussion	46
3.4	Corpora	46
3.4.1	Named Entity Recognition	46
3.4.2	Natural Language Understanding	48
3.5	Taggers	50
3.6	Summary of the methodology	51
4	Pattern Filling Generation	53
4.1	Pattern filling generation	53
4.2	Generated data	55
4.2.1	Description	55
4.2.2	Results	56
4.2.3	Data Analysis	57
4.2.4	Following natural distributions	61
4.2.5	Conclusion	62
4.3	On the number of patterns	63
4.3.1	Experiment Description	63
4.3.2	Results	63
4.3.3	Conclusion	64
4.4	Augmentation	65
4.4.1	Results	65
4.4.2	Conclusion	66
4.5	Final experiments	66
4.5.1	Settings	67
4.5.2	Results	67
4.6	Conclusion	69
5	Language modeling and Semi-supervision	71
5.1	Semi-supervised learning	71
5.2	Tri-training with a static set of generated data	72
5.2.1	Tri-training	72
5.2.2	Generation methods	75
5.2.3	Tri-training experimental settings	75
5.2.4	Results	75
5.2.5	Conclusion	77
5.3	Generative Tri-training	78
5.3.1	Generation methods	79
5.3.2	Semi-supervised learning methods	81
5.3.3	Experimental settings	82
5.3.4	Results	84
5.3.5	Conclusion	86
5.4	Impact of a small development set and performances of generated data	87
5.4.1	Small development set	87

5.4.2	Training on the data generated	89
5.4.3	Comparison to pattern-filling generation	92
5.5	Conclusion	93
6	Modeling with patterns	95
6.1	Span-replacement with T5	95
6.1.1	Preliminary experiment	96
6.1.2	Data Analysis	98
6.1.3	Filtering and prompting	99
6.2	Combining pattern-filling and tri-training	101
6.2.1	Concatenation of Pattern-filling and Tri-training data	102
6.2.2	Applying pattern-filling on semi-supervised data	104
6.3	Conclusion	106
7	Conclusions	107
7.1	Contributions	107
7.2	Discussions	108
7.3	Summary of Findings	109
7.4	Perspectives	110
	Bibliography	113
	A ChatGPT conversation	123
	B Pattern Generation Appendix	125
	C BERT tagger architecture results	127

Terminology

F_1	Measure used to evaluate tagger performance, further elaborations in subsection 3.2.3.
tp	True positives, mentions where the prediction matches the known labels.
fp	False positives, mentions predicted by the tagger that do not match known labels.
fn	False negatives, mentions that were not predicted by the tagger.
n	Number of sentences sampled from the training corpus.
S_n	Natural subset of data sampled from corpora.
G_n	Generated subset of data generated from S_n .
t	Episode number of tri-training.
i, j, k	Index of tri-training taggers.
L_i^t	Pseudo-labeled set of data at episode t for model i .
U^t	Unlabeled data of tri-training at episode t .
m_i	Tagger i , used to describe the tagger in a broader setting.
m_i^t	Tagger i at episode t , used for comparisons between tri-training episodes.
M_{slot}	Number of mentions in the list associated with the slot.
Size	Size of S_n used for generation or augmentation.
Baseline	Experiment where the tagger is trained on S_n .
Topline	Tri-training experiment where the tagger is trained on S_n and the unlabeled set is natural data.
PatGen 20k	Experiment where the generated data is generated with pattern-filling generation with uniform distribution.
PatGen 20k m	PatGen 20k where mention (m) natural distributions are used.
PatGen 20k p	PatGen 20k where pattern (p) natural distributions are used.
PatGen 20k (p+m)	PatGen 20k where pattern (p) and mention (m) natural distributions are used.
PatGen 20k aug.	PatGen 20k used in an augmentation setting, can be combined with the m and p denominations.

Follow-up	F_1 scores of tri-training ensembles where the generation method is the generation of follow-up sentence with causal language modeling.
Completion	F_1 scores of tri-training ensembles where the generation method is sentence-completion with causal language modeling.
Mention rep.	F_1 scores of tri-training ensembles where the generation method is mention replacement with span replacement.
Context rep.	F_1 scores of tri-training ensembles where the generation method is context replacement with span replacement.
Combined	F_1 scores of tri-training ensembles where the generation method is sampled from the four previous generation methods
$\Delta Method$	Average of seed by seed deltas between the <i>Method</i> and the Baseline.
<i>Method</i> Individual	F_1 score of individual taggers trained during tri-training experiment <i>Method</i> .
<i>Method</i> Outputs	F_1 score of taggers trained on the data produced by experiment <i>Method</i> .
T5 20k mention	F_1 score of taggers trained on data augmented with mention replacement.
T5 20k context	F_1 score of taggers trained on data augmented with context replacement.
T5 20k both	F_1 score of taggers trained on data augmented with either context or mention replacement.
T5 20k filtered	T5 20k both with regex filtering.
T5 20k filtered prompt.	T5 20k filtered with prompts added in front of the modified sentence.
Follow-up + PatGen	F_1 score of taggers trained on data generated in Follow-up and PatGen 20k and S_n .
Mention rep. + PatGen	F_1 score of taggers trained on data generated in Mention rep. and PatGen 20k and S_n .
PatGen on Follow-up	F_1 score of taggers trained on Follow-up data outputs augmented by pattern-filling generation without natural distributions.

1 - Introduction

Natural Language Processing (NLP) is a rapidly growing field that deals with the interaction between computers and human language. One of the major challenges in NLP is dealing with low-resource scenarios, where limited amounts of labeled data are available. In this thesis, we focus on the use of text generation as a means to produce additional data for improving sequence labeling tasks in low-resource settings. Sequence labeling is a fundamental task in NLP that involves assigning labels to individual elements in a sequence, such as words in a sentence or tokens in a text. The goal of this research is to explore the potential of text generation to enhance the performance of sequence labeling models by augmenting the available low-resource training data. We investigate various text generation techniques and evaluate their effectiveness on different sequence labeling tasks in low-resource settings. The contributions of this thesis include a novel approach for using text generation to improve sequence labeling in low-resource settings and an in-depth analysis of the impact of text generation on the performance of sequence labeling models in such scenarios.

This first paragraph of the introduction contains good surface knowledge of the domain and presents the problems faced in this thesis, but it fails to deliver on the contributions made by this thesis. It is also very short for a thesis introduction; at this point, you might be wondering why we are discussing the qualities of the first paragraph. It is because a human did not write the previous paragraph; it was generated using ChatGPT¹. Thanks to the advances in NLP, generating compelling text has become as simple as writing some description of the logic and elements of the wanted text, and this kind of generation will be one of the main topic of this thesis.

1.1 . Motivation

Using NLP for information extraction requires labeled data to build statistical models that perform annotation. Usual corpora for sequence labeling typically contain around fifteen to twenty thousand sentences that have to be cleansed of private information, manually labeled by multiple people, and for some, have to be transcribed from speech to text, which adds yet another layer of work. Problems can arise at every step of the process, which makes it a lengthy and costly process that can easily hinder innovation and applications of NLP methods to new domains or domains where the target data contains private information. For example, suppose you want to create an application that can extract relevant information about treatments, tests, or problems from patient records. In that case, you will need to have access to patient records that have to be labeled by medical professionals who have very low time to spare for labeling. Fake medical records could be written by medical professionals to avoid privacy issues, as can happen for cases studied in a university or used for exams. This would, however, take

¹<https://chat.openai.com>

even more time from the medical professionals, which goes against the goal of automating cumbersome tasks through A.I. and automation. If writing sentences and labeling them could be automated to create datasets, that would solve the problems highlighted before. This sounds like the serpent eating its own tail, as it is our aim to be able to automate labeling with machine learning approaches, which require labeled data. This limitation of needing labeled data is a limitation of machine learning for which solutions exist, such as using non-labeled data as additional training examples. Text generation, on the other hand, has improved to the point of being able to generate compelling text thanks to the ever-larger models and datasets used for their training.

The following quote is an example of text relevant to the medical task generated using ChatGPT in the same manner we have generated the introduction, with mentions manually labeled.

“John Smith, a 54-year-old patient, visited his primary care physician complaining of **chest pain** and **shortness of breath**. The physician ordered an **ECG test**, which revealed that the patient was experiencing an **acute coronary syndrome**. The doctor prescribed **aspirin** as a treatment and referred the patient to a cardiologist for further evaluation. The cardiologist recommended a **cardiac catheterization procedure** to assess the extent of the damage to the coronary artery. Based on the results, the patient was diagnosed with **atherosclerosis**, and the cardiologist recommended a treatment of **angioplasty** and **stenting**. The patient underwent the procedure and was discharged from the hospital the following day. The patient was advised to follow a healthy lifestyle and to take medications to lower his cholesterol levels to prevent future **cardiac events**.” – ChatGPT.

In the previous quote, the cyan-colored text represents the problems, the green text represents tests, and the red text represents treatments. This text was labeled manually as per our understanding of the concepts present, to highlight the mentions that could exist as ChatGPT doesn't label the data generated. While ChatGPT did not exist during our thesis, the underlying principles of language modeling and the architecture of GPT models [Radford et al., 2018] existed and already produced compelling text. Recent models are trained on large amounts of data, tens to hundreds of gigabytes of text, which contain information about a wide amount of subjects. These subjects can directly be linked to the domain of the downstream tasks we wish to solve or adjacent enough to bring relevant information in the generation setting. These models are, however, not trained to produce labeled sentences and would require thousands of sentences to be fine-tuned without problems for this task.

Our aim is to reduce the barrier of entry of NLP by reducing the quantity of natural data necessary to train information extraction tools. In particular, we will work on sequence labeling tasks such as named entity recognition or natural language understanding. To reduce the need for data, we propose to study text generation in order to generate the missing data.

1.2 . Research objectives

As presented previously, machine learning requires a high quantity of data to produce well-performing models. If the target task is to predict labels on the data, the training data must also be labeled, which makes the data more difficult and costly to produce. The goal of this thesis is to study and devise methods to improve sequence labeling via the use of generated data. The methods need to be usable in the context of the AIDA project, the project on which this thesis is financed. The goal of this project is to build A.I. tools to help the French private sector have access to A.I. This project needs to easily be applied to new tasks where data is only available in extremely small quantities due to privacy or industrial constraints. Our goal in this project is to be able to produce labeled data that can be used to improve the performance of information extraction models. Our methods must be applied in a setting where the quantity of labeled data is extremely low due to privacy considerations.

1.3 . Outline and contributions

To be able to understand this work, a review of the literature on text generation, with a focus on language modeling, is given in the following chapter. This chapter is oriented toward language modeling, which is also a technique used for the downstream task tackled in this thesis. The methodology is then described, which sets experimental settings shared between the chapters. This includes setting up the low-resource setting, where the quantity and nature of the data and resources used are discussed. We propose a method of evaluation for task-driven data generation and a generic experimental protocol describing how the natural data is used as a starting point for generation. The corpora and taggers used in this thesis are also presented in this chapter. The contributions of this thesis are described in the following paragraphs in the order they are presented.

Pattern-filling generation: The first generation method presented in this thesis is the pattern-filling generation method. We have proposed our implementation² of the method. We have also provided extensive analysis of the performance of the method in multiple settings [Boulanger, 2020] that has been presented at RÉCITAL 2020. The first setting tested is a setting where the entirety of the data used to train the taggers is generated. The second setting aims at finding the required number of patterns needed to produce generated data of a quantity. The third setting is a setting where the generated data is used alongside the natural data to train the taggers. We also propose the study and evaluation of the methods used with recent models and methodology to tackle the limits of previous evaluations.

Tri-training and language modeling: The second approach for labeled text generation presented in this thesis is the use of language modeling coupled with

²<https://github.com/HugoBoulanger/Pattern-Filling-Generation>

tri-training, a semi-supervised learning method. We have first contributed a study presented at Deeplo 2022 of the behavior of the tri-training algorithm with data generated with language models³ [Boulanger et al., 2022]. Our second contribution to this approach is generative tri-training, a modified tri-training algorithm including the language modeling text generation process, which is currently in the publication process. With this method, we further formalize the language modeling methods used for text generation in the first contribution and add new methods to the pool of available methods. Generative tri-training is both a labeled text generation technique and a training algorithm, and the previous contributions evaluated only the taggers trained during the tri-training procedure. Our final contribution to this method is the test of the quality of the labeled data generated by the generative tri-training process.

Combining language modeling and patterns: The language modeling techniques used to generate text presented in the previous chapters are combined in this chapter using different methods. To improve upon the lack of additional mention or patterns in the pattern-filling generation method, we proposed span-replacement modeling as a means to produce new labeled sentences with a process similar to pattern-filling generation, which did not produce positive results. This method is inspired by language modeling methods used with generative tri-training. As each generation method introduced in previous chapters improved upon the baseline, we also propose using them in parallel and in sequence.

³<https://github.com/HugoBoulanger/Tritraining-Gen>

2 - Text Generation

Training models on small quantities of data is a difficult task. Gathering more data for these situations might be a solution, but it is not always possible. While the availability of some textual data has improved over the years, many domains and languages remain that do not or cannot benefit from such openness. In those cases, using available data to augment the small quantity of in-domain labeled text might not be the right solution. This thesis aims to provide insights into tackling the low resource problem through generation. This chapter presents an overview of NLP techniques used to generate text. Some of these techniques could be adapted to generate labeled data.

2.1 . Introduction

Language is a medium of expression, a means for transmitting information and emotions. The canals through which language is expressed have evolved from spoken language to writing, recordings, or signing... They include various modes of expression across multiple temporalities. In this work, the focus is brought on the textual representation of language. Textual representation of language has been a tool for transmitting information and recording data. Due to the vast amount of data recorded in textual form throughout history, being capable of automatically retrieving information from text became a logical goal. The applications of information retrieval from texts are endless. Being capable of extracting information from a discussion is key to obtaining dialogue systems that can effectively do their tasks. Extracting information from documents can lead to a better and faster access to key documentation. Learning how to automatically create new texts following the rules of language learned intuitively became a goal.

The field of linguistics has studied language construction to understand better what mechanisms were at play to form language. Formalizing the properties of languages has led to theories on the construction of sentences. As such, one of the results of their studies is the production of methods to construct sentences. "A generative grammar is formally a collection of statements, rules, or axioms which describe, define or generate all well-formed utterances in a language and only those." [Halle, 1962]

While most linguistic knowledge is turned towards forming correct sentences, recent applications in NLP aim to solve problems around text which might not be well-formed and might include lists or tables. The text meant to be processed by NLP tools is often produced during human interactions. Whether it is a chat interaction, a tweet, or the automatic transcription of a piece of speech, these texts often contain errors at multiple levels. Chat interactions are prone to typing mistakes or wrong auto-corrections, which will change the meaning and structural information. Tweets are by nature limited in length, which causes users to produce text with lexical and structural errors to fit the format of tweets. Users use touches of sarcasm, subtext, and expressions linked to the site to express them-

selves in tweets. Tweets will contain URLs and hashtags, which might disrupt the structure of the tweets. These are the types of problems that are faced with the data processed by our tools. Our hypothesis is that the modeling techniques are better suited than linguistic techniques for this sort of data.

This chapter presents an overview of natural language processing oriented toward text generation. This chapter focuses on the techniques adjacent to language modeling. An overview of how text is represented as it contributes to the techniques used to analyze and generate text is given. Language modeling is presented, from how to decode a sequence of text to the ever-larger models and their applications. Text generation conditioned on non-textual data is presented. Finally, methods aimed at adding data generated from the available data are presented.

2.2 . Data representation

As NLP included more and more statistical tools, the need for data representation adapted to the tools used arose. Text representation impacts how text is processed, which, in turn, impacts how text is generated. This section presents how textual data can be transformed into a format fit for statistical models. Textual data is a sequence of bytes, or a sequence of characters if you know the encoding, and must first be split into relevant segments before processing. The first idea would be to split sentences into words. That would require to have shared definition of what a word is. This definition may vary depending on the mode of expression of language, the linguistic framework of analysis, and the language itself. For text, a first approach might be to use separators like spaces of any types or punctuations as markers of word frontiers. This approach might work for European languages, but there exist languages such as Japanese in which there are no separators for words. Keeping only words might also limit the capacity of analysis. Thus, any segments coming from an appropriate segmentation method are used. These segments are called tokens. Tokens can be used to represent segments at different granularities depending on the sentence's segmentation. The process of transforming text sequences into sequences of tokens is tokenization and is the first process described in this section. Tokenization is not the end of the transformation pipeline necessary to use text with neural networks. These models operate non-linear transformations on vectors. The tokens need to be transformed into vectors containing relevant information. The different techniques used to create appropriate representations are explored in the second subsection. These representations include semantic information and structural information.

2.2.1 . Tokenization

In this work, every experiment is done on strings of text. Transforming text into our models' format is the first step of every experimental pipeline made. It is important to understand the techniques used to process text. The first step of this process is tokenization. Tokenization is the technique with which a string of text is split into relevant substrings called tokens. Dividing a sentence into a sequence of relevant tokens is a problem. The first idea would be to use words as

tokens. As previously stated, this might have been a good idea if strong criteria existed to define and separate what segments were words. Even if words could be separated perfectly, they are not the only interesting features of sentences. Punctuation or other segments of characters, such as URLs, dates, or numbers, can have relevance for the analysis of the sentences. As such, they need to be represented by tokens. The simplest approach to separating tokens is splitting the sentences over white spaces. This method mostly works for western languages, but other languages, such as Japanese or Mandarin, do not contain white spaces. Techniques presented later can deal with such cases. The problems of such a simple approach can also be seen with western languages. These tokenizers would have problems with punctuation like the simple tokenizer in Figure 2.1. The punctuation will not be split from the words in the tokens, thus multiplying the occurrences of the same words with different punctuation attached to the vocabulary. New combinations of words and punctuation would produce unknown tokens. The unknown token is a token used to represent all the out-of-vocabulary tokens. This token keeps some integrity for the structure of the text, but the information is ultimately lost. In the case of the combination of words and punctuation, the model loses access to information that would be easily available. Pre-processing can be done to alleviate some of these issues. Punctuation can be split off of the words to obtain better representation. There exist other problems similar to the problem of punctuation involving casing. A change of word casing means additional entries in the vocabulary or more usage of the unknown token. This problem can be minimized by normalizing the case of the words. Normalization might, however, result in a loss of information. In our European framework, word case contains information about sentence boundaries, as most sentences start with an upper case. Some tokens will not have the same nature with upper or lower case and will represent different types of entities, such as the word "brown," which is a color, and the word "Brown," which is a family name. Finally, the case can also represent tone. In less formal texts, such as tweets, sentences or words in uppercase often represent shouting, which might be useful for downstream tasks.

The goal of the tokenizer is to represent the strings of text reliably. If two strings are close, it would be easier for the downstream model to manipulate two close representations.

Different approaches to this problem have been explored. More meaningful representations might be found by dividing the word-level tokens into multiple sub-word tokens. This can be seen to a certain extent in the example of "It's" being divided into "It" and "'s" in Figure 2.1. Character representations could be a solution to obtain a more generic approach. Byte-level representations have also been used. Such methods are even more generic as they are agnostic to character encoding.

Byte pair encoding [GAGE, 1994] (a.k.a BPE) is a compression technique originally used to accelerate pattern matching. This algorithm learns a set of merging rules to compress the data. These merges work by pairing the bytes. If you replace the bytes with the base set of symbols used to write text, this method can produce merging operations that create sub-words. For example, in English, the letters "t" and "h" are often found together, which means that there is would likely

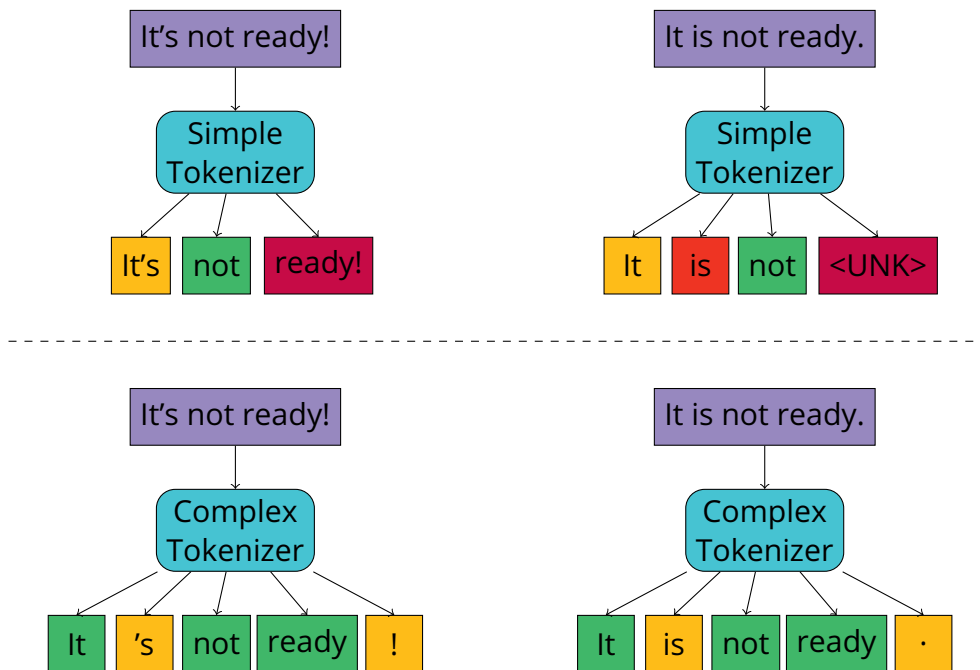


Figure 2.1: Examples of a simple tokenizer versus a complex tokenizer on two close sentences: "It's not ready!" and "It is not ready." On the first line, the simple tokenizer produces two sequences of tokens of different lengths, with only one token being shared between the two sequences. On the second line, the complex tokenizer produces two sequences of tokens of the same length with three exact matches and two similar tokens for the parts of the sentence that have changed.

be a merge operation that combines them into a subword "th" when possible. The compression algorithm becomes an algorithm that produces a sequence of tokens out of a sequence of symbols. BPE can be used to transform words into sequences of sub-words by using the bytes or the characters as the base unit. BPE tokenizers learn merge rules that produce a new token for the most frequent pair of tokens until the maximum vocabulary size is met. As long as the characters used to produce the words are part of the vocabulary of characters, BPE tokenizers should be able to produce a token or a sequence of tokens that better represent the input words than any unknown token would. This technique has recently been used to improve NLP performance for out-of-vocabulary words. Machine translation [Sennrich et al., 2016b] was the first task to use this type of encoding.

Wordpiece [Schuster and Nakajima, 2012, Wu et al., 2016] is another representative of the sub-word tokenizers. The principle is to create subword or word representations by iteratively adding tokens that are mergers of other tokens present in the vocabulary. The choice of the combinations is made by seeing which combination increases the likelihood of a language model built on the training set with the current vocabulary. This process is repeated until either a maximum vocabulary size or a quantity of likelihood is reached. Unigram [Kudo, 2018] is a subword tokenizer built with the aim of regularizing neural language model training. This subword tokenizer does not produce a unique sentence segmentation but contains probabilities for the appearance of each token. The segmentation can then be done by taking the most likely sequence or by sampling the tokens with the probabilities. SentencePiece [Kudo and Richardson, 2018] is another subword

tokenizer. While previous tokenizers presuppose that the sentence is already split into word-level representations, this tokenizer does the entire sentence-to-subword pipeline. This tokenizer implements both BPE and unigram with further optimizations. SentencePiece is a lossless tokenizer, which means that it is built to be able to reconstruct the text exactly as it was before tokenization.

Most recent transformer models use subword tokenizers. This tokenization has helped capture a better representation of words that would not have been included in the vocabulary. Subword tokenizers have also reduced the vocabulary size, which is important for the modeling task. Text can be split into relevant sequences. The next subsection describes how those tokens are represented for modeling.

2.2.2 . Vectorization

Transforming text into information-relevant segments, or tokens, is the first step toward statistical modeling. This section presents the different techniques used to represent tokens, a form of discrete data, in a vector space. The techniques that use basic heuristics to represent the context are first presented. The embeddings and the different techniques that have an impact on learning them are then presented. Finally, the current works around the representation of structural information in the embedding space are presented.

	a	and	cat	cats	dogs	have	I	like	They
I like cats	0	0	0	1	0	0	1	1	0
They like dogs	0	0	0	0	1	0	0	1	1
I have a cat	1	0	1	0	0	1	1	0	0
They have cats and dogs	0	1	0	1	1	1	0	0	1

Table 2.1: Bag of words examples. On the left column are the sentences, and on the right columns are the vector weights associated with the tokens at the top of the column.

A bag of words, or BoW, is a method to generate a vector representing which words or n-grams are within a sentence. This method creates a vector of the vocabulary size with the number of occurrences of each word in the sequence of text. Examples illustrate this approach in Table 2.1. BoWs are a simple method to produce vectors representing a sequence of text. This method is prone to many problems. Firstly, due to the counting of the words in the sentence, words with low semantic meaning and high frequency might create problems. These stop words, such as [the], [and], [it], etc., help structure sentences but bring a low amount of information. Secondly, this vector is very large since its size is the same as the vocabulary size. Finally, this vector does not contain information about word order, which may Term frequency-inverse document frequency, or TF-IDF, functions in a similar manner. It constructs a vector with the frequency of appearance of words within a sentence weighted by the inverse frequency of the word within the document the sentence is taken from. This inverse term lowers the weight of stop words which bring little semantic information. Those two methods are not usable as such for word prediction. Still, they represent the first type of information used

to build vectors which is the co-occurrence of words within an environment, here the sentence.

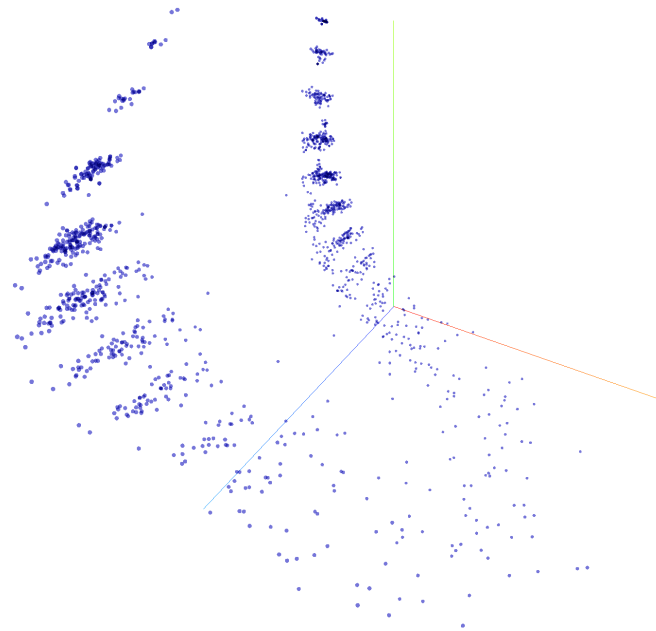


Figure 2.2: Naïve representation of the power of representation of vectors. This is a vector representation of sentences in the latent space of an auto-encoder using LSTMs after one epoch of training and. A PCA was applied on the vectors to obtain this representation. The two ribbons correspond to the presence of either a full stop or an interrogation mark at the end of the sentence. The stripes correspond to the length of the sentence.

Embeddings are a means to represent semantic objects in a vector space. They can represent tokens, sentences, and images in a vector space. An illustration of the power of representation of vectors is shown in Figure 2.2 where the representation of sentences in the latent variable of an auto-encoder at the end of a single epoch of training. The space is very structured during this training period and only contains simple structural information about the sentences encoded. Exploration of these objects has been on the rise alongside the popularization of neural networks. Embeddings are technically a lookup table linking the token to a vector representation. There are two interesting factors to embeddings, the first is how to obtain them, and the second is how information is contained within these embeddings. In the following paragraphs, the most impactful training methods for embeddings are reviewed, and the types of information retrieved from these representations are discussed.

Continuous bag-of-words [Mikolov et al., 2013] representations use a feed-forward neural network language model [Bengio et al., 2000] without the hidden layer trained with the objective of predicting the current word based on the context. Skip-gram [Mikolov et al., 2013] representations also use a feed-forward neural network language model without the hidden layer, this time trained with the objective of predicting the context based on the current token. These embedding techniques are illustrated in Figure 2.3. These embeddings were designed with the idea that other language models could use them. They were effectively used

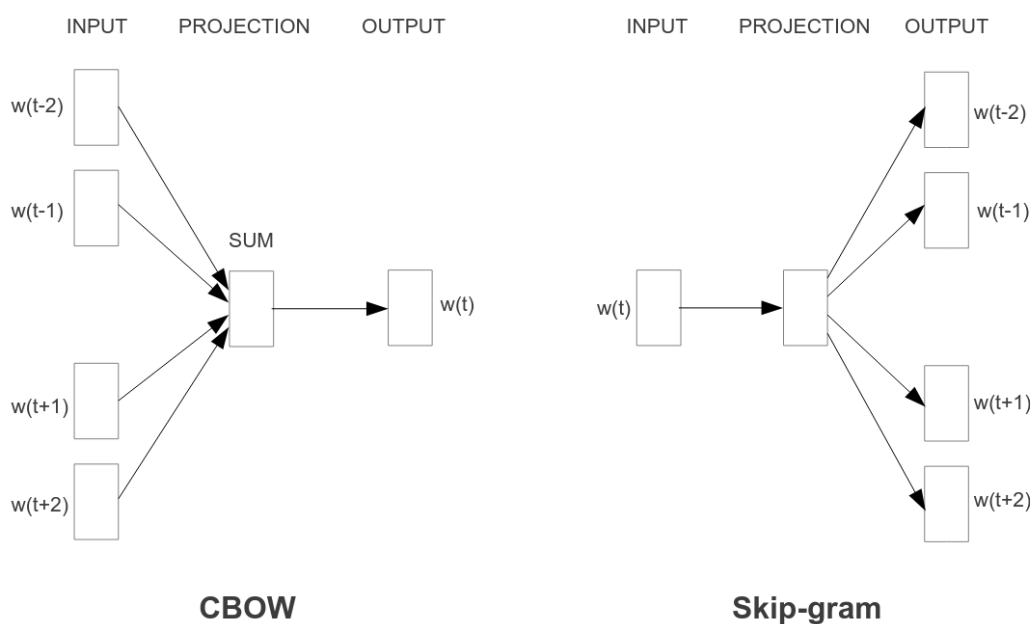


Figure 2.3: CBOW and Skip-gram description from the original article [Mikolov et al., 2013]. The CBOW embeddings are learned using the context as an n-gram input to predict the current token. The Skip-gram embedding is learned by predicting the context n-gram from the current token.

across various models. GloVe [Pennington et al., 2014] embeddings pushed the analysis of the capabilities of embeddings further. The technique used to produce GloVe embeddings is a factorization of the co-occurrence matrix. FastText [Bojanowski et al., 2017] embeddings use the skip-gram approach over a sub-word representation of the tokens. These embeddings have been at the forefront of neural network-based NLP before the rise of large model fine-tuning. They have improved models' performances on sequence labeling tasks [Chiu and Nichols, 2016]. These improvements have raised questions on what properties of pre-trained embeddings were the cause of these improvements. Outside of being a compressed representation of the large quantity of data they were trained on, pre-trained embeddings have shown that relations existed between certain types of tokens. For instance, if you take the vector of the token "Paris" and you subtract the vector of the token "France," you obtain a vector representing the relation "is capital of." If you add this relation vector to "Germany," you might obtain a vector close to "Berlin."

Word embeddings were initially used to represent the tokens independently of the context. The context was used to train the embeddings, as seen with CBOW, Skip-Gram, and GloVe, but these vectors represented a global representation of the token. The model's job was to use these embeddings as the initial representation of the token and add contextual information to them. New models looked at the representation of tokens through character representation. FastText and ELMo [Peters et al., 2018] are representatives of this exploration of the character-level space. The ELMo embeddings are derived from a language model, which is composed of a character-level convolution network and a two-layered BiLSTM

with residual connections on top. The representations derived are a linear combination of the outputs of each layer. This model is used to produce the representations on a sentence-to-sentence basis, not to produce a fixed matrix of embeddings. As such, ELMo takes into account local context when producing representations. In a way, embeddings were one of the first sources of transfer learning for neural networks. One of the first use of recent context-focused models were the creation of contextual embeddings. Rather than fine-tuning the models for the task, which was a more costly option, the pretrained models were used to generate embeddings with contextual information. Models such as ELMo and BERT [Devlin et al., 2018] were at the forefront of this movement. The BERT and the transformer architecture are described with more details in a future section and in Figure 2.7. This model was designed to produce embeddings by combining the outputs of the last layers. One of the major changes BERT brought to the embedding landscape is the use of subwords and not character-level representation. Using BERT and its' derivatives became a standard approach in NLP due to the performance gains it often brought.

The transformer architecture [Vaswani et al., 2017] has become one of the most used architectures for large language models, or LLMs. Embeddings used to represent the tokens need to include positional information for transformers. Multiple methods have been designed to include structural information in the embedding space. Absolute positional embeddings added to the semantic embeddings have been used to include structural information [Vaswani et al., 2017]. In this case, one vector is assigned to every position. These embeddings can take two forms: a static form based on a mathematical formula or a trainable form. Rotary position embeddings [Su et al., 2021], or RoPE, have been a highly represented large language model embedding technique. They are rotation matrices dependent on the position of the token. The semantic embeddings are multiplied by these rotation matrices to obtain the final representation.

Most recent models have stopped using pre-trained embedding matrices in favor of fine-tuning the model producing the vectors. This stop comes from the wider availability of high-capacity computing resources and the better representation of contextual information obtained with a fine-tuned model. During this Ph.D., we used pre-trained embeddings in our first publication at RÉCITAL [Bou-langer, 2020] but have since moved to transformer-based model fine-tuning for those reasons. However, these models still need an initial vector representation of the tokens, which they train during training. Transformers also need positional encoding to better represent the structural information. Embeddings are the first block of many language models. The next section explores language modeling, focusing on deep learning techniques. Most of these techniques require text to be processed through the tokenization and vectorization techniques presented above.

2.3 . Language Modeling

Transforming a text into a sequence of representative vectors is the first step to processing text using statistical models. The role of the model is to take these

pieces of information and compute probabilities related to the tasks they are trying to solve. Language modeling is the task of computing the probability distribution of tokens at a given position while knowing other parts of the sentence. Thanks to transfer learning, many NLP tasks have seen improved methods using language models at the center of their pipeline. Every aspect of models, from architectures to the tasks on which they are trained, has seen plenty of development. The task of language modeling is a task of text generation conditioned on the previously known tokens. Language modeling is a key component of NLP and a possible method for text generation. We will describe the language modeling landscape in this section for these reasons.

2.3.1 . Introduction

Language modeling is the computation of the distribution of probabilities over a sequence of tokens. The parameters and architecture used to obtain the distribution is the language model. Modeling can be done in two different manners. The model can produce the probability distribution for each token simultaneously or sequentially. Simultaneous prediction is called non-autoregressive (NAR) modeling. This method is generally faster to compute because there is no need for iteration over the sequence of generated tokens. While it may be interesting for downstream tasks, this type of modeling offers fewer possibilities in terms of generation.

The most common way of doing language modeling is to sequentially model the probabilities of the tokens. This method of generation is called autoregressive (AR) modeling. AR modeling takes longer to train and to use at inference due to the computation of the model over all the tokens up to the current token for every token of a sentence being modeled. However, AR modeling tends to produce better results than NAR modeling. The performances of AR modeling depend on the algorithm used to choose which token to keep for the following inference.

Obtaining a probability distribution for the tokens helps find the most likely tokens for text generation. The model does not, however, make a choice. Decoding algorithms have been designed to explore the space of possible sentences based on the probabilities output by the language models and are first presented. Language model architectures are then reviewed. An overview of the landscape of large language models and the datasets used to train and evaluate them is given. Finally, general model usage and usage in this thesis are discussed.

2.3.2 . Decoding algorithms

Language models produce probabilities for tokens based on what they have learned and the context. However, choosing the token with the highest probability is not always a good solution. The choice of a token will impact the probabilities of the following tokens. Decoding the tokens from a model can be seen as a tree search. However, searching through all the branches of the tree would take exponential time on the size of the vocabulary, which is too long. The second problem of a simple tree search is that, in the case of decoding, traveling to the leaves of a node is very expensive. These two problems combined show the necessity of finding algorithms that can work with partial information to decode a sequence

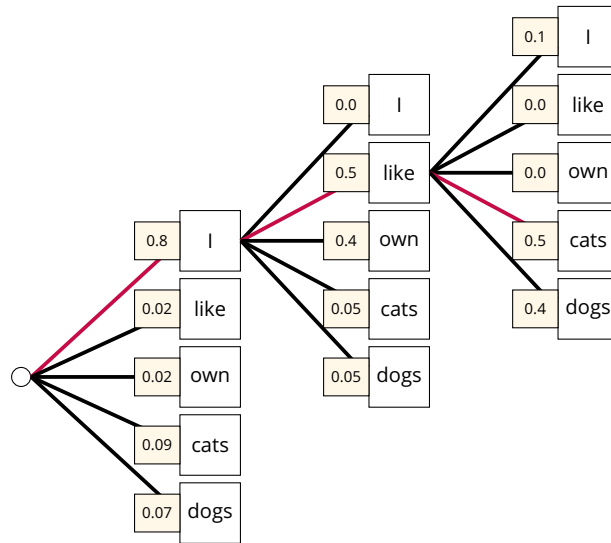


Figure 2.4: Decoding using greedy decoding. The chosen tokens are the token with maximum probability and are indicated with a red line.

of tokens.

The most simple decoding algorithm is greedy decoding and is illustrated in Figure 2.4. Greedy decoding functions by selecting the token with the maximum probability. The strength of this algorithm is its' simplicity, but the simplicity is also its' weakness. This algorithm tends to produce less varied and meaningful sentences because meaningful tokens with slightly lower probabilities get ignored.

The beam search algorithm is a decoding algorithm predicting a fixed number of sequences or beams. This algorithm is illustrated in Figure 2.5. Using a width of 1 results in having a greedy search. This algorithm is supposed to introduce more variety to the generated sentences than greedy search, but in practice, this variety is often localized at the end of the generated sequences. Improvements to the beam search algorithm in terms of diversity have been brought with the diverse beam search algorithm [Vijayakumar et al., 2016]. This algorithm forms groups of beams that are decoded sequentially. Diversity is forced between the groups by reducing the probability of the chosen tokens of previous groups for the current group. Repetitions of sequences of tokens are also a common problem with the beam search decoding algorithm. This problem can be worked around by reducing the probabilities of tokens generating a previously decoded n-gram to 0 [Paulus et al., 2018].

Sampling the next token using the probabilities given by the model is another solution which improves diversity. The highest probability tokens are almost guaranteed to be chosen, but it is also not unlikely to choose a less fitting token. These low-probability tokens can cause some issues. To solve those issues, limiting the sampling to the top k tokens [Fan et al., 2018] is possible. This study states that sampling from the top 10 most likely tokens improved variety in length and avoided repeating sequences. One of the pros of this technique is the fixed number of tokens generated which allows for building other algorithms, such as reinforcement agents on top. It is also possible to limit the sampling to the top tokens

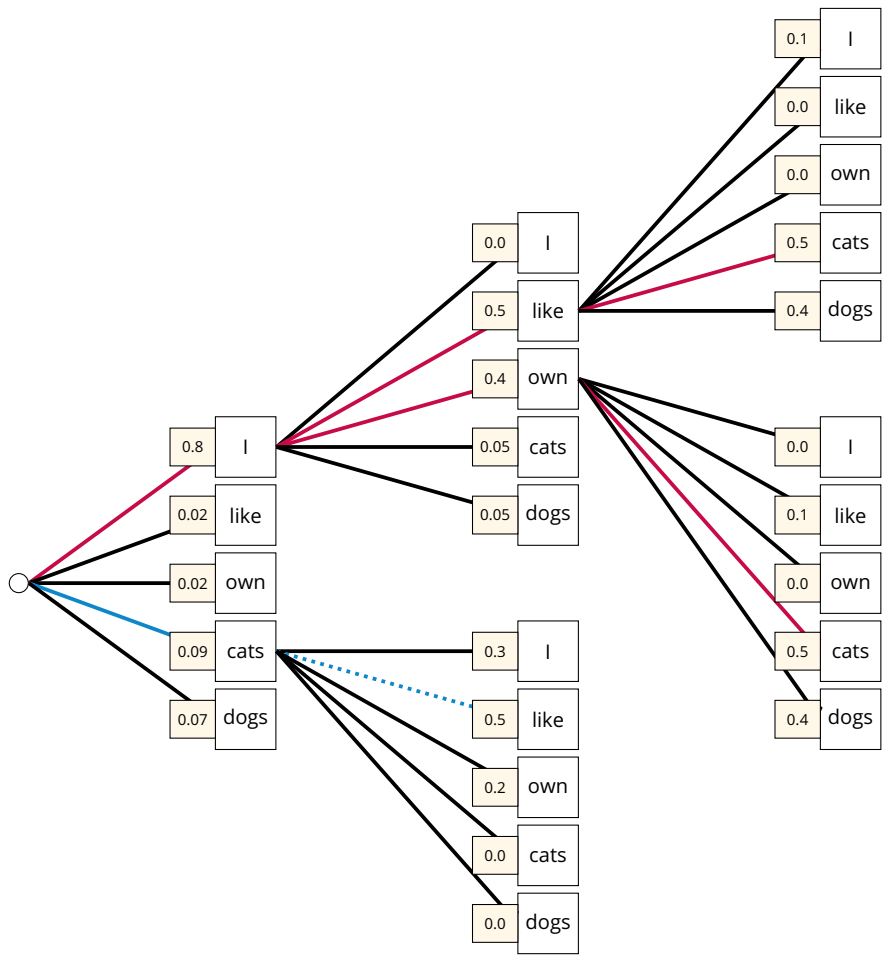


Figure 2.5: Decoding using beam search with a beam of size 2. The first tokens chosen are the tokens with maximum probability: I and cats. The blue beam on the second step, “cats like,” has a probability inferior to another choice from the red beam, “I own,” which explains why it is replaced by this beam.

whose probabilities sum equals p [Holtzman et al., 2019]. This technique, called top- p or nucleus sampling, is more flexible than top- k sampling. These two techniques allow a better quality of sampling, which in turn lead to better sentences.

Simply taking the maximum probability does not produce useful sentences. A combination of rules to enforce diversity and sampling seems to provide the best results in terms of sentence generation. In the following section, the architectures of the models that produce the probabilities necessary for the decoding algorithms to work are presented.

2.3.3 . Model architectures

Model architectures have played a significant role in the performance of language models. The capacities of a model depend on the operations performed by this model. For language modeling, this comes in the form of being capable of capturing multiple scales of context. This section will recap the architectures still relevant for language modeling and downstream labeling tasks. The evolution of architecture has been informed by the development of new techniques and ad-

vancements in hardware technology. The impact of these developments on the landscape of techniques and architectures for language modeling is discussed.

N-gram models are statistical models using the previous $n - 1$ tokens as the context to predict the current token. These models are Markov models. As such, these models follow the independence assumptions, meaning that the current token only depends on the previous $n - 1$ tokens. This hypothesis means that long dependencies are not explicitly represented. As a result, studying language properties on such models is limited. But it was never the goal of such models, which are oriented towards practical applications, such as spelling correction. These models are cheap in computation power but have a scaling problem in terms of memory. The higher n is, and the larger the vocabulary is, the more memory the model takes. Conditional random fields [Lafferty, 2001] (a.k.a CRF) are another type of statistical model following the Markov properties. These models are still mainly used for sequence labeling tasks by adding a layer on top of the other model architectures described below. But they have also been used on tasks close to text generation, such as machine translation [Lavergne et al., 2011].

One of the issues of previous modeling techniques is the vocabulary size. Neural network language models were introduced as a means to counter that problem [Bengio et al., 2000, Schwenk and Gauvain, 2005]. They project the vocabulary upon embeddings, which are limited in terms of the number of features compared to the vocabulary size. These models use the n-gram approach for modeling and combine the representation of the last $n - 1$ tokens to predict the current token. These models were small enough to be computed on CPU. Vocabulary was still a problem for token prediction, and techniques were developed to counter these problems, such as the structured output layer [Le et al., 2011].

Deep neural networks truly had a breakthrough thanks to the advent of parallel computing hardware, namely Graphics Processing Units, or GPUs. This evolution of computing capacities allowed for more complex neural network architecture to be trained within a reasonable time frame. Convolutional neural networks [Le Cun, 1989] (a.k.a CNN) is a type of neural network applying convolution operations to the data and learning the convolution kernels. This type of model is commonly used with images, but it can be used for one-dimensional inputs such as text [Pham et al., 2016, Dauphin et al., 2017]. The convolution operation works as a filter over a context window, which allows a good representation of the local context. Recurrent neural networks (a.k.a RNN) are a family of neural networks that take sequential inputs and, at each step t , use the information computed by the model at step $t - 1$. RNN is also a denomination of models using a tanh to fuse the current input with the previous output. Long short-term memory units [Hochreiter and Schmidhuber, 1997] (a.k.a LSTM) is a type of RNN that shares two types of information between each step. The pieces of information shared between each step are the previous prediction of the unit and a vector modified by the unit. These pieces of information, or memories, are supposed to act as short-term and long-term memory, respectively. Gated recurrent units [Cho et al., 2014a] (a.k.a GRU) are a type of RNN that uses gates rather than tanh to fuse the input with the previous output. Gates were previously used by LSTMs to mix the different memories and inputs. Increasing context information is key to im-

proving model performance. While RNNs improved the context by keeping some information from previous predictions, the biggest improvement was to use of bidirectional versions of the architectures. Having one model in each direction helps language modeling by giving a broader context to the token. Sequence-to-sequence [Cho et al., 2014b] modeling uses an encoder-decoder model architecture where the encoder’s goal is to encapsulate a sentence’s different aspects, and the decoder’s goal is to use this encapsulation to create an appropriate response. An illustration of the sequence-to-sequence architecture can be found in Figure 2.6 which represents the dialogue response generation task. This technique was prominently used for machine translation, dialogue response generation, and summarization. This approach has been used for both RNN-based models and CNN-based models [Cho et al., 2014a].

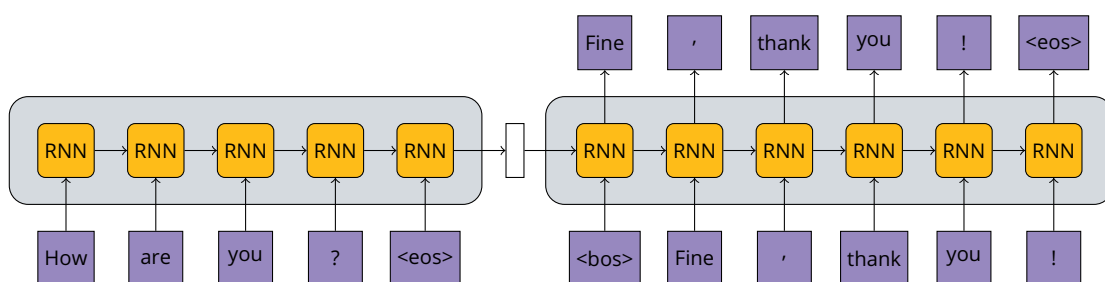


Figure 2.6: Sequence to sequence model with RNNs.

The self-attention mechanism is a re-weighting technique based on vector proximity [Cheng et al., 2016]. This technique has been introduced to improve long-distance relations with LSTM models for sequence-to-sequence modeling. Each output becomes a linear combination of the outputs, with weights being the dot product between the output computed and the other scaled, so the sum equals 1. While the principles of self-attention have permitted a better representation of long-distance relations between words, multi-headed attention has been a revolution for NLP. This type of attention layer was introduced alongside the transformer architecture [Vaswani et al., 2017]. Instead of computing attention on one set of fully-sized queries, keys, and values, multi-headed attention computes multiple attention in parallel on h smaller set of queries, keys, and values. These queries, keys, and values are obtained by projecting the original queries, keys, and values to a smaller size through linear projection layers that are independent and trainable. The output of the multiple heads is then concatenated and projected to the size.

Transformers [Vaswani et al., 2017] is an architecture of models using multi-headed attention layers as the core of the architecture. A description of the architecture can be seen in Figure 2.7. This architecture is at the center of recent language modeling improvements to performances. This architecture uses an encoder-decoder format. The encoder and decoder are blocks of N consecutive layers, respectively the encoder and decoder layers. The composition of the encoder and decoder layers is different, but they are both built with multiple blocks made of a computation layer and a normalization layer. The computation layer can be either a feed-forward network or a multi-head attention layer. The blocks

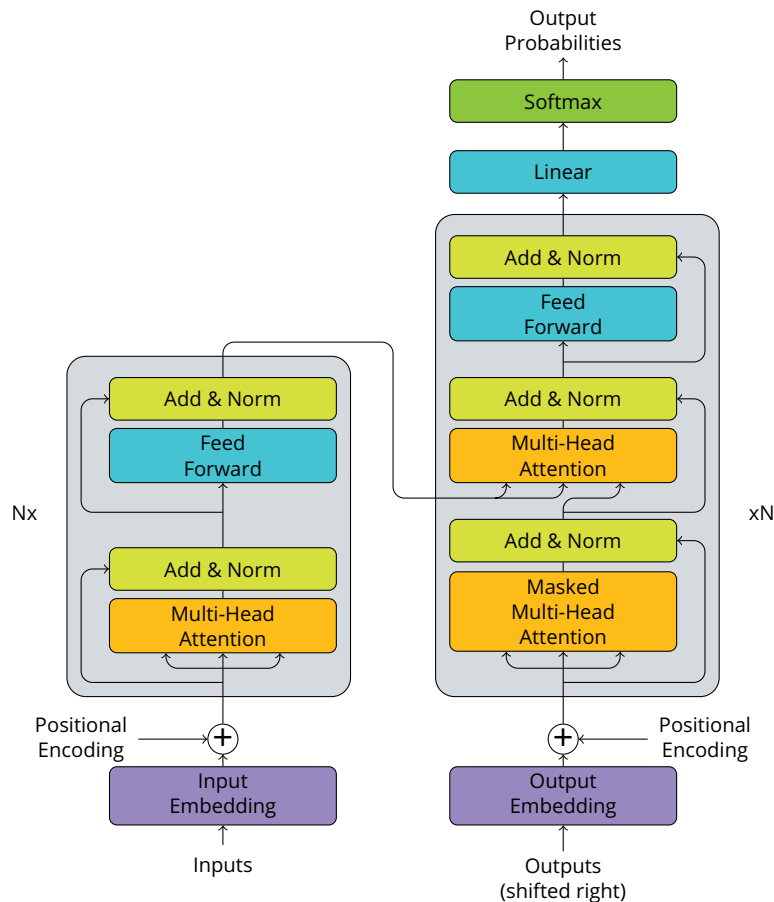


Figure 2.7: Transformer architecture from Attention is all you need [Vaswani et al., 2017]. On the left side is the encoder layer repeated N times and on the right side is the decoder layer which is also repeated N times.

will be referred to by their computation layer type. Within a block, the outputs of the computation layer are summed with the inputs before being normalized through LayerNorm [Lei Ba et al., 2016]. The encoder layer, seen on the left in Figure 2.7, is composed of a multi-headed attention block and a feed-forward block. The decoder layer, on the right of Figure 2.7, is composed of three blocks, two multi-headed attention blocks, and a feed-forward block. The first multi-headed attention layer takes masked inputs, as the inputs are, in fact, the previously generated tokens. Rather than padding, what would be the padded space is masked. The second multi-headed attention layer of the decoder takes the encoded inputs from the encoder and uses them as the keys and queries, while the outputs of the first block are used as values. The encoding part is done in a non-auto-regressive fashion, while decoding is done in an auto-regressive fashion. Many instances of this model use either the encoder, the decoder, or both.

The transformer architecture has been widely adopted for language modeling. These models are often pre-trained on generic data and then fine-tuned on downstream tasks. This sort of transfer learning strategy is costly as it requires all parameters to be fine-tuned. Most transformer-based models are huge, with the number of parameters ranging from tens of millions to hundreds of billion. These

models can exist thanks to the ever-increasing size of GPUs and Tensor Processing Units (TPUs) but also thanks to the creation of larger clusters of computing power and the parallelism techniques going with it. Fine-tuning these models is costly in computing power, even for the smaller ones. Adapter [Houlsby et al., 2019] layers were designed to counterbalance this problem. These layers are added between the multi-headed attention or feed-forward networks and the residual connection. While training such a model, only the adapter layers are trained. This makes this technique a highly efficient technique for training transformer models on downstream tasks.

The current landscape of neural-based NLP is to use fine-tuned pre-trained models to adapt them to the task at hand. Most pre-trained recent language models use the transformer architecture. In the next section, the reasons why this architecture and those models have taken over the entire neural-based NLP space are explored.

2.3.4 . Pre-trained Large Language Models

Recent progress in language modeling has been made through an increase in parameter count and a higher quantity of training data. Since the advent of transformers [Vaswani et al., 2017], many models using this type of architecture have emerged. This architecture has shown that performances scaled with the number of parameters. These models can be used to try to solve tasks in a zero-shot or few-shot manner [Brown et al., 2020]. This is helped by the various ways the models have been trained. These training objectives are part of the self-supervised paradigm. This learning paradigm is as data-oriented as it can get as the model learns the task through the unlabeled data it is trained on. The multiple training objectives encountered in this section are causal language modeling, masked language modeling, and translation language modeling. Causal language modeling is the classic unidirectional modeling objective. The model is used to predict the token at the end of the sentence using the context from a single side. The common use is to have the context being the previous tokens from the sentences. Masked language modeling is a modeling objective where a percentage of the sequence is replaced by mask tokens. The goal of the model is to predict the tokens that have been masked. Translation language modeling is a task resembling masked language modeling in the sense that the goal is for the model to replace mask tokens in the token sequence. Masked language modeling and translation language modeling are illustrated in 2.8. The token sequence is built off of two aligned sentences from two different languages. The alignment comes into play during the masking process. The tokens masked in the first sequence are not masked in the second, and vice versa. This means that information exists only in one language in the input. In the following section, we go through the different pre-trained transformer models that have had a significant impact on neural-based NLP.

BERT [Devlin et al., 2018] is an encoder-only transformer initially designed as an embedding-producing model. This model is non-auto-regressive and is trained using masked language modeling and next-sentence prediction. The input on which BERT is trained contains two sentences separated by a separator token. The next sentence prediction objective consists in having the model predict whether

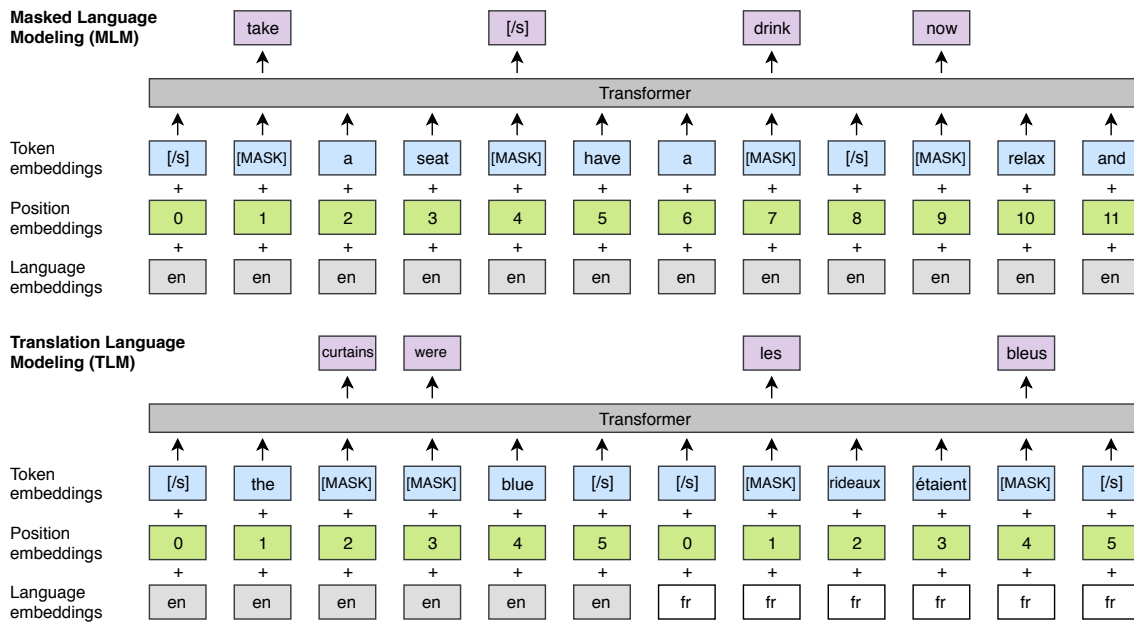


Figure 2.8: Masked language modeling and translation language modeling tasks description from the XLM article [Conneau and Lample, 2019].

the second sentence follows the first. This objective trains the model to produce a representation of the relations between the two inputs, which can be used for downstream tasks such as natural language inference or question answering. This model is not trained for causal language modeling, but it can be used for augmentation like in NLP Aug [Ma, 2019] and to work on downstream tasks such as the sequence labeling task. It is on these downstream tasks that BERT has become a staple of neural-based NLP. BERT set a trend of other models with similar architectures but different specificities. RoBERTa [Liu et al., 2019] is an optimization of BERT by modifying the training objective, optimizing the hyperparameters, and using a new training dataset. BART [Lewis et al., 2020] is a denoising auto-encoder using a transformer architecture with a bidirectional encoder. A Lite BERT or ALBERT [Lan et al., 2020] is a light version of BERT thanks to cross-layer parameter sharing and splitting the embedding matrix in half. Techniques like these have been developed in order to reduce the size of the models while keeping similar performances.

The GPT family of models [Radford et al., 2018] is a staple of the scaling capacities of transformers, with their parameter counts ranging from 110 million parameters for the first iteration to 175 billion parameters in the last iteration of the model. These models are decoder-only transformer models. Contrary to the way BERT was pre-trained, GPT was pre-trained on the classic auto-regressive modeling task. GPT became state of the art on multiple downstream tasks thanks to its pre-training technique and the multi-task fine-tuning objective. This multi-task objective is the combination of the modeling task alongside the downstream task at fine-tuning. The impact of this multi-task objective can be felt in the following models. GPT-2 [Radford et al., 2019] aimed at solving the multi-task learning problem in an unsupervised manner. The multi-task objective is viewed with a

meta-learning perspective; as such, each task is viewed as a training example. To be able to generalize would mean to be able to obtain and produce thousands, or even more, corpora, each dedicated to its task. This would also require the model to accommodate each of those tasks with their structure. The authors of GPT-2 assume that most NLP tasks can be formulated through language alone and that many of these tasks have been asked, described, and solved by people. As such, their approach is to build a corpus that contains as many tasks and domains as possible. Their first approach was to use Common Crawl, a large internet scrape, but the data was too fuzzy to be used as is. They instead built a corpus of human-validated data. They built the WebText [Radford et al., 2019] dataset, which contains the linked content of Reddit posts with over three user appreciation points. The use of this larger corpus and a few changes in the architecture, such as the change of how layer normalization was applied throughout the model, the extended vocabulary size, the extended context size and batch size, and the addition of larger models have shown that the transformer-based model performances scaled with parameter count and that tackling tasks through language modeling alone was feasible. GPT-2 can be applied to other tasks through conditioning applied at inference. Conditioning in this setting means adding information in front of the sequence of text meant to be processed. This conditioning was applied for translation and meant adding a few solved examples of the task as context for the model to understand what to do.

The study of this mechanism was pushed further with the creation of GPT-3 [Brown et al., 2020]. This model is similar to GPT-2 with further refinements, including alternating dense and sparse attention layers as in the sparse transformer [Child et al., 2019]. GPT-3 was also trained on a large corpus composed of CommonCrawl, WebText, and Wikipedia examples. GPT-3 was made with the explicit intent to use conditioning as a means to solve tasks. Conditioning can be categorized depending on the quantity and nature of the data used. They consider the zero-shot setting as a setting where the conditioning is done with a description of the task written by the users. An example of a sentence conditioned for the English to French translation task would be: "Translate from English to French </s> I love cats." The few-shot setting does not include the description but n examples of the task being solved. These methods of conditioning have inspired a new wave of exploratory work on text conditioning or prompting. The family of GPT models have shown that language modeling could be directly applied to the tasks at hand. The large size of the models and their capacity to be conditioned on a prompt has led to new techniques such as prompt design, where the user creates prompts using available tokens, or prompt tuning, where new special prompt embeddings are trained to adapt the model to the task without changing the parameters of the model.

Another model has explored the principle that all tasks can be formulated in a language-modeling fashion. T5 [Raffel et al., 2020] is the model which explored this paradigm. This model is an encoder-decoder transformer pretrained on a task resembling BERT's masked modeling task. This task is the span replacement task, in which spans of tokens are masked in the input, and the decoder is tasked with the production of the spans. The dataset produced for the pre-training step

is a subsample of 750 GB of Common Crawl named C4. Two versions of the model exist, a version fine-tuned on downstream tasks (known as T5) and a version only pre-trained (known as T5v1.1). While the pre-training objective of T5 was peculiar, its fine-tuning developed the model's potential. All the downstream tasks of T5 were reformulated to be learned as language modeling tasks. Part of the reformulation of the task was the addition of prompts at the beginning of each sentence, which would help the model associate with the task. A multilingual version of the model exists [Xue et al., 2021]. The particularity of the pre-training of this model is the span replacement objective used. The model is trained with sentences where around 15% of the sentences were masked and is tasked with producing the different missing spans in a structured fashion.

Other pre-trained LLMs have had impacts on various parts of NLP. With the improvement of language modeling on single languages, multilingual modeling or cross-lingual modeling became a target. XLM [Conneau and Lample, 2019] is a transformer-based model pretrained on three tasks. Those tasks are causal language modeling, masked language modeling, and a combination of masked language modeling and translation language modeling. Transformer-XL [Dai et al., 2019] is a decoder transformer architecture with a caching mechanism that stores long dependencies. As such, it allows faster training and longer sequences to be processed. XLNet [Yang et al., 2019] is a decoder transformer architecture that uses the caching mechanism from Transformer-XL and is trained by decoding auto-regressively but in a disordered manner. The training objective of XLNet is permutation language modeling. This objective is similar to left-to-right or right-to-left auto-regressive modeling, but in this case, the order of decoding can be permuted. Thus it allows auto-regressive modeling to use bidirectional information. Bigscience Large Open-science Open-data Model [Scao et al., 2022], or BLOOM¹, is a recent large language model aimed at providing a public research implementation of a large transformer model. The particularity of this model is the transparency of all the parts of the process of its training.

One of the methods to use ever-larger models without increasing the computational costs is conditional computing. Conditional computing is a method in which part of the model is deactivated on a per-example basis [Bengio et al., 2013, Davis and Arel, 2013]. The basic block of this method of computation in neural networks is the sparsely-gated Mixture-of-Experts (a.k.a MoE) layer [Shazeer et al., 2017]. The MoE layer functions by having a gate selecting two expert layers to perform the computations on and modulating the combination of the results. Switch-C [Fedus et al., 2022] proposes an encoder-decoder architecture with 1.6 trillion parameters similar to T5 but using improved MoE layers. GLaM [Smith et al., 2022] is 1.2 trillion parameters decoder-only architecture using MoE layers. These MoE-based models effectively use fewer parameters during inference, respectively 1.5 billion and 96.6 billion active parameters. Other models push the engineering limits, such as PaLM [Chowdhery et al., 2022], as the 540 billion parameters of this model are all active at all times. Pushing the size of models higher is an on-going race that must be highlighted, and puts into perspective the size of the models used in this thesis which are not so large compared to these mastodons.

¹<https://huggingface.co/bigscience/bloom>

Scaling the models has been a computational arms race. Datasets are necessary to train these models. These datasets have a significant impact on the performance of the models on downstream tasks [Cattan et al., 2022].

2.3.5 . Data

Training LLMs require a high quantity of data. What a high quantity of data means has evolved through time. Using large datasets to train models is not new; the Penn Treebank corpus [Marcus et al., 1993], a corpus made for part-of-speech tagging, has often been used to train language models. Wikipedia has been a constant source of data for multiple languages. WikiText-2 and WikiText-103 are two datasets made from favorited and high-quality articles containing respectively 2 million and 103 million tokens [Merity et al., 2017]. The BooksCorpus [Zhu et al., 2015] dataset used by GPT contains around one billion tokens across 74 million sentences. WebText [Radford et al., 2019] is a corpus of 40 GB of Reddit-linked documents used to train GPT-2. The documents kept were documents where the Reddit post obtained at least three points of user score, ensuring at least a small quantity of humans gave positive reviews to the post. C4 [Raffel et al., 2020] is a 750 GB corpus of Common Crawl data used to train T5. This dataset is built using a set of heuristics to filter out noisy, irrelevant, or duplicate text. The trend for language model pre-training is to use more and more data.

Not everyone has access to the same quantities of data. It can be difficult to obtain large quantities of data in a target language, but efforts are being made by the communities of less prolific languages such as Maltese [Micallef et al., 2022] to produce corpora and LLMs on these languages. The quality of the data can impact performances on downstream tasks. Camembert [Martin et al., 2020] obtains varying performances on natural language understanding tasks depending on the corpus it was trained upon [Cattan et al., 2022]. Specialized LLMs are trained on corpora collected with the intent to contain more information about a domain. Corpora such as the PMC OA² or MIMIC-III [Johnson et al., 2016] contain information through the articles, and that can be of use to train medical LLMs.

One of the goals of training ever larger LLMs was to improve performances in a zero or few-shot setting [Radford et al., 2019]. Benchmarks to evaluate the performances of such models have been designed by collating existing benchmarks. The GLUE [Wang et al., 2018a] benchmark is a composite benchmark designed to evaluate multiple types of downstream tasks. This benchmark has been heavily used by LLMs to display their performances.

While this is not an in-depth analysis of the multitude of corpora available, it shows that the size of the corpora necessary to train large language models is large. The quality and specialization of the data are also important for the performance of the end model. In light of these constraints, training high-performing language models from scratch is not a viable approach for this thesis which focuses on low-resource generation. Pre-trained LLMs are, however, used for text generation without any additional training in this thesis. We will also fine-tune models for sequence labeling.

²PubMed Central Open Access

2.3.6 . Language modeling in this work

Language modeling is a great tool for NLP. Language models have evolved to contain the condensed representation of the knowledge present in the training dataset or to gain the capacity to represent contextual information. With the rise of powerful parallel computing hardware, namely GPUs, large language models have become the centerpiece of recent neural-based NLP models. Large Language Models are used to tackle the sequence labeling task in our experiments. In most of our experiments, the results are computed using a model from the BERT family of models, fine-tuned on the task. For our earlier experiments, BiLSTMs were used as the computational power required to train them was much smaller. These models were used as they were easier to train from scratch, and the method to use them was tried and tested. Our aim is not to bring improvements to the sequence labeling pipeline.

Our aim is to develop techniques of data generation that improve sequence labeling when a low quantity of labeled natural data is available. Exploration of the generative capacities of language models is done in this thesis. LLMs have few-shot or zero-shot capacities for downstream tasks, our work does not focus on these capacities, but our hypothesis is that they can help produce the synthetic examples for sequence labeling. Causal language modeling is the most direct way an LLM can generate new text with. This thesis features GPT-2 and BLOOM as models using this type of language modeling. Our hypothesis is that other types of modeling, such as text-to-text modeling, can be used for augmentation. T5 and mT5 are used for their span replacement capacities.

2.4 . Vector-to-Text generation

As seen previously, language modeling is a tool that can produce text with the help of contextual inputs. These inputs are generally sequences of text that will direct the generation based on how the model was trained or based on the nature of these contextual inputs. Some techniques have been developed to manipulate parts of these inputs to guide the model into producing outputs that try to solve tasks. Prompt design is a technique that focuses on appending a sequence of text designed by humans to push the model to produce the wanted outputs, but another technique, prompt tuning, uses a sequence of vectors trained to condition the model into producing the wanted outputs. This idea of conditioning a generative process upon a vector is not new. This section explores the different model architectures that construct text from a vector representation.

2.4.1 . Autoencoders

One of the model architectures that can be used to generate text is the autoencoder architecture [RUMELHART, 1986]. This type of model uses two sub-models, an encoder and a decoder, as can be seen in Figure 2.9. The goal of this architecture is to learn how to reproduce its input. This is the main difference with sequence-to-sequence modeling, in which the decoder produces an output based on the task it is trained on. The encoder transforms the input from its original representation to a singular vector. The space of this vector is called the latent

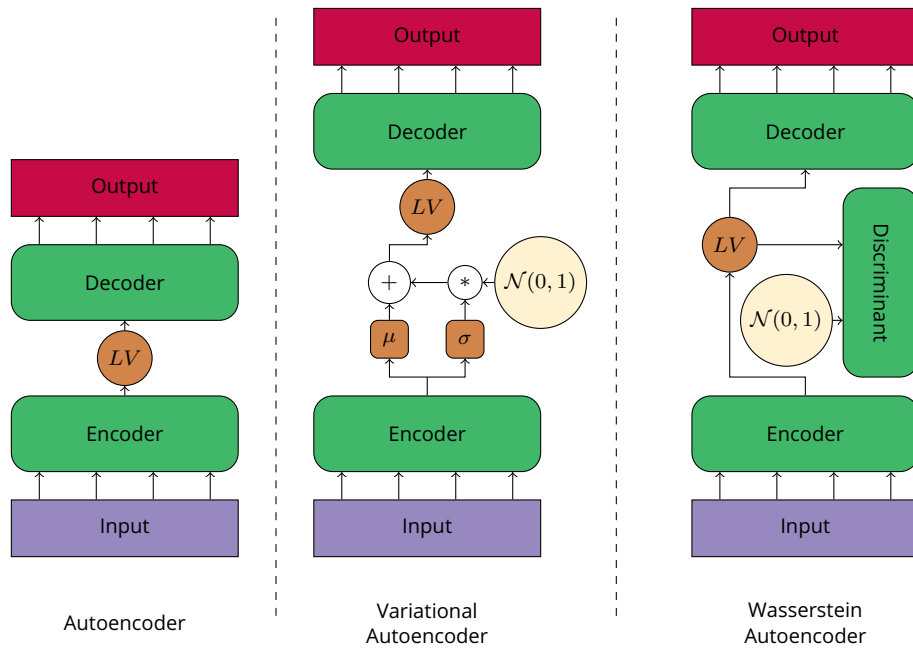


Figure 2.9: Autoencoder families described in this section. At the left is the most simple autoencoder architecture. At the center, the Variational Autoencoder with the reparametrization trick is displayed. At the right is the Wasserstein Autoencoder with its use of a discriminant network for latent variable constraint.

space and is supposed to represent the data space in a compact representation. The decoder then uses this latent representation to reconstruct the inputs of the encoder.

From this perspective, autoencoders are a good candidate for generating labeled text. Finding a way to sample from the latent space is needed. Fortunately, constraining the latent space to a known space is a technique that exists and is widely used within variational autoencoders [Kingma and Welling, 2014]. This architecture uses the Kullback Leibler divergence [Kullback, 1968] (KL-divergence) to constrain the latent space to a Normal multivariate distribution. Due to this constraint, the latent variable needs to be formed by sampling a vector from a multivariate normal distribution, multiplying it by σ , and summing it to μ as shown in Figure 2.9. The new optimization objective for VAE is the Evidence Lower Bound, or ELBo. It is the negative sum of the reconstruction loss and the KL-divergence.

The constraints brought by KL-divergence are strong, which has led to multiple improvements to counterbalance some unfavorable effects. The largest effect is posterior collapse, which removes all transmission of information from the encoder to the decoder. This is characterized by the KL-divergence coefficient being too strong during optimization and ends up being minimized down to zero. A controllable hyperparameter β was introduced in β -VAE [Higgins et al., 2017] to change the importance of the KL-divergence. This parameter can be used to either reduce or increase the rate of transmission of the information at the expense of increasing or reducing the distortion of the output [Alemi et al., 2018].

Another way to constrain the variable is to use Wasserstein autoencoders [Tolstikhin et al., 2017]. This architecture is presented in the third column of Figure 2.9.

This way of constraining the latent variable is close to how Generative Adversarial Networks, or GANs [Goodfellow et al., 2014], operate. A discriminant network is added to the architecture with the role of discriminating the latent variable from vectors sampled from a multivariate normal distribution. This forces the latent variable to follow the normal distribution. However, this is supposed to be less degrading to the transmission of information through the latent distribution than the original KL-divergence term.

2.4.2 . Generative Adversarial Networks

Generative adversarial networks [Goodfellow et al., 2014] (GANs) are neural network models trained to generate target data from sampled noise. For NLP, the target data is text, but this architecture of models has been used with other types of data, such as images [Zhang et al., 2017a]. This type of model uses a discriminant model trained jointly to estimate whether the sentence is synthetic or natural. The interest in GANs has mostly been based on the idea that the adversarial method of learning would solve issues of traditional learning that created models that would generate uninteresting and repetitive sentences. While GANs have been effective with other types of data, the discrete nature of text and tokens has caused some issues. Backpropagating through discrete inputs is still an ongoing problem. One of the solutions to solve this problem for text generation is to use reinforcement learning. Policy gradient learning [Sutton et al., 1999] was the first reinforcement learning method used to train the generator. Examples with a causal LSTM model such as SeqGAN [Yu et al., 2017] or sequence to sequence models [Li et al., 2017] have been developed using this method. These methods, however, suffered from mode collapse, a problem where the generator ignores the input vector. TextGAN [Zhang et al., 2017b] seeks to alleviate this problem by having the discriminant learn the appropriate features for natural/synthetic classification and reconstruction of the vector input of the generator. The generator aims to match the relevant features. GANs are a promising technique, but the issues raised with text and the randomness of the approach are not suitable for our goal.

2.4.3 . Conclusion

While the techniques described in this section fit the context of this work, none of them will be featured in the experiments. Multiple reasons motivate this choice. A series of preliminary experiments have been conducted with autoencoders, as they are simpler to work with than GANs, which have not yielded positive results. While the reproduction capacities of the models were correct, generating new examples via sampling the latent variable proved to be difficult. Generating labels did not go far either. As for GANs, the difficulties we experienced with autoencoders deterred us from pressing in this direction, as GANs are difficult to train with text.

2.5 . Augmentation

Augmentation is a paradigm of techniques used to create new data to improve training. Data augmentation techniques aim to create synthetic data from available data. The goals of augmentation are varied, as it ranges from better generalization through debiasing the data to synthesizing the majority of the training set. Depending on those goals, the techniques used may vary. Our goal is to generate a new dataset. As such, we want to create new relevant information with the original data and the augmentation method. Techniques such as token corruption or permutation, which are handy for training more robust models, go against this goal of creating relevant information. They are not discussed in this thesis.

Training a NER tagger requires a large quantity of labeled data. In natural language processing, data augmentation has been used to improve performance. Techniques such as back-translation [Sennrich et al., 2016a] or Easy Data Augmentation [Wei and Zou, 2019] have been used in sentence classification. Paraphrasing using BART as a generator [Dopierre et al., 2021] is also a method that has been successful on the classification task. However, paraphrasing using back-translation on medical data for a tagging task is not effective [Neuraz et al., 2018]. The span-replacement techniques used can be associated with this family of tasks which aims at rewriting sentences. In our case, there are fewer constraints on the outputs, as it is not required to strictly be a paraphrase.

A method using language models to augment data, DAGA [Ding et al., 2020], has proven successful both in a supervised and semi-supervised learning setting. This method uses the training data to train a BiLSTM language model to generate either labeled or unlabeled data. Language models are used to generate new unlabeled data necessary for semi-supervised learning. Our experiments don't use BiLSTMs to generate new sentences but the more widely available GPT-2 [Radford et al., 2019] and T5 [Raffel et al., 2020] models. These models are used in their 1B parameters version. This data will serve to train our taggers, which use pre-trained BERT [Devlin et al., 2018] models as the basis of their architecture.

2.6 . Conclusion

Producing synthetic text requires a combination of techniques. Raw text cannot be processed by the statistical tools used. The techniques used to segment and transform sentences into sequences of vectors were presented. Neural networks can then use these representations to produce probability distributions that can be used to generate new texts. These language models have shown promising results on downstream tasks through various methods of application, such as model fine-tuning or prompt tuning. These techniques can also be used to change the generative properties of the models. The recent surge in the capacity of these models comes from two factors, the transformer architecture and the datasets used for pre-training. Pre-training these models requires a large quantity of data, namely multiple gigabytes of text, and fine-tuning such a model to orient the generation towards a specific domain requires thousands of sentences. However, our goal is to be able to produce an in-domain training corpus for tagging

tasks while having access only to a small quantity of target data. The constraint of the small quantity of data removes the possibility of fine-tuning a large model for text generation. It also removes the possibility of training a model from scratch. We have to rely on the capacity of the available pre-trained models to generate in-domain sentences in a few-shot manner. For the generation tasks, T5, GPT-2, and BLOOM are used in their versions close to 1B parameters as they have multiple versions ranging from 100M parameters to 176B parameters.

This chapter also presented models used on the downstream task. The models used in this thesis are BERT, BioBERT[[Lee et al., 2020](#)], and mBERT, following the BERT protocol for sequence labeling. This chapter introduced the techniques used at multiple levels of our experiments. In the next chapter, the methodology of how our experiments were conducted is described.

3 - Methodology

Research is a domain in which we try to highlight and understand phenomena. Experiments are designed with the purpose of producing data that should give insights into the studied phenomena. The validity of these results depends on the methodology with which the experiments were conducted. Prodding around might be useful, and some intuitions might be found this way, but justifying these results, replicating them, and understanding the deeper principles and limits will not be possible without a clear methodology. This thesis presents different methods of generation of labeled text for sequence labeling model training. To be able to evaluate these different generation methods, the setting in which these methods are applied must first be understood. The purpose of these generation methods is to be used in a low-resource setting. As such, being able to quantify the resources used is key to understanding the limitations of our methods. To be able to analyze the results of our experiments, a solid evaluation framework is needed. The metrics used to quantify the quality of the synthetic data and the performances of the trained models are explored. With these pieces of information, we have built a protocol of experimentation. These experiments must be conducted on some corpora that will preferably have previous results computed on them to have a point of comparison.

3.1 . Setting

The goal of our research is to explore generation methods that would improve the performances of a model trained in a low-resource setting. What setting can be considered low-resource is a question that must be answered. This section explores how the different quantities and qualities of data available affect the variety of techniques that can be used. We place the work presented in this manuscript in the low-resource topology presented in this section.

3.1.1 . Resource topology

Different levels of resources allow different machine-learning techniques to be used. This subsection explores these scales of data to be able to place our work in the resource topology.

The first divide of the resource landscape is the language of the corpus. Not all are equal in the face of data acquisition. English is by far the language with the most available data. Other languages with large corpora, such as Spanish, Mandarin, or French, do not compare in terms of data availability. A classification of the language divide found in Figure 3.1 in six stages was proposed [Joshi et al., 2020] and took into account the availability of labeled and unlabeled data. In this study, of the roughly 2500 languages studied, 88% did not have any unlabeled or labeled data. Training large language models to use in a zero-shot setting or as bases for transfer learning is not possible in this case. Not all techniques require a set of labeled or unlabeled data. For example, pattern-filling generation, a gener-

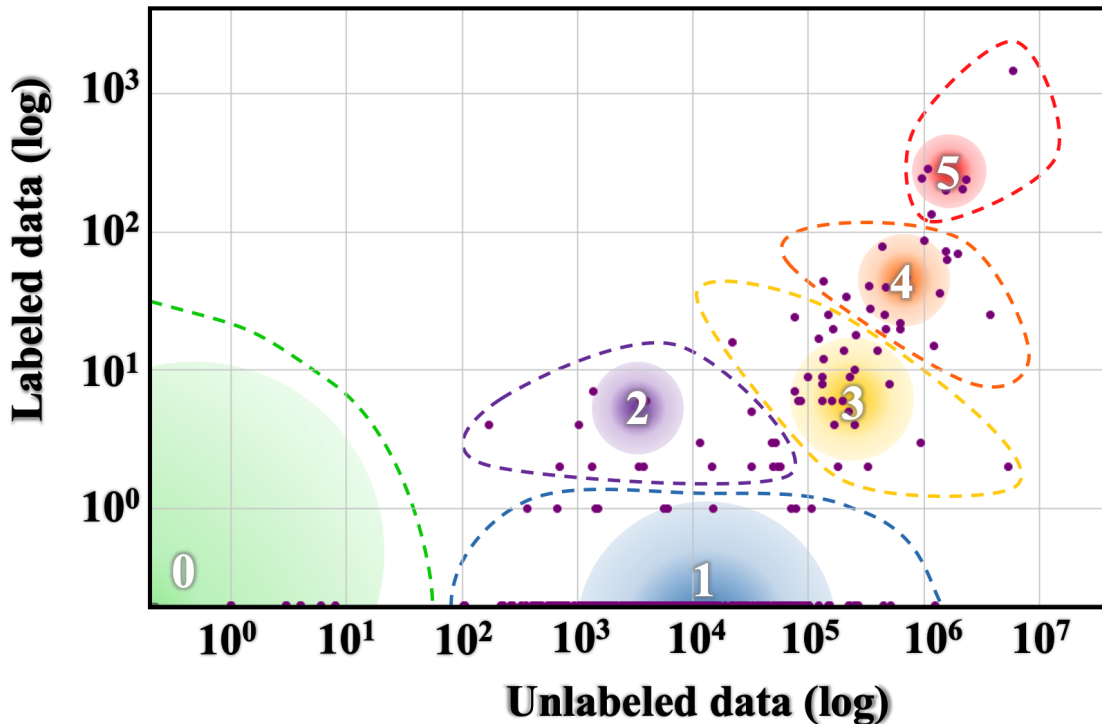


Figure 3.1: Taxonomy of languages through the available data [Joshi et al., 2020].

ation method presented in the next chapter, does not require such resources, but it requires a list of patterns and lists of mentions, which would be harder to come by if nothing is available. For this thesis, we place ourselves in a setting where language resources are available. This is motivated by the AIDA project being in French, which is a well-represented language (category five in Figure 3.1). Experiments are also proposed in English to have results comparable to a larger set of previous works.

Another axis of low-resource is the availability of domain-specific data. This type of resource allows training or fine-tuning large language models to be specialized in the domain. These models have better performances on downstream tasks than models trained on general-purpose data. Our results on I2B2 [Uzuner et al., 2011] follow this trend. In the following experiments, some of our taggers use specialized models, such as BioBERT [Lee et al., 2020] on the I2B2, which is a BERT-like model trained with additional PubMed data.

The last axis on which a low-resource setting can be set is the task-related low-resource setting. This axis is related to the downstream tasks, such as sentence classification or labeling. The most restrictive setting is the zero-shot setting. Solving a task in a zero-shot fashion implies solving the task without having data on this task. A popular approach to solving tasks in a zero-shot setting is using large language models and reframing the task as a language modeling task [Brown et al., 2020]. The second setting, while less restrictive, is the few-shot setting, where a small quantity of task-related data is available. This data isn't always labeled, but we assume that if the quantity of data is small enough, the data can be annotated for a small enough cost. Transfer learning and semi-supervised learning are meth-

ods used in this thesis to tackle this setting for the sequence labeling task. Our setting is the few-shot setting, as we have a small number of labeled sentences, and we will use available large language models to help with both generation and sequence labeling.

3.1.2 . Target setting

As stated in the previous subsection, this thesis aims to solve sequence labeling in a few-shot setting. There is no benchmark for the sequence labeling task in a few-shot setting. Evaluation is done using existing corpora that are transformed to respect the few-shot setting to obtain a good evaluation of our methods on the sequence labeling task. To that end, the methods tested are evaluated across multiple quantities of data. This allows having a chart of the behavior of the method at multiple quantities of data. The low-resource setting is a setting that can apply to multiple situations.

Developing methods to improve model performances at small quantities of data will improve the applicability of machine learning to new tasks or domains where data is hard to come by, such as the medical domain or the financial or banking domain. In the case of AIDA, we work with confidential data and want the method to be applicable to other confidential data. Our method needs to be applied to data that can easily be annotated, which means a very small quantity of data.

3.2 . Evaluation methods

A key point of methodology is how to quantify the results. Different tasks require different evaluations, which will have limitations on what they can measure. A first instinct would be to evaluate the quality of the generated data and the quality of the labels produced. Evaluating open-ended text generation is an ongoing problem described in the first subsection. The purpose of our generation methods is to produce data that improves tagger performances. As such, our evaluation of text generation can be done through the evaluation of tagger performances. Evaluation of the taggers is performed on the labels produced by these models. The different scores and measures that exist for the sequence labeling task are explored in the last subsection. With the knowledge learned in this section, we will then be able to build a protocol that is used in our experiments.

3.2.1 . Evaluating generation

Open-ended text generation is a complex problem. Models can be taught to reproduce sentences or to continue a previous text. The training objective used forces the model to produce sentences as close as possible to what the training sentence is. Other sentences could be viable, but there is no automatic way to evaluate their relevance. The quality of the synthetic text can be evaluated following grammatical and syntactical correctness. Truthfulness [Oshikawa et al., 2020] is also a measure that could be explored in order to validate the text. These measures give partial information on the quality of the synthetic text. Perplexity, which computes the information needed by the model to predict the sentence, is

another good candidate. However, this indicator is as much of an indicator for the quality of the model evaluating the sentence as an indicator of the difficulty of the sentence.

These previous methods of evaluation work only on the text generated and not on the combination of text and labels. It is complicated to evaluate text generation, and our goal is not to generate grammatically or factually correct text. Labeled data is generated in order to improve tagger performances. The quality of the synthetic text to the human eye is not important for our goal. We have chosen to have an extrinsic evaluation of the quality of the data generated. The evaluation is done through the performances of a model trained on it.

The data used to train these models is classified based on the composition of the data. The data from corpora is named natural data as it comes from corpora that either used human inputs to a system or human-written data. Generated data is the data built using generation methods. Augmented data is the data that contains both natural data and data generated from this natural data. When it is clear that there are no experiments done using only generated data, the term generated data is also used to write about augmented data.

Our baseline models are trained on a natural subset of data S_n , which is the same subset used to generate data. Models are also trained on augmented data which is the union of the natural subset of data and the generated set of data $S_n \cup G_n$. This allows us to have an evaluation of what the generation method adds to the natural data (in some experiments, the models are trained only on the generated data G_n). We value this evaluation method because our goal is to have better performance on the task. Due to the lack of meaningful automatic evaluation for text generation and due to the purpose of our generation, we chose to evaluate the quality of our method on the sequence labeling performance of the end models trained on the augmented data.

3.2.2 . Sequence labeling

To evaluate sequence labeling, we must first understand what sequence labeling is. Our work focuses on the sparse sequence labeling task, which may take the form of named entity recognition or natural language understanding. As opposed to non-sparse sequence labeling, not all the tokens are labeled with concepts. This causes one of the labels, the empty or outside label, to be over-represented compared to the other labels. In practice, the format from CoNLL 2002 [Tjong Kim Sang, 2002] is used for our sequences. This is a format that uses external labels and can also be called the BIO format. This format uses three types of tags: beginning (B), inside (I), and outside (O), where the B and I tags are tied to concepts. An application of this tagging scheme is presented in Figure 3.2. In this example from CoNLL 2003, there is only one entity labeled with the ORG or organization concept. Sentences can contain multiple entities coming from different concepts. With this tagging scheme, entities cannot overlap.

3.2.3 . Evaluating sequence labeling

Measuring the performance of labeling is an easier task than evaluating generation. Generating a sentence with the same meaning but a different structure or

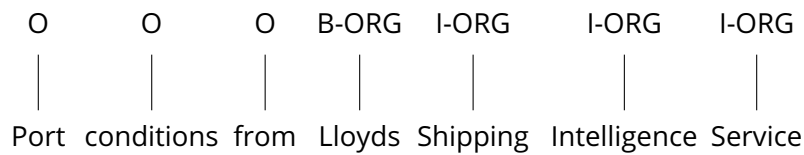


Figure 3.2: BIO format example from CoNLL 2003. This sentence contains one entity: [Lloyds Shipping Intelligence Service], which is an organization.

choice of terminology is hard to evaluate outside of human evaluation. This sort of problem does not exist for sequence labeling if labeled data is available. However, due to the sparse nature of our sequence labeling tasks, we cannot simply use accuracy or error rate. A model predicting only outside tags would have a high score with these measures. A more common measure of tagger performance is the F_1 score of the segment prediction. This measure is the harmonic average between precision and recall. Precision is the number of correctly predicted elements (true positives) over the number of predicted elements. It measures the confidence to have in the classification of found elements, in this case, mentions. The recall is the number of correctly predicted elements over the number of elements that are correct. It measures the capacity of the model to find and delimit what is a mention of the correct concept. For sequence labeling, the F_1 score is computed over the mentions. A mention is considered correctly labeled if the tags are exactly the same as in the reference. For example, in Figure 3.2, “Lloyds Shipping Intelligence Service” is labeled as an organization. If only “Lloyds Shipping Intelligence” is labeled as an organization, this mention counts as a false positive. If “from Lloyds Shipping Intelligence Service” is labeled as an organization, this mention also counts as a false positive. For sequence labeling, this measure is computed for each concept. The final measure obtained is the micro-average of the F_1 score of each concept. Micro-averaging takes into account the number of inputs of each concept. True positives are the exact matches between a ground truth mention and the predicted mention.

$$F_1 = 2 \frac{\textit{precision} \cdot \textit{recall}}{\textit{precision} + \textit{recall}} = \frac{2tp}{2tp + fp + fn}$$

Most results in this thesis were computed using seqeval [Nakayama, 2018]. The preliminary results of pattern-filling generation were computed using conlleval [Ramshaw and Marcus, 1995]. These methods output the same results on the same sets of data, which allows for a comparison of results.

Other possible measures exist. The concept error rate, or CER, is the ratio of falsely predicted concepts over the total number of concepts. These falsely predicted concepts considered for this measure are deleted, inserted, and confused concepts. CER provides information on the mentions that were wrongly tagged as opposed to the F_1 score. The sentence error rate, or SER, is the ratio of the wrongly tagged sequence over the number of sequences containing a concept. This information is interesting when considering how the tagging errors are spread across the sentences.

We have chosen to use the F_1 score as this measure was most widely used on the corpora we worked on, which made the comparison to other works easier.

3.3 . Protocol

The previous sections describe the target setting of our experiments and the evaluation metric used in this thesis. These bricks are the foundation of our experiments, but pitfalls still remain. To avoid these pitfalls, a protocol that is shared between most of our experiments was designed. Our generation methods are supposed to work in the target low-resource setting. A testing set is needed to evaluate our generation method, which is not something that you would have in a real low-resource setting. Our method needs to be comparable to other results. As such, our work must be on existing corpora on which the low-resource setting is simulated. We have settled on the F_1 score as the measure of performances of sequence labeling. We must outline what is to be compared with this measure in order to avoid meaningless experiments and incomparable results. At the end of this section, a generic protocol for generation method evaluation is outlined.

3.3.1 . Low-resource simulation

The target low-resource setting is a setting in which language-level and domain-level resources can be used when available, but specific task-related data is limited. The task-related data is considered to be small enough to be labeled manually. A means to simulate the low-resource setting needs to be included in the experimental protocol. In the real case of a new small-sized corpus, a test set would be of reduced size due to the constraints leading to a small corpus. Small test sets might cause problems for evaluation as they might not cover enough of the space to be relevant for evaluation. However, we are not trying to solve this problem. As such, we chose to keep the test sets as they are. This allows us to compare our results between the different sizes of the subsets that we work with and with other studies working in similar settings. Using the real-size test set also removes the question of whether the sampling or choice of the test set added any additional bias. It is easier to project the results obtained on the full-size test set than try to simulate the low resources.

In our publications [Boulanger, 2020, Boulanger et al., 2022], the full validation set was used. This had the advantage of making sure the training went through smoothly. However, it does not represent the exact situation of the few-shot setting simulated. As such, we have rerun the experiments in this thesis with a small validation set. This validation set is obtained with the same method the training set is obtained, namely sampling.

How do we simulate the few-shot setting for the training set? A first choice to simulate the few-shot setting would be to sample the data while conserving the frequencies with which the concepts appeared. While this would probably increase performances, our hypothesis is that it is unlikely that a small sample of data would keep the same properties as the whole set of data. We do not want to place ourselves in an ideal case. The data subsets are sampled without forcing any similarities in distribution from the entire dataset. Due to the natural ran-

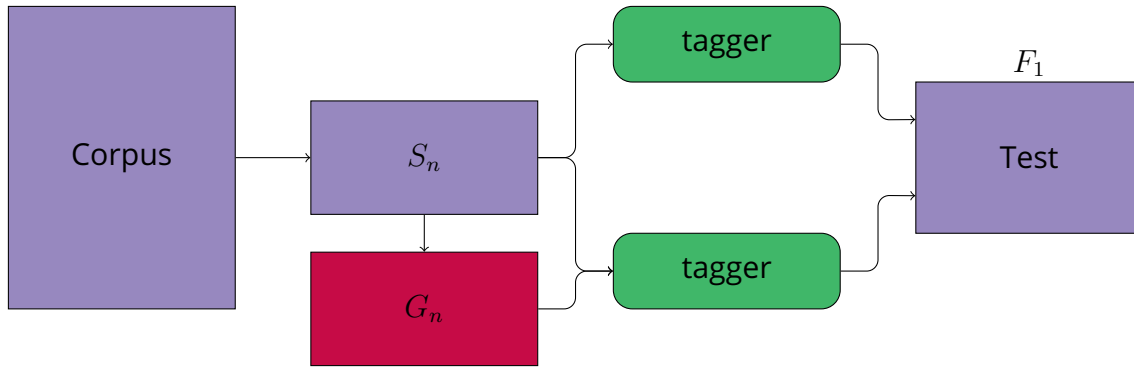


Figure 3.3: Protocol of evaluation. The corpus is sampled into a subset S_n of size n . S_n is used to train the baseline, which gets evaluated on the test set. S_n is also used to generate G_n and are together used to train another tagger which is also evaluated on the test set.

domness of this process, multiplying the experiments across seeds is required to compute the standard deviation and take into account the variations. One would expect that working in a low-resource setting would consume less computation due to the reduction of data size. But this reduction of training time for the model is largely compensated by the amounts of settings tested within each experiment. Seeding and sample sizes are the biggest factors in the multiplicity of the experiments. In some of the experiments, ten seeds are used, but in most experiments, only five seeds are used due to computational constraints. Uniform sampling is used to create our training subsets S_n and their associated validation sets. The training subsets are sampled from the training set, and the validation subsets are sampled from the validation set. While the validation set is not represented in the diagram, this sampling is represented as the first step from corpus to S_n in Figure 3.3. We have described the way the low resource setting is simulated, and we must now describe how generation is evaluated.

3.3.2 . Evaluation

We have settled on the use of F_1 as our measure of sequence labeling performance. How do we evaluate the impact of synthetic data? As stated in the previous section, the generation of data is evaluated through the performances of the taggers learned with the augmented data. This protocol is represented in Figure 3.3. Rather than a simple evaluation of the model, we employ comparative evaluation between models. The results of a model trained on the augmented set $S_n \cup G_n$ are compared to a model trained on the subset S_n . The performances are compared using the F_1 scores of the taggers on the test set. The implementation of the F_1 score for sequence labeling used is sequeval [Nakayama, 2018]. With this protocol, the performances are comparable between the baseline and the augmented training set. We can also compare the results obtained with the full-sized training set of the different corpora if we ever reach comparable results with a much lower quantity of data.

3.3.3 . Discussion

This method of evaluation does not give information on the general quality of the sentence generated. We do not have information on the impact of sentence quality on the performance of the models trained on the synthetic data. Generating sentences following grammar and syntax rules might improve performance. While this evaluation could be interesting, evaluating the linguistic rightness of sentences would require additional tools for each language used. Truthfulness is also an indicator that could be used to improve generation. This is especially true for tasks where the information extracted can have a critical impact.

Including mentions in the training set that should not be part of a concept might introduce problems in some cases. This problem might not appear with every generation method. For example, the pattern-filling generation method used has control over the mentions thanks to the mentions list, which is not the case for language models. Our hypothesis is that the entity-linking systems using the extractions that are produced by our models would be able to sort out mentions that are outside of the scope of the task as they would probably not link to any entity. These problems would require additional testing, which is out of the scope of this thesis. But we must also acknowledge these issues as our generation methods are not without any limits. The evaluation used does not evaluate the presence of this phenomenon unless it degrades the taggers' performances on the test set. Solving this issue might require specific knowledge of the domain and the task. We do not wish to bring individual adjustments to the methods presented as we want the methods to be applied to any sequence labeling corpora. Our first goal is to improve the performance of the tagger on the task through data generation, which is already vast.

3.4 . Corpora

Within the previous section, we have seen that the low-resource setting used is simulated on existing corpora. Our methods need to be applied to existing corpora to obtain a good evaluation. This section describes the various corpora used within our experiments. Various corpora are used in this project, spanning multiple sequence labeling tasks. The two major tasks worked on are named entity recognition, or NER, and natural language understanding, or NLU. First, the NER corpora used are described, then the NLU corpora. We will describe these corpora and why we have chosen to work on them.

3.4.1 . Named Entity Recognition

The named entity recognition task is a sequence labeling task that aims at extracting named entities in sentences. A named entity is a real-world instance of a concept. For example, "Martin Luther King Jr." is a person, "Rue Victor Hugo" is a location, and the "Pythagorean theorem" is a theorem. This task can be combined with entity-linking to assign an entity of a knowledge base to the instances extracted with NER. The first corpus of NER used is a classic NER corpus named CoNLL 2003. The second corpus is a clinical concept extraction corpus named I2B2. This task is essentially the same as NER but specialized for clinical entities.

Corpus	language	train	dev	test	tokens	vocab	concepts
CoNLL	English	14987	3466	3684	302811	30290	4
I2B2	English	11483	4832	27626	416790	28393	3
ATIS	English	4478	500	893	65364	950	83
MEDIA	French	12916	1259	3518	132256	2460	76
SNIPS	English	13284	500	700	132771	14348	40

Table 3.1: Corpora statistics. Train, dev, and test refer to the number of sequences in the train, dev, and test sets. The tokens column refers to the length of the corpus in the number of tokens. The vocab column refers to the number of unique tokens within the corpora. A cased and uncased version of the same word would count as 2 in the vocab column.

CoNLL 2003 English [Sang and De Meulder, 2003]

EU rejects German call to boycott British lamb .
Crohn 's is an inflammation of the bowel that can sometimes require surgery .

Rishon - Moshe Sabag (10th minute) , Nissan Kapeta (26th) ,
(Sweden) , Nick Faldo (Britain) , John Cook , Steve Jones , Phil

Table 3.2: Examples of sentences from the CoNLL corpus. At the top are some “regular sentences,” and at the bottom are some table-like sentences.

CoNLL 2003 English is a corpus of Reuters news in English. This corpus contains four concepts: persons, location, organizations, and miscellaneous. Reuter news contains a high quantity of tables and lists, which can be a problem for language modeling. These tables or lists are cut across lines, as can be shown in Table 3.2. The news comes from between August 1996 and August 1997 and is mostly focused on sports results, the economy, and politics. The distribution of the concepts of this corpus can be seen in Table 3.3. The corpus has overall similar quantities of organizations, persons, and locations. The miscellaneous concept is less represented, with around half of the other concepts’ entity count. This corpus is chosen as it is one of the most used corpora for the NER task. While this corpus is specialized in news, this domain is likely to contain overlap with a more general domain. This corpus is also chosen to be an example of a more general-focused corpus.

Type	ORG	MISC	PER	LOC
train	6321	3438	6600	7140
dev	1341	922	1842	1837
test	1661	702	1617	1668
Total	9323	5062	10059	10645

Table 3.3: Concept table of CoNLL.

I2B2 VA 2010 [Uzuner et al., 2011]

OTHER ENZYMES & BILIRUBINS Lipase
WHOLE BLOOD , MISCELLANEOUS CHEMISTRY Glucose Lactate Na K Cl
In general , the patient looked acutely and chronically ill .

Table 3.4: Examples of I2B2 sentences.

I2B2 VA 2010 is a corpus of medical records in English. This corpus contains three concepts: treatment, test, and problem. We can observe that treatments and test contain roughly the same amount of mentions, with around 14000 mentions overall in Figure 3.1, and the problem concept contain more mentions with 19664. The medical records contained in the dataset can have a messy case, a lot of abbreviated terms, acronyms, or specialized terms such as chemical names. These examples are highlighted in Table 3.4. A particularity of this corpus is the size of the test set. The test set contains more sentences than the train and dev set combined, as can be seen in Table 3.1. This corpus is used as it is a highly specialized corpus.

Type	treatment	problem	test
train	3360	4995	3252
dev	1481	2077	1354
test	9344	12592	9225
Total	14185	19664	13831

Table 3.5: Concept table of I2B2.

3.4.2 . Natural Language Understanding

Natural language understanding, or NLU, is a sequence labeling task aimed at extracting information from conversations. This task is usually used in task-oriented dialogue systems to understand what a user might want from the system. The NLU task is usually formed of a sequence labeling and a sequence classification task. The sequence labeling aspect of this task is the only aspect of the task treated. The NLU task is often linked to the information extraction part of a chatbot system. This sequence labeling task is highly similar to NER, with entities being labeled but with additional specificities linked to the task. Other types of token sequences can be labeled, which relate to information necessary for the chatbot system, such as dialog state information and quantities. Three NLU corpora are represented in this work: ATIS, MEDIA, and SNIPS. We will describe these corpora and the reasons we have chosen to use them.

ATIS [Dahl et al., 1994]

ATIS, or Air Travel Information System, is a corpus of plane travel booking. This corpus is a simple corpus as most of the remaining errors stem from labeling errors from the test set [Béchet and Raymond, 2019]. ATIS contains a high quantity

of concepts compared to the other corpora, as shown in Table 3.1. The vocabulary of the corpus is small, with less than a thousand tokens. This corpus is used as validation to see if there are any major problems with the methods when they are applied to it. This corpus is used in our publication on pattern generation [Boulanger, 2020]. We have not produced results outside of the preliminary experiments of pattern-filling generation with this corpus, but it will show up in discussions of the pattern generation method.

Type	toloc city_name	fromloc city_name	depart_date day_name	...	booking_class	flight
train	3919	3892	785	...	0	0
dev	424	434	104	...	0	0
test	716	704	212	...	1	1
Total	5059	5030	1101	...	1	1

Table 3.6: Concept table of ATIS. Due to size restrictions, only the three concepts with the highest quantity of mentions and the two lowest are displayed.

i want to fly from baltimore to dallas round trip
what does fare code y mean
show me the flights from atlanta to denver on friday

Table 3.7: Examples from the ATIS corpus

MEDIA [Bonneau-Maynard et al., 2005]

Type	reponse	command tache	nombre chambre	...	hotel etat	localisation codePostal
train	5926	2102	1761	...	1	1
dev	552	207	203	...	0	0
test	1627	554	450	...	0	0
Total	8105	2863	2414	...	1	1

Table 3.8: Concept table of MEDIA. Due to size restrictions, only the three concepts with the highest quantity of mentions and the two lowest are displayed.

MEDIA is a French corpus of hotel reservations made from telephonic transcriptions. This corpus is still quite complicated, with performances of recent models only reaching 90 F_1 score [Cattan et al., 2022]. The transcription of the corpus has kept many specificities of spoken language. The utterances often contain repetitions of words or sequences and even humming, as can be seen in Table 3.9. This corpus also contains a wide variety of sentence lengths. Most of the “reponse” concepts come from two one-word sentences, which are “oui” and “non.” Some

oui
 euh hum est-ce que vous pouvez vérifier la disponibilité à l' hôtel Novotel
 je souhaite deux chambres individuelles des chambres voisines deux étoiles

Table 3.9: Examples of sentences from the MEDIA corpus.

of the utterances reach multiple hundreds of tokens. We use this corpus in all the experiments designed for this thesis. This corpus brings French to these experiments, which has two aims, testing our method on an additional language and having an example of French, which is the language of our target task.

SNIPS [Coucke et al., 2018]

Type	object_type	object_name	playlist	...	poi	genre
train	2984	2936	2092	...	143	144
dev	201	0	0	...	0	0
test	156	151	109	...	6	3
Total	3341	3087	2201	...	149	147

Table 3.10: Concept table of SNIPS. Due to size restrictions, only the three concepts with the highest quantity of mentions and the two lowest are displayed.

SNIPS is a corpus of home assistant queries in English. The queries are spread between 7 classes, all containing around 2000 sentences. Some of the queries are very formulaic, such as the addition of music to a playlist, as can be seen in Table 3.11. The distribution of concepts is less disparate compared to other datasets. SNIPS contains a similar quantity of tokens as MEDIA, but its vocabulary is larger, with around 14000 unique tokens. We have used SNIPS for pattern generation as it is a corpus that is more complicated than ATIS on the NLU task, and it is also in English.

Add Tranquility to the Latin Pop Rising playlist.
 Book a table at a french restaurant for fern and I
 What is the weather like at Basaseachic Falls National Park

Table 3.11: Examples of sentences from the SNIPS corpus.

3.5 . Taggers

This section describes the architecture of taggers used throughout the experiments. The architecture of the taggers used for the set of experiments done to homogenize the results obtained throughout the three years of the thesis is described. We will not go into as much detail about the architectures of the models coming from the publications if they are different. The first architecture used is a

BiLSTM, but to be closer to state-of-the-art methods, the architecture used in most of our experiments is a different one. The tagger architecture now used is a BERT + MLP architecture. The results on the full-sized set can be found in Table 3.12 BERT is here used widely to represent any of the models sharing the architecture with BERT or derivatives of BERT. In specific, BERT-large-cased¹ [Devlin et al., 2018] for CoNLL 2003, BioBERT-cased² [Lee et al., 2020] for I2B2, and mBERT³ [Devlin et al., 2018] for MEDIA are used. The MLP is composed of two dense layers separated by a ReLU.

	CoNLL	I2B2	ATIS	MEDIA	SNIPS
Train sentence count	14,986	11,482	4478	12,916	13,284
Test sentence count	3,683	27,625	893	3518	700
BiLSTM	87.5	79.1	95.3	84.6	91.8
BERT base	90.0	84.0			
BERT large	92.0	85.0			
BioBERT		86.6			
mBERT base				88.2	
SOTA	94.6	90.3	99.0	90.0	98.6

Table 3.12: Reference models used as topline for our work and viability check against the current state of the art. F_1 of BERT + classifier models on CoNLL and I2B2 using different pre-trained models. Metrics computed by seqeval [Ramshaw and Marcus, 1995, Nakayama, 2018]. The best model based on development set F_1 , trained on 50 epochs, with a batch size of 32. The last line is the state-of-the-art at the time of writing based on papers with code⁴ when available. State-of-the-art (SOTA) for CoNLL [Wang et al., 2021a], I2B2 [Si et al., 2019], ATIS [Wang et al., 2018b], MEDIA [Cattan et al., 2022] and SNIPS [Chen et al., 2019a].

3.6 . Summary of the methodology

This chapter has described the method with which our experiments are conducted. We place ourselves in a low-resource setting akin to the few-shot setting because the AIDA project is set in such a setting. This setting is simulated with existing corpora by sampling training and validation subsets. The test set is kept intact for comparisons to other results. Our generation method is evaluated through a comparative evaluation of a tagger trained on the natural subset S_n and a tagger trained on the augmented subset $S_n \cup G_n$ as described in Figure 3.3. Finally, the corpora used throughout our experiments are described. It is now time to describe the experiments, beginning with pattern generation.

¹<https://huggingface.co/bert-large-cased>

²<https://huggingface.co/dmis-lab/biobert-base-cased-v1.1>

³<https://huggingface.co/bert-base-multilingual-cased>

⁴<https://paperswithcode.com/>

4 - Pattern Filling Generation

An overview of the generation landscape in neural-based NLP was presented. A methodology was set up that should allow us to have correct comparable results. We must now explore and propose generation methods.

The first generation method explored is the pattern-filling generation method. Using patterns to generate sentences is a simple and efficient method. Patterns have been used to generate sentences to train models used for automatic speech recognition [van Schooten et al., 2007]. Corpora created using a rule-based approach can be interesting to test the capacities of models such as the bAbI tasks [Weston et al., 2016] or the dialog state tracking tasks introduced in following works [Bordes et al., 2017]. Sequence labeling in the medical domain has been tackled using pattern-based generation [Neuraz et al., 2018].

As a result, this approach is the first considered and explored for the generation of data for sequence labeling. This type of method is cheap as it doesn't require machine learning to generate new examples. Machine learning is heavily reliant on data, and removing this potential bottleneck in the first method tried is not a luxury. Pattern-based generation allows control and consistency of what ends up in the generated data. This is also why methods from this family are used in commercial applications. Our goal is to obtain a strong baseline for generation and augmentation with this method.

Our pattern-filling generation method first needs to be introduced. A series of preliminary experiments were conducted to test and refine this method. In the first experiment, generated data is the sole source of data for tagger training. This experiment sets the base experimental settings alongside the first improvements to the generation methods. Analyzing how many patterns are necessary to produce a tagger that could be used in a bootstrap setting became an objective to better understand the applicability of the method in a real-case scenario. Obtaining patterns will most likely result in obtaining valid labeled sentences. In this case, augmentation would be a more data-savvy solution. These preliminary experiments mostly don't follow the methodology presented in chapter 3. These experiments have helped shape the methodology through their limitation, which includes the lack of multiple seeds or augmentation as the setting in which we place ourselves. A final round of experimentation using the methodology of chapter 3 was done to validate the claims of the preliminary experiments.

4.1 . Pattern filling generation

This section describes the pattern-filling generation algorithm used. Patterns must first be understood to understand the pattern-filling generation method. Patterns are sentences with empty slots associated with a concept or a class. An example of pattern would be : *<person>* is from *<location>*. A slot can be filled by a mention, which is an instance of a concept.

For the pattern-filling generation process, mentions are taken from lists associated with a class, as shown in Figure 4.1. This gives control over what the taggers can learn to label. Mention lists can be obtained from knowledge bases if they exist for the domain related to the task. Mention lists can also be created through extraction from labeled sentences, as access to sentences is not entirely out of the window since this thesis places itself in the low-resource setting.

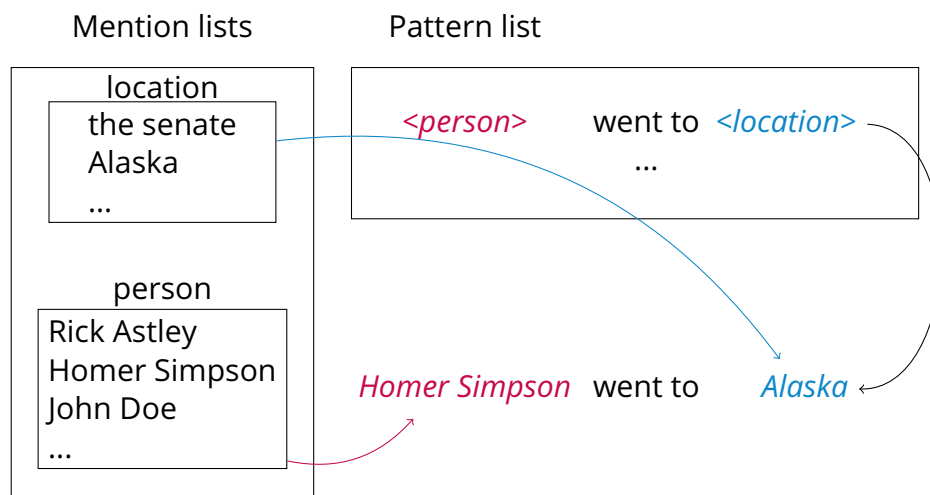


Figure 4.1: Example of pattern filling generation. This is a diagram showing how a pattern at the top is filled by taking mentions from lists of the appropriate concepts. The mentions are sampled uniformly.

Patterns can be extracted from labeled sentences or have to be written. Writing the patterns might cause problems because they might not fit the natural data. But it might be the first step to provide a system that would acquire natural data, which would, in turn, be labeled and used to create patterns. Extracting a pattern is done by following the process from Figure 4.1 in reverse. Given the labeled sentence : **Homer Simpson** went to **Alaska**. The mentions **Homer Simpson** and **Alaska** would be extracted and placed in their respective mention lists. The mentions would be replaced in the sentence by special tokens representing the concepts associated to produce the pattern.

Pattern-filling is the creation of sentences via the association of the patterns and the mentions, as can be seen in Figure 4.1. The mentions are used to fill the slots in the patterns. Because of the slots and their associated concepts, the sequence of tags can be reconstructed by assigning empty tags to the non-slot parts of the pattern and concept-related tags to the mentions inserted.

The number of sentences that can be generated explodes as illustrated in the following equation where M_{slot} is the number of mentions in the list associated with the slot.

$$\sum_{p \in \text{pattern}} \prod_{slot \in p} M_{slot}$$

A pattern can produce a number of sentences equal to the product across slots of the number of possible mentions in the slot. This number quickly explodes when summed across the available patterns. The mentions and patterns used

need to be chosen. Because information on possible existing combinations of patterns and mentions is supposedly unavailable, we have chosen to sample the mentions. The initial approach is to sample uniformly from lists of mentions that contain unique mentions. This point will evolve throughout the experiments.

To be able to apply the pattern-filling generation method, access to patterns and mentions is needed. As stated in the methodology chapter, existing corpora are used on which our method is applied. From these corpora, a subset S_n is sampled. And from this subset S_n , the mentions and pattern are extracted.

4.2 . Generated data

The objective of this first experiment is to propose a first evaluation of the pattern-filling generation method. As part of the preliminary experiments surrounding the pattern-filling generation method, the setting and methods are different in some aspects from those presented in chapter 3. They are described in the first subsection. The results obtained in these conditions are then observed and analyzed. The behavior of the approach on some corpora raises questions that require further investigation of the data, both natural and generated, to propose solutions. Finally, the proposed correctives are applied, and the approach is evaluated.

4.2.1 . Description

This section describes the initial pattern-filling generation experimental setting and protocol applied in the preliminary experiments. Both the setting and protocol will evolve to question different aspects of the generation methods. This section is the basis on which the following preliminary experiments are built.

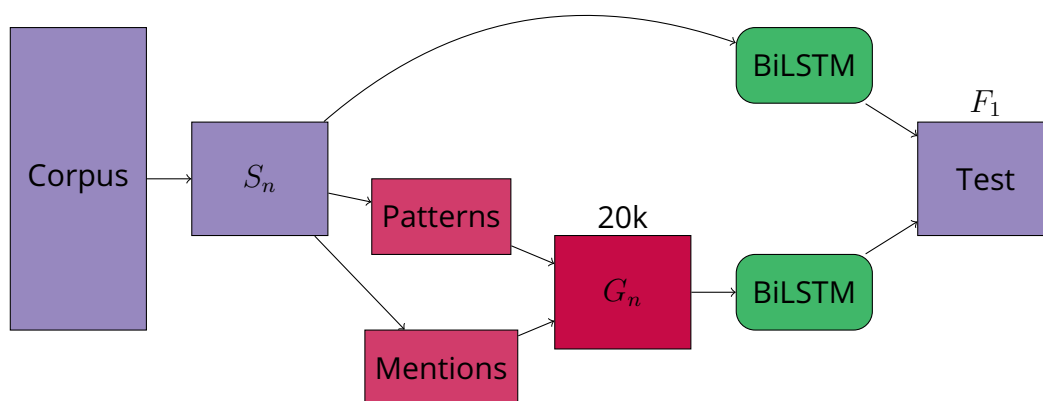


Figure 4.2: Overview of the protocol for the generated data experiment. The S_n data subset is sampled from the training corpus. Patterns and mentions are extracted from this subset and used to generate G_n of size 20k. Both S_n and G_n are used to train BiLSTMs that end up being evaluated on the Test set from the corpus.

A compelling setting to use pattern-based generation is a setting where knowledge bases and patterns are accessible. In this setting, the generated method produces a generated set and not an augmented set. The choice made was to

sample sentences from the training set of the corpora to extract the mentions and patterns from. The problem with this method is that it possibly doesn't represent the way data will be distributed in the pattern and mention lists in a real setting. This problem can, however, not be answered, as it is relative to the task, the domain, the project, or the organization you are part of. The benefit of this choice is to have a clear set of data that can be used for comparative evaluation. The generated data can be compared to the natural data it was generated from. In this setting, the generated training set can be of almost any size. The size of this set was chosen to be higher than the largest training set of the corpora used but in a similar size bracket. On the sequence labeling task on a specific dataset, the more data available, the better the tagger performs. A single number of generated data was evaluated whatever the size of the natural subset used to produce it. This allows comparison between these different sizes of initial data used.

With the setting now described, the protocol of the experiment is as follows. Our patterns and mentions are obtained by extracting them from the subset S_n of data, as can be seen in Figure 4.2. For this experiment, the mentions and patterns are extracted without duplicates. This means that the mentions and patterns are uniformly distributed, with only a single occurrence of each within the lists. This detail will be useful for understanding the differences between this experiment and future experiments. The size n of the subset S_n sampled from the corpora varies between 100 sentences and the full size of the corpus. The subsets S_n follow this rule : $S_{i < j} \in S_j$. This is done by shuffling the training set once and taking the n first sentences for each n . This experiment was done using a single seed as a result of former blind spots in methodology. Twenty thousand utterances were generated with the pattern generation method, regardless of the size of the original subset. The corpora on which the experiment was applied are ATIS [Dahl et al., 1994], I2B2 [Uzuner et al., 2011], MEDIA [Bonneau-Maynard et al., 2005], and SNIPS [Coucke et al., 2018]. For this experiment, the tagger used is a BiLSTM [Hochreiter and Schmidhuber, 1997]. Using a BiLSTM allows us to plug in pre-trained embeddings. The embeddings used were pre-trained on Wikipedia using the CBOW method [Ghannay et al., 2020]. The development set used for training was the full-sized development set. Evaluation of the pattern-filling generation method was done by comparing the performances on the test set of the BiLSTMs trained on the natural data S_n and the generated data G_n .

4.2.2 . Results

This section presents the results obtained with this first experiment. The F_1 scores of the model of this first experiment are presented in Figure 4.3. The results are mostly positive for I2B2, ATIS, and SNIPS, where the F_1 score of the models trained on the generated data is higher than on natural data for lower-sized subsets, and this tendency flips for SNIPS and I2B2 on the largest sizes. For I2B2, at size 100, the taggers trained on the generated set improve their performance by 6.5 points and then stay within 2.5 points of the natural results, whether it is for improvements or degradation, with mostly positive results. For ATIS, the taggers are improved at every size step, from a 3.9 improvement at size 100 to a 0.5 improvement at full size. For SNIPS, the results are impressive as pattern generation

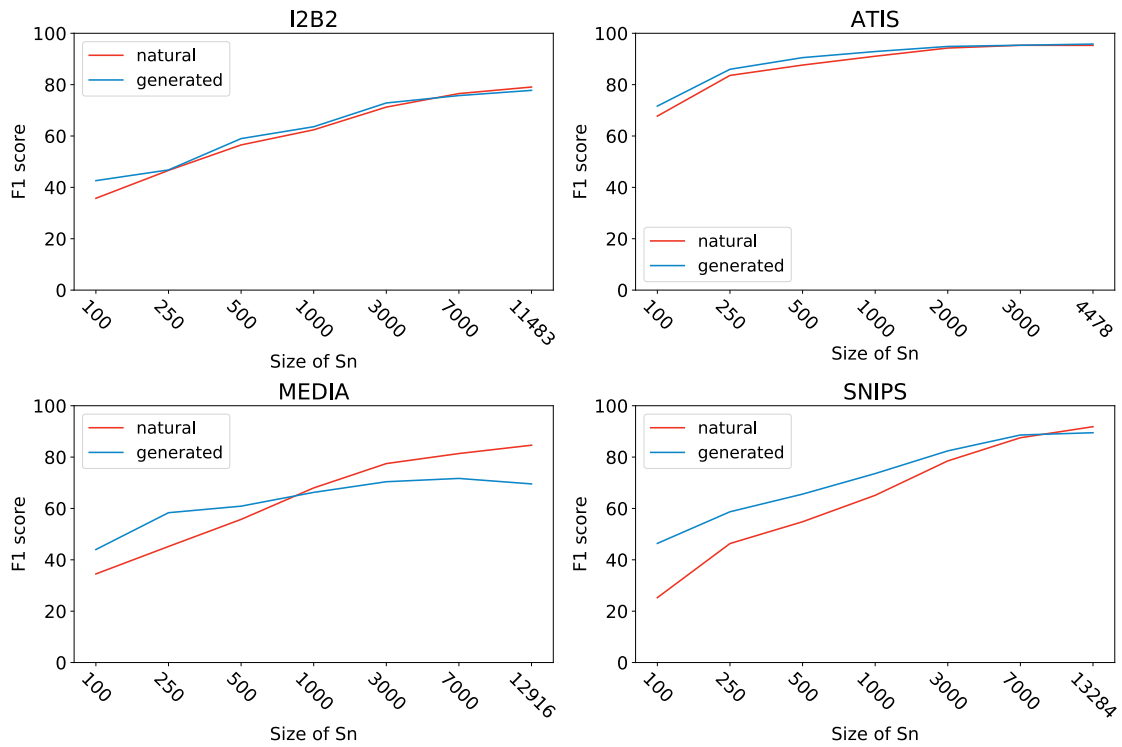


Figure 4.3: F_1 score of natural and generated data with BiLSTM taggers.

allows to train model at size stamp n with better performances than the baseline at size stamp $n + 1$ for $n \in \llbracket 100 ; 500 \rrbracket$. The performances are increased over the baseline up to the full size. The case of MEDIA is much more dramatic, with high increases in performance (+9.5 points) at size 100 but high degradation of performances (-15 points) at full size. The first observation of these results shows an overall tendency to have better performances with low quantities of data and lower performances with higher quantities of data. Observation shows that ATIS is the most simple task by the way its natural data curve behaves. It starts very high, even with the smallest amount of data, which is 100 labeled sentences, reaches over 80 F_1 score at 250 sentences, and finally reaches a plateau at around 95 F_1 score for 2000+ sentences. The results from MEDIA are also concerning. Degradation at high quantities of data can be expected, but this degradation appears with larger quantities of natural data than for MEDIA. Investigation of the data needs to be done in order to understand what causes the problem with MEDIA.

4.2.3 . Data Analysis

This section describes the analysis performed on the natural and generated data to understand the problems raised previously and devise ways to counter these problems. The initial results of pattern generation using generated data are mixed. For multiple corpora such as I2B2, ATIS, and SNIPS, the results are mostly positive, but for MEDIA, the results were positive on small quantities of data but highly negative on higher quantities of data.

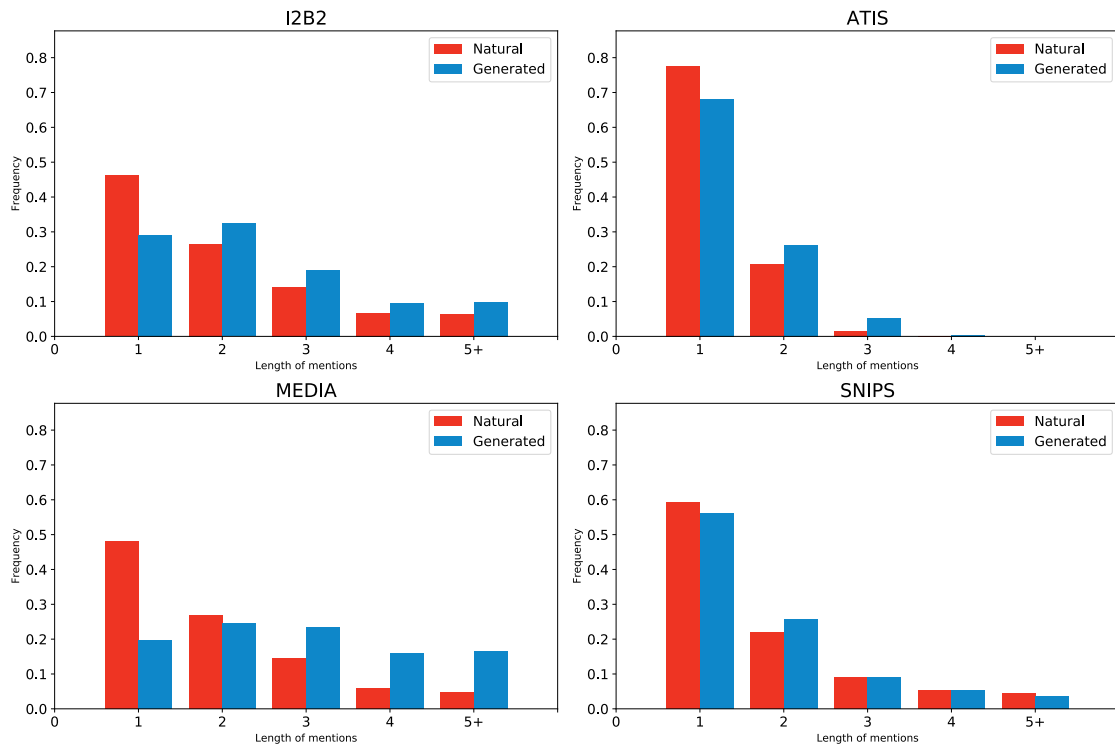


Figure 4.4: Distribution of the length of mentions in the natural and generated data. Each histogram represents the percentage of mentions in the category. The categories are the sizes of the mentions in the number of tokens from one to four, and a category combining the mentions of five tokens and above.

First, finding indicators that could be representative of issues must be done in order to be able to pinpoint the issue and then correct it. Because the results were mostly worse at higher quantities of data, the data is analyzed at full size.

One of the things that generation has changed compared to the natural data is how the mentions and patterns are distributed. Due to the unicity of the mentions and patterns in the lists and due to the uniform sampling, the mentions and patterns are uniformly distributed with regard to their concepts in the data. However, the natural distribution is different because duplicate sentences and mentions exist in the natural corpora. Figuring out the difference in the individual distribution for mentions and patterns is not that interesting because this knowledge cannot be transferred to a real-case scenario where the mentions are extracted from a knowledge base, and patterns are written by hand. Indicators that could be used in these scenarios are needed to inform users of the generation method on how to proceed to improve their generated data.

Our first hypothesis is that mention length plays a role in the difficulty of the labeling task because it might become harder for a model to delimit the frontier of a longer mention. Longer mentions could contain parts that would be related to another concept, such as the name of a person in a street name, which is a location. The distribution of mention length will be impacted by the change in mention distribution. Figuring out a trend in how the mentions are distributed following their length in a natural setting will help try to replicate this distribution in a real-case

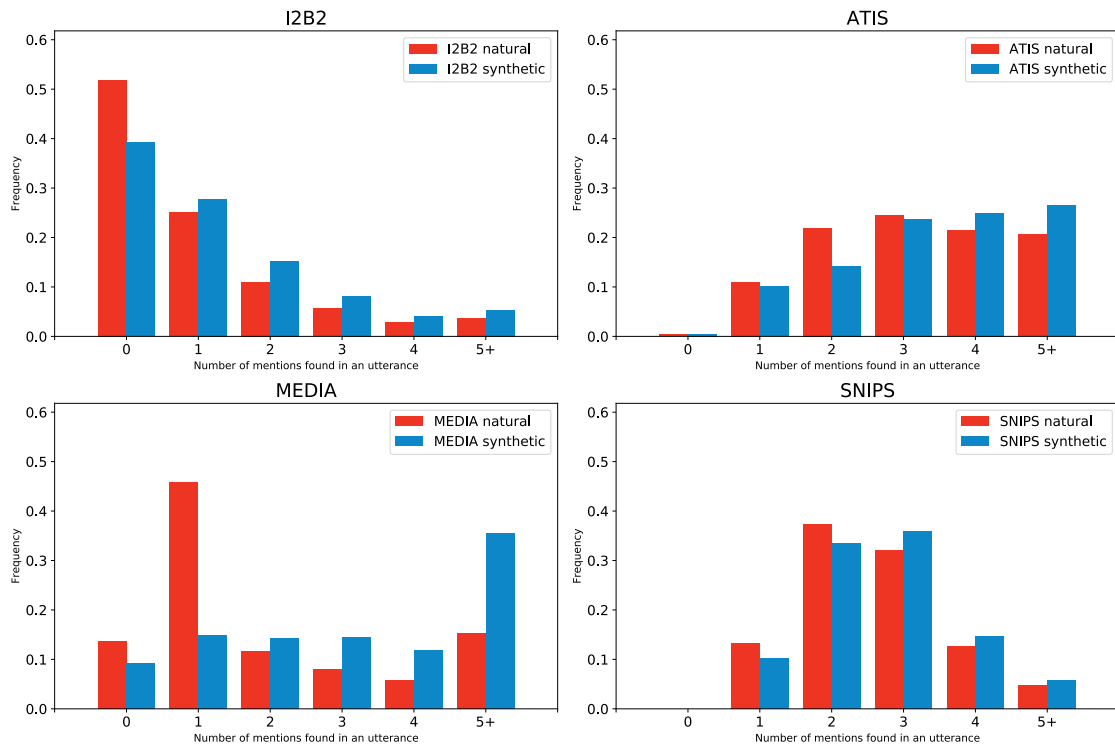


Figure 4.5: Distribution of the number of mentions per utterances in natural and generated data. Each histogram represents the percentage of utterances in the category. The categories are the number of mentions in the utterance from zero to four and a category combining the utterances containing five mentions and above.

setting. The results of this analysis can be found in Figure 4.4. Natural data show a Zipf-like distribution in the mention length on the five datasets. The longer the mentions, the less they are represented. This distribution shifts towards longer mentions in the generated data. That shows that short mentions exist with more multiplicity in the natural setting than longer mentions. ATIS and SNIPS have minimal changes in the distribution. I2B2 changes are greater, with close to 15% of the mentions of length one being shaved off and distributed across the other length. The most represented length of I2B2 shifted from 1 to 2. This did not seem to affect the pattern generation method negatively. The distribution shift on the MEDIA corpus was the strongest shift. The Zipf-like distribution turned into a very flat Gaussian distribution centered between 2 and 3-word mentions if you consider the 5+ mentions to spread like a classic tail.

Exploring the impact of pattern distribution is important as patterns dictate the concepts present in the generated sentences. Because of that, pattern distribution controls concept distribution. Our hypothesis is that controlling pattern distribution might have a high impact on the performances of taggers due to the linked concept distribution. Patterns are inherently linked to concepts as the number of slots in a pattern represents the number of mentions in the utterance or sentence made from the pattern. We have chosen to look at the distribution of mentions across sentences. This quantity gives us information on how the change in pattern distribution has impacted the landscape of mentions and, by extension,

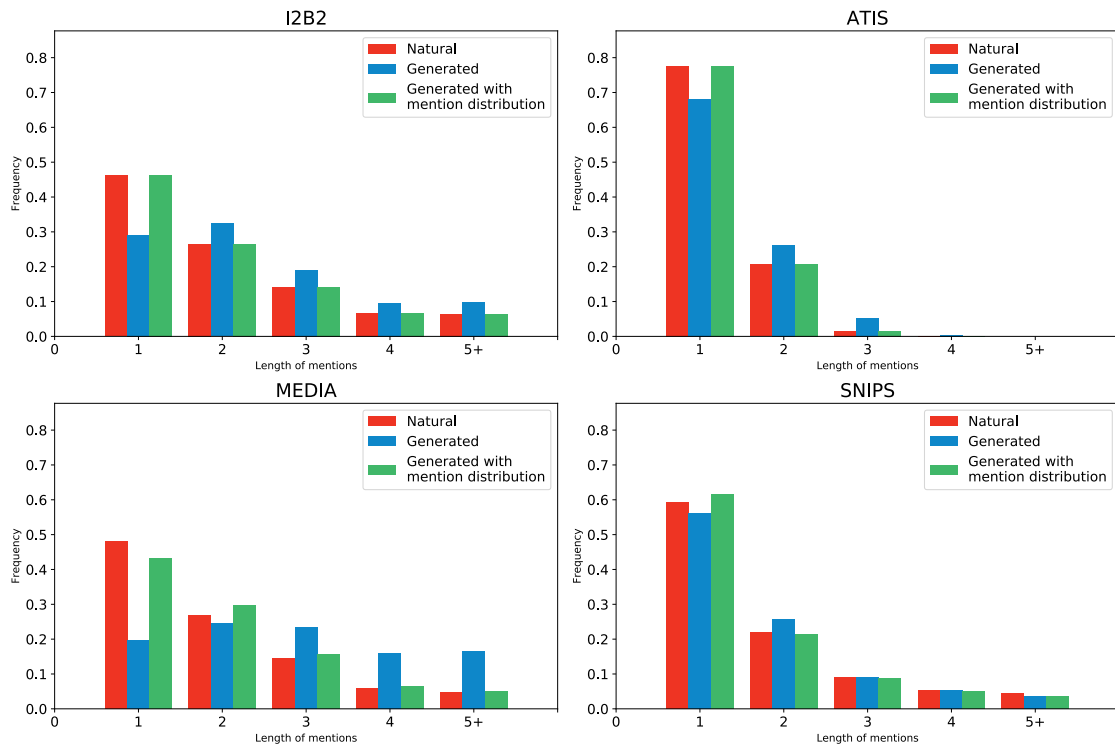


Figure 4.6: Distribution of the length of mentions in the natural and generated data. Each histogram represents the percentage of mentions in the category. The categories are the sizes of the mentions in the number of tokens from one to four, and a category combining the mentions of five tokens and above.

concepts. The distributions of mentions per utterances across corpora can be found in Figure 4.5. The way this quantity is distributed does not follow a general rule like the length of mentions. The corpora can nevertheless be divided into two categories: containing empty sentences or not. Empty refers to the absence of a mention in the sentence. SNIPS does not contain any empty sentences, and ATIS contains almost zero empty sentences. The other corpora contain a significant amount of empty sentences. The distribution of the number of mentions per utterance in I2B2 follows a Zipf-like distribution. For ATIS, the distribution is close to a Gaussian distribution centered on three mentions per utterance. For SNIPS, the distribution is close to a Gaussian distribution centered a little higher than two mentions per utterance. Finally, for MEDIA, there is a spike of sentences with one mention and a long tail. The generated data shifts the distributions towards more mentions per sentence. The effects are not that important except for MEDIA, where the spike gets flatted, and the quantity of sentences containing more than five mentions explodes. The reduction of the 1-mention per sentence spike can almost fully be explained by a pattern containing only the concept “reponse.” Around 5000 sentences of “oui” or “non” are part of this spike in the natural distribution.

The effects of pattern-filling generation on the distributions have been shown. How can it be solved? As the data used doesn't come from knowledge bases, the natural distribution of mentions and patterns is available. Applying these distribu-

tions requires adding small changes to the extraction pipeline. The pattern-filling generation follows the same behavior. The changes are the pattern list and the mention lists fed to the algorithm. Options were added to the extraction pipeline to either add mention with multiplicity or add patterns with multiplicity to the respective lists. This can be applied if data is available initially but not in the knowledge base scenario. If it doesn't work with this setting, our hypothesis is that manual changes to the distribution won't solve the problem either. While there can still be a difference in mention length when applying mention distribution due to the shift in pattern distribution as slots are linked to the patterns, the number of mentions per utterance should be exactly the same. Informing pattern generation with the mention distribution generally solves the distributional shift, as can be seen in Figure 4.6. Some small differences can still be found due to the way pattern distribution might change the concept distributions. MEDIA is highly impacted by uniform distributions. Using the natural distribution of the mentions and then the patterns correct these distributional divergences can be seen in both Figure 4.6 and Figure 4.5. Let us see what the results of the taggers trained on the corrected generated data are.

4.2.4 . Following natural distributions

This section presents the results of changing the distributions in the pattern-filling generation method to the natural distributions. Two sets of experiments have been done. The first one preserves the natural distribution of the mentions in the generated data. The second one preserves the natural distribution of the patterns in the generated data. Outside of these changes, the protocol remains the same as the one presented in subsection 4.2.1.

The results obtained can be observed in Figure 4.7. Precise scores are available in Appendix B. For I2B2, at size 100, natural pattern and mention distribution bound the uniform score and are respectively 7.3 and 6.3 points ahead of the natural data. At size 250, the new methods perform similarly and are better than the uniform method, with a 47.8 F_1 score against 46.8. From sizes 500 to 3000, the natural mention and pattern distributions are bounding the uniform distribution with patterns being worse. For 7000+ sentences, the new methods are better than the simple pattern-filling generation method, as they are better than the taggers trained on natural data at 7000, but still obtain lower performances than natural data at full size. For ATIS, both new methods obtain positive results at every size step. Both methods obtain 2.8 points more than the baseline for size 100 but are still 1.1 points below the previous pattern-filling generation results. Using the natural distribution of patterns obtains the overall best results with 95.8 points at size 3000 which is equivalent to the results of the previous experiments at full size. For MEDIA, the results have improved at large numbers of sentences when following the distribution of mentions. With natural mention distribution, the degradation has gone from -15 points at full size to -1.9 points. The gap is further increased by another point when using the natural pattern distribution. Both methods improve results at size 100, with mentions obtaining better results with a 14.5 increase from the baseline. Both methods obtain better results than the baseline up to size 1000 and obtain better scores than the uniform distribu-

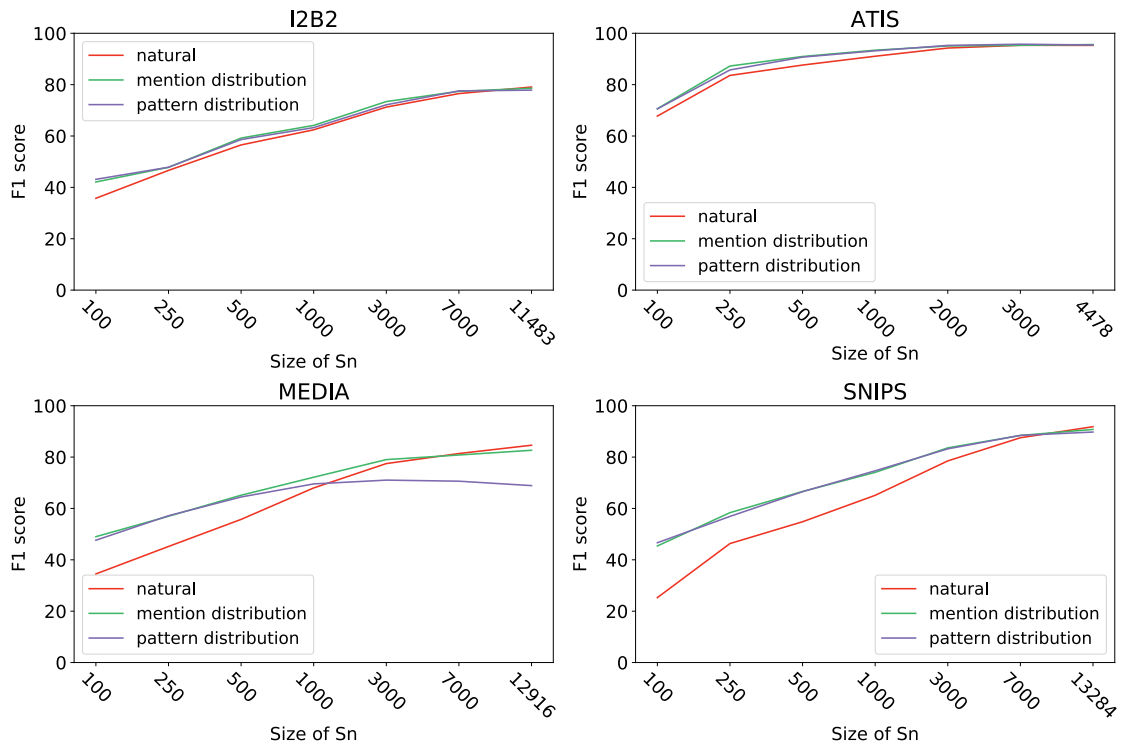


Figure 4.7: F_1 score of natural and generated data with BiLSTM taggers. The generated data either preserves natural mention distribution or natural pattern distribution.

tions from 500 onward for mention distribution and from 500 to 3000 for pattern distribution. For SNIPS, both methods are roughly equivalent and obtain better results than the uniform method from 500 sentences onwards.

Our hypothesis that controlling pattern distribution might have a high impact on the performances of taggers does not seem to hold as following the natural pattern distribution performs worse than following the mention distribution. The performances of the taggers trained with pattern distribution on MEDIA are similar to taggers trained with uniform distribution. However, following the natural mention distribution seems to sort out the problem.

4.2.5 . Conclusion

We tried to use pattern-filling generation to create generated training sets for our taggers. This method has been successful for the majority of the corpora using a uniform distribution of mentions and patterns. A problem was found for the MEDIA corpus where a high degradation of performances appeared at high quantities of data. This problem has pushed us to look deeper into the corpora and understand that the distribution of the number of tokens in a mention played a key role in the difficulty of the corpus. Pattern distribution was expected to play a key part, but the performances of the taggers trained on data with natural pattern distribution did not show any significant improvement, especially on MEDIA. The distribution of data has an impact on the model performances. Mentions are likely more available than patterns through the use of knowledge bases. Obtaining patterns is done via writing them or labeling sentences. These processes are

difficult and tedious. Figuring out how performance scales with the availability of patterns becomes an important question.

4.3 . On the number of patterns

Generation with pattern-filling generation requires patterns and lists of mentions. Lists of mentions can be extracted from knowledge bases. What about patterns? Patterns are a limiting factor because they will likely require the most effort to produce or obtain. Access to sentences in the domain of the task is not guaranteed due to privacy issues, a domain not covered by readily available data, or any other type of problems that occur during data acquisition. Even if you have access to data, patterns are extracted from labeled sentences. Labeling the sequences is not an easy task, as it requires multiple points of view to avoid biases, and that does not guarantee the quality of the annotation. If you don't have access to unlabeled data, you have to produce the patterns by hand with the same problems as annotating. For both of those cases, understanding the way performances scale with the number of patterns becomes an important problem to avoid spending a lot of time and energy on something that could bring little to no improvement.

4.3.1 . Experiment Description

This section studies the impact of the number of patterns on the tagger performances. For this experiment, the quantity of natural data used is irrelevant. As the focus is on the impact of patterns, the quantity of data that will vary in the experiment is the number of patterns used for generation. The whole training set is used for the extraction of the patterns and mentions. For this experiment, the patterns and mentions extracted are unique, as in the first pattern-filling experiment. The patterns are then shuffled once, and the sets of patterns used are the first patterns of the list. Since the number of patterns is the quantity that is studied, the list of mentions must be of fixed size and content. Models are prone to over-fitting. The more mentions there are in the lists, the higher the quantity of overlap there will be with mentions in the test set. This will positively impact the performances of the taggers through means that are not the means studied. As a result, we have chosen to cut the mention lists obtained from the extraction over the full training set in half.

4.3.2 . Results

The results of this experiment can be found in Figure 4.8. Before analyzing the impact of patterns, let us note the impact of the reduction of mentions. The NLU corpora (ATIS, MEDIA, SNIPS) seem to be more affected by the reduction of mentions at a high number of patterns than the NER corpus (I2B2). This degradation is most important for ATIS, where the F_1 score with high quantities of data is below 80 points versus around 95 points with full-sized mention lists. Apart from ATIS, the F_1 score is below 20 points for ten or fewer patterns. ATIS's score is around 40 points which might indicate the presence of a pattern containing one or more slots of the most populated concepts. A difference appears between the behavior of NER corpus and NLU corpora. On NLU corpora, the performance of additional

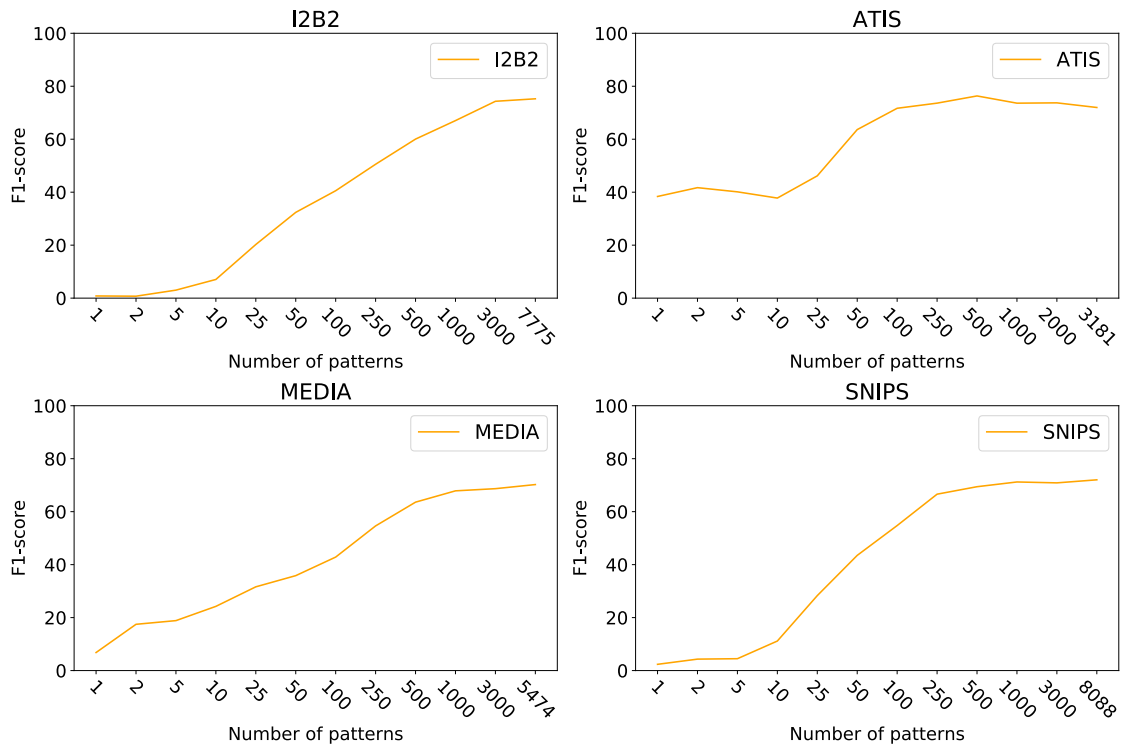


Figure 4.8: F_1 score of BiLSTM trained on generated data. The generated data was generated using half of the full-sized corpora’s available mentions and a number of patterns found in the abscissa.

patterns reaches a plateau early on, whereas, for the NER corpus, the plateau might have been apparent with more data. This plateau is located around 100 patterns for ATIS, 250 patterns for SNIPS, and between 500 and 1000 patterns for MEDIA. Pattern scaling is much more pronounced with NER corpora, as can be seen in Figure 4.8. The F_1 score of the taggers trained on the data scaled up to 3000 patterns for I2B2. This number of patterns is not something that can be achieved in a low-resource setting without any pattern-based generation method. The only recommended quantity we can give is to prepare at least 500 patterns for the training set. This quantity is enough to reach the plateau in NLU tasks and brings the taggers to over 50 F_1 score for NER.

4.3.3 . Conclusion

To generate a training set, at least 250 patterns are needed before diminishing returns start to be felt on some corpora. This quantity varies from 250 patterns for ATIS to 3000 for I2B2. From this state of affairs, the best compromise is to collect at least 500 patterns to generate a generated training set. This amount of patterns does not cover validation or testing, which would probably require naturally labeled data to be effective. To obtain patterns, one might write them or label sentences. If the chosen method is to write the patterns, during writing, you will likely produce a sentence that might be used. If it is to label available data, you have labeled data at hand. This data might as well be used during training. In the next section, we take a look at augmentation using pattern-filling generation.

4.4 . Augmentation

The previous experiments have been with generated data. Obtaining patterns is either done through labeling sentences or writing the patterns or sentences. It is highly likely that the writing process will result in valid examples rather than just patterns. Natural labeled sentences are likely to be obtained through the pattern obtention process. These natural sentences should be used in the training data set as they most likely are better examples than generated ones.

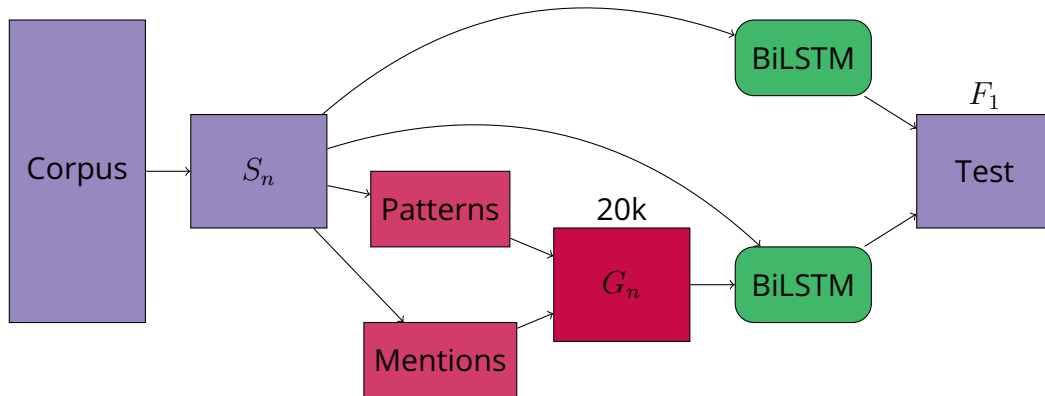


Figure 4.9: Overview of the pattern generation pipeline for augmentation. The S_n data subset is sampled from the training corpus. Patterns and mentions are extracted from this subset and used to generate G_n of size 20k. Both S_n and G_n are used to train BiLSTMs that end up being evaluated on the Test set from the corpus.

For this experiment, a similar setup to the one in the generated data section with a major revision is used, as can be seen in Figure 4.9. This time, the natural subset S_n is augmented with the generated set G_n to train the BiLSTM for the generated side. This configuration of the experiment is tested using natural distribution in the generation process. The natural distribution of mentions is used as it was the overall best method, but a method is also tested where both the patterns and mentions' natural distributions were preserved.

4.4.1 . Results

F_1 score curves can be observed in Figure 4.10 and the exact results can be found in Annex B. A tagger trained on augmented data following both mention and pattern distribution is better than a tagger trained on natural data at lower quantities of data. For I2B2, a tagger trained on augmented data using both distributions is better than a tagger trained on augmented data that follows the distribution of mentions. They are, however, equivalent between 500 and 1000 sentences. For the ATIS and MEDIA corpora, both experiments produce equivalent results. For SNIPS, taggers trained on augmented data following the natural distribution of patterns and mentions are better than their counterparts trained on data following the distribution of mentions. This is true on all sizes of subsets except full size. As expected, the results of augmentation are better than fully generated data sets.

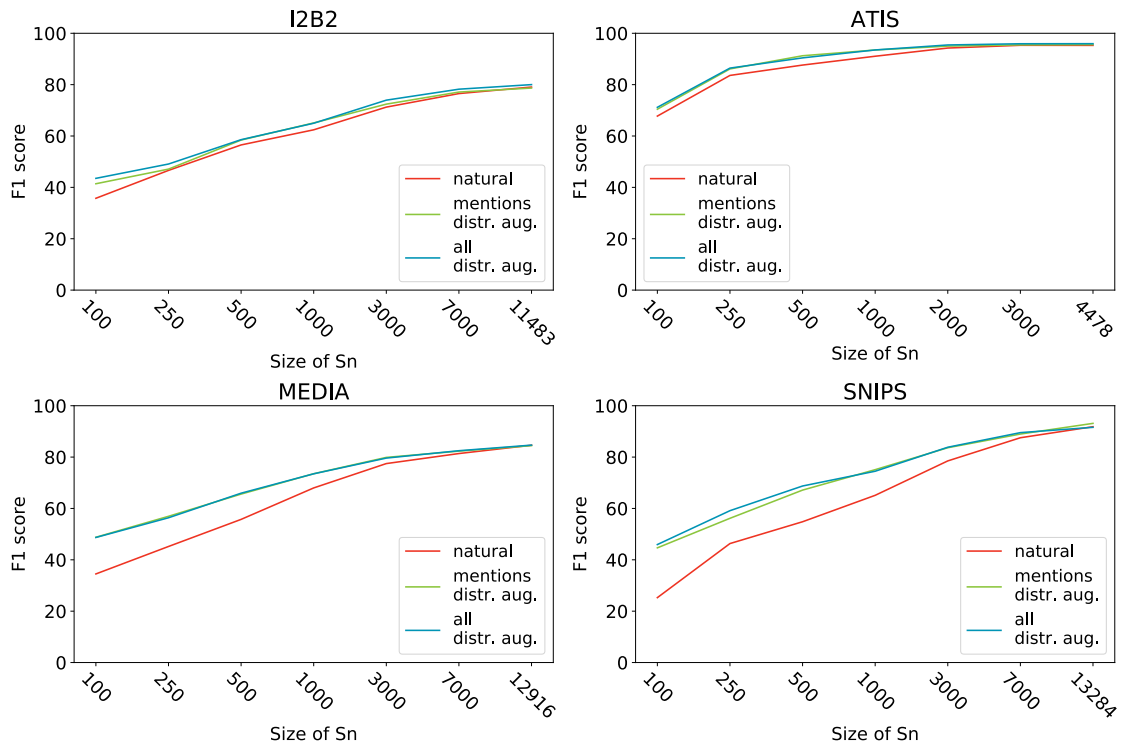


Figure 4.10: F_1 score of natural and generated data with mention and pattern redistribution with BiLSTM taggers.

4.4.2 . Conclusion

The source of patterns is most likely going to be labeled natural data. This data should be in the training set of the taggers. This section has explored the use of augmentation instead of pure generated data. As expected, using available natural data does not degrade previous performances and can be better for tagger performances.

In these preliminary experiments, we have explored multiple ways of using the pattern-filling generation method. These experiments have led us to the conclusion that augmentation was the most effective use of this generation method. However, the scope of these experiments was limited. In the next section, we propose a final experiment in which augmentation is used while following the methodology from Chapter 3.

4.5 . Final experiments

Previous sections have shown that pattern-filling generation can be used to generate generated training sets. They have also shown that pattern-filling generation can be used as an augmentation tool for the training sets of sequence labeling tasks. However, the experiments used to make these claims had many limitations. They were done using the full-sized validation set. This is not realistic with regard to the few-shot setting the experiments should be set in. The setting is low access to data, which means that the natural data should be funneled to-

ward the training set and not the development set. The oversized development set might have an impact on the performances of the methods. Another problem that needs to be addressed is the absence of measurements of the variability of the method. This information is needed in order to bring a stronger validation of the previous results and frame them as more than possible good runs. The multiplicity of corpora in the previous experiments served this purpose but was not enough. Finally, the taggers used to produce the results in the previous sections are outdated. These BiLSTM taggers were cheap to train and produced correct results at the time of the experiments. Stronger baselines exist, such as BERT [Devlin et al., 2018]. We use BERT-based taggers for this experiment. This section presents an experiment tackling these issues.

4.5.1 . Settings

The experiments in this section follow the methodology presented in the methodology chapter 3. The BERT + MLP taggers replace the BiLSTM taggers used previously. The BERT models used are pre-trained models trained on large quantities of data. This architecture provides better results on the full-sized corpora than the previous BiLSTM architecture.

The multiplication of experiments means that it becomes more costly to run such experiments. Some of the previous experiments were cut to reduce the cost of the experiments. Only the augmentation experiments are done in this section, as in a real-case scenario, no one would discard its natural data. The number of corpora used for the experiments has been reduced. The corpora we have chosen to use are CoNLL 2003, I2B2, and MEDIA. These corpora cover multiple problems linked with our objective dataset, which is AIDA. They cover the target language with MEDIA, a specialized domain with I2B2, and the Named Entity Recognition task with CoNLL 2003.

In particular, while doing the experiment on multiple corpora gave us indications on the viability of the methods employed, it did not give us a true account of the capacity of the method. To be able to assess the performances of the method on a single corpus, the sampling bias coming from the creation of the subsets needs to be taken into account. Experimenting using multiple seeds becomes crucial to be able to bring a definitive assessment of the pattern-filling generation method. The results presented are the average and standard deviation across five seeds. Multiplying the seeds comes at a cost. Five seeds are used and not more in order to have an indicator of the dispersion of the results but stay within reasonable energy and time consumption. Finally, our target setting has been reduced to a low-resource setting. Our results are computed with subsets S_n with sizes ranging from 50 to 1000 sentences. The development sets are sampled from the natural development set and are of the same size as S_n unless S_n is larger than the natural development set. The generated set G_n is still composed of 20,000 sentences.

4.5.2 . Results

The results are presented in Table 4.1. The BERT + MLP architecture shows improvements on the baseline compared to the BiLSTM, with around 10 additional

Size	50	100	250	500	1000
CoNLL 2003 EN					
Baseline	57.8±2.9	70.1±5.6	81.2±1.8	85.3±1.1	88.0±0.3
PatGen 20k	66.8±3.3	71.7±2.8	80.3±1.6	83.6±1.8	86.7±0.7
PatGen 20k (p+m)	66.3±2.6	74.3±2.1	80.8±1.3	84.5±1.7	86.8±0.7
I2B2					
Baseline	37.0±3.2	49.4±1.6	63.2±1.6	72.0±0.7	77.3±0.4
PatGen 20k	45.7±4.4	56.6±2.2	67.4±0.2	73.8±0.9	79.1±0.8
PatGen 20k (p+m)	44.0±6.7	56.0±1.6	67.8±1.2	73.5±1.3	79.1±0.8
MEDIA					
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
PatGen 20k	50.7±3.8	57.5±1.5	67.1±2.2	74.0±0.9	78.0±0.3
PatGen 20k (p+m)	50.0±3.9	57.9±2.1	68.3±1.1	74.3±0.8	78.8±0.3

Table 4.1: F_1 score of taggers trained following the methodology of Section methodology. The results are the average and standard deviations over five seeds. Baseline taggers are trained using only the subset S_n . PatGen 20k taggers are trained using the subset S_n and the training set generated with the pattern-filling generation method. PatGen 20k (p+m) taggers are trained using the subset S_n and the training set generated with pattern filling generation with natural pattern and mention distributions. The size of the subset is a number of sentences.

points at each size step for CoNLL and MEDIA and around 15 additional points at each size step for I2B2. For I2B2, BioBERT + MLP on a subset of size 1000 with augmentation obtains performances close to the performance of the BiLSTM trained on the full-sized corpora with 77.3 against 79 as can be seen in Table B.1 in Appendix B. The bulk of this improvement is thanks to the architecture as it obtains better performances with S_{1000} at 77.3 F_1 score than a BiLSTM with S_{7000} at 76.5. However, the missing 1.5 points to bridge the gap to full-sized BiLSTM is reached thanks to pattern-filling generation. For CoNLL, the performance of BERT + MLP is over the performance of the BiLSTM at full size, with 88.0 points against 87.5 Such a behavior is not as prevalent for MEDIA, where the current score for size 1000 is over the score of the BiLSTM with 79.1 against 77.5. As expected, the standard deviation is greater with smaller subsets of data for all methods and all corpora, with values ranging from 2.6 to 6.7 at size 50 and values ranging from 0.3 to 0.8 at size 1000.

In our results, the performances of the base pattern-filling generation method and the method using natural pattern and mention distribution are presented. For I2B2, both methods produce data that allows the model to outperform the baseline with 45.7 and 44.0 for the base and redistributed method against 37.0 for the baseline. The uniform distribution works better than using pattern and mention distribution with S_{50} and S_{100} with 45.7 and 56.6 versus 44.0 and 56.0. For CoNLL, the method not following the natural distribution and the methods following the natural distributions produce improvements over the baseline for sizes 50 and 100, with respectively 66.8 and 66.3 against 57.8 for the baseline and

71.7 and 74.3 against 70.1 for the baseline. The pattern-filling generation then degrades the score of the baseline at larger sizes of S_n . For MEDIA, the situation is close to the CoNLL situation, with the augmented set improving performances up to size 500 and then degrading the performances at size 1000 with 78.0 and 78.8 for the base pattern-filling method and the method preserving the natural distribution against 79.1. This degradation changes the results seen in the previous experiment, where augmentation improved performances for MEDIA. This change can come from the reduction of the development set or a change of behavior due to the change of model.

4.6 . Conclusion

This chapter presented our interpretation of the pattern-filling generation method. This method is a computationally cheap algorithm that combines empty-slotted sentences with mentions. We have shown that training sets generated with this algorithm produce taggers with equivalent or better performances than the natural data from which the patterns and mentions are extracted. The lower-sized subsets of data benefited the most from using generated data. Generating new sentences with pattern-filling requires patterns and mentions. It is hard to quantify the number of mentions needed to produce data containing enough information for training a tagger, as the number of mentions found for each concept varies greatly. We also made the claim that obtaining mentions is easier than obtaining patterns. Further experimentation using outside sources of mention is needed to validate those claims. We have instead chosen to focus on quantifying the number of patterns. Patterns are harder to obtain as they are task-related and are not part of knowledge bases. Our results on tagger performance scaling depending on patterns are that 500 patterns are enough to produce good results on easy and medium tasks, but more difficult tasks require as many patterns as they can get. For simpler tasks, this number of patterns is more than enough to reach a plateau of performance. For harder tasks, the more patterns the better, but 500 should be enough to train a first tagger. Obtaining patterns is done through manual labeling or writing. In these cases, it is likely that the patterns are obtained from labeled sentences. Using pattern-filling generation as a tool to augment these labeled sequences was a successful endeavor. Performances of taggers were increased at all sizes of data used for the BiLSTMs. The final experiment showed that with better-performing models, the methods obtained more nuanced results. On I2B2, the results are positive at all natural subset sizes, but for MEDIA and CoNLL, this is only true for the smaller sizes of data.

Pattern-filling generation is a viable method of augmentation to obtain significant improvements in tagging performances at low-resource. This method has been used in practical situations [Roca et al., 2020]. This method creates rigid sentences due to the fixed nature of patterns. It is possible that further improvements could be made with less rigid sentence generation. We will explore using language modeling for labeled sentence generation in the next chapter.

5 - Language modeling and Semi-supervision

In previous experiments, language models were not used to generate text. Language models have, however, shown a high capacity for generating text. Recent language models can generate human-like text. We want to use this generative capacity to generate labeled text. Generating text with labels would require fine-tuning, which requires high quantities of data to avoid over-fitting. With this in mind, using pre-trained language models to generate the text became a natural step of the process. The question is how labels are assigned to the newly generated examples. Since a small quantity of labeled data and a possibly very large quantity of unlabeled data are available, supervised learning cannot be applied to train taggers. However, another paradigm of machine learning exists that can use a mix of labeled and unlabeled data to train models. This paradigm is semi-supervised learning. Our first goal is to show that semi-supervised learning can use synthetic data as its unlabeled set. Our second goal is to evaluate the quality of the pseudo-labeled dataset generated by the language models and labeled during the semi-supervised process.

This chapter presents our work combining semi-supervised learning and language modeling. The semi-supervised paradigm is described in the first section. From this paradigm, we have chosen to use the tri-training algorithm first with a static set of data, then with a set of data dynamically generated. To generate these sets of data, multiple generation setups are proposed based on existing language modeling objectives. Finally, the quality of the data generated is tested by training taggers on it.

5.1 . Semi-supervised learning

Semi-supervised learning is a learning paradigm aiming at improving the performance of the models trained by adding unlabeled examples to the training set [Van Engelen and Hoos, 2020]. This paradigm has multiple branches depending on how the unlabeled data is used. Inductive semi-supervised learning [Van Engelen and Hoos, 2020] aims at improving the performances of models through the addition of unlabeled data. For Named Entity Recognition, *pseudo-labeling* is a method that has been used [Chen et al., 2019b]. *Pseudo-labeling* is one of the semi-supervised learning methods. The unlabeled data receives pseudo-labels from the models trained. This pseudo-labeled data is then used alongside labeled data to train the models. Using weak labels or pseudo-labels at various stages of training has worked for NER [Wang et al., 2021b]. The models at previous learning states generate these pseudo-labels.

This branch of semi-supervised learning contains a variety of algorithms. These algorithms depend on how many models are trained and which models are used to generate the pseudo-labels. The most simple algorithm is self-training [Yarowsky, 1995], which trains one model and uses the model to create the pseudo-labels, with all the caveats that ensue. DAGA [Ding et al., 2020] successfully used

this method with synthetic data. Other methods using ensembles of models were created to reduce the biases induced by having the same model generate its pseudo-labels. Co-training [Blum and Mitchell, 1998] is a method for training two models in which each model generate the pseudo-labels for the other. A generalization of this method exists in which an ensemble of n models is trained, and the pseudo-labels for a model m_j are produced using a voting system across the n other models.

The semi-supervised algorithm used in this section is a variant using three models called tri-training [Zhou and Li, 2005]. This method has been used to solve Clinical Concept Extraction in the medical domain [Chen et al., 2019b] on new data. Each model is trained on unlabeled data that receives pseudo-labels from the other two models. Pseudo-labeled sentences are used when the two models labeling the sentence reach an agreement. In tri-training with disagreement [Søgaard, 2010], the sentences are discarded when all the models reach an agreement. Our goal is to produce data with tri-training. This version of tri-training is not implemented as it would reduce the quantity of relevant generated data when tri-training is used dynamically. We propose a generative tri-training algorithm, which uses the voting system to discard newly generated samples on which there is no agreement.

5.2 . Tri-training with a static set of generated data

As presented in the previous section, semi-supervised learning is classically applied to a dataset composed of labeled and unlabeled examples. The available data is a small quantity of task-related data in the low-resource setting tackled. This small quantity of data is assumed to be labeled because the size of the data should allow it. As semi-supervised learning requires additional unlabeled data, some additional unlabeled data is needed. We chose to produce this unlabeled data using pre-trained large language models. This section presents the preliminary experiments that examine the feasibility of using LLMs to generate the unlabeled data needed by a semi-supervised learning method: tri-training. The question is whether the data generated by LLMs is of sufficient quality to be used in the semi-supervised scenario. The semi-supervised algorithm used, the tri-training algorithm [Zhou and Li, 2005], is introduced in the first subsection. The generation methods using GPT-2 [Radford et al., 2019] used to generate the synthetic data are then presented. This section is concluded with an analysis of the results obtained.

5.2.1 . Tri-training

This section provides details on the tri-training process for sentence tagging and how language modeling is leveraged as an unlabeled data provider. Tri-training is an inductive semi-supervised learning [Van Engelen and Hoos, 2020] method using an ensemble of three models. Pseudo-labeled sentences are sentences labeled by the models during training. Pseudo-labels are subject to change throughout the training procedure depending on model predictions. In this work, pseudo-labeled sentences are trained upon without distinction from regular labeled sen-

Algorithm 1 Tri-training ([Zhou and Li, 2005], [Ruder and Plank, 2018])

```
1: for  $i \in \llbracket 1 ; 3 \rrbracket$  do
2:    $m_i \leftarrow \text{train\_model}(\text{sampling}(S_n), m_i)$ 
3: end for
4: while Any  $m_i$  still learns do
5:   for  $i \in \llbracket 1 ; 3 \rrbracket$  do
6:      $L_i^t \leftarrow \emptyset$ 
7:      $j, k \leftarrow \llbracket 1 ; 3 \rrbracket - |i|$ 
8:     for  $x \in G_n$  do
9:       if  $m_j(x) = m_k(x)$  then
10:         $L_i^t \leftarrow L_i^t \cup \{(x, m_j^{t-1}(x))\}$ 
11:       end if
12:     end for
13:   end for
14:   for  $i \in \llbracket 1 ; 3 \rrbracket$  do
15:      $m_i^t \leftarrow \text{train\_model}(L_i^t \cup S_n, m_i^{t-1})$ 
16:   end for
17: end while
```

tences. It is possible to do special treatment such as balancing the batches between natural and pseudo-labeled data or adding a weight to the loss of the pseudo-labeled data. These techniques are not done within tri-training, so they will not be done in this thesis. During tri-training, the models are trained in a supervised learning manner on a set of labeled and pseudo-labeled data. The models used for the ensemble are taggers that are trained to perform the sequence labeling task. Further description of the taggers can be found in chapter 3. The tri-training algorithm found in Algorithm 1 is described in the following paragraphs and in Figure 5.1. The addition of GPT-2 as a generator is highlighted in the figure. Tri-training is an episodic training method that stops when each model of the ensemble has stopped improving. The most crucial feature of tri-training is the construction of the training set of the models. This is shown from line 4 to line 9 in Algorithm 1 and in the second line of Figure 5.1. In the following lines, the indexes i , j , and k represent the elements of $\llbracket 1 ; 3 \rrbracket$ where i is the referencing the current model, and j and k reference the other models. For each model m_i , a pseudo-labeled set L_i is constructed. L_i is composed of the unlabeled sentences $x \in G_n$ for which the predictions of the models m_j and m_k $i \notin \{j, k\}$ are equal. These predictions are added to L_i alongside x as their pseudo-labels. A threshold can also be used to remove uncertain annotations. However, it was concluded that it was not necessary for simple tri-training [Ruder and Plank, 2018]. The models are then trained on both the natural and synthetic data $S_n \cup L_i$. S_n is the labeled training data. The operations described above are repeated until all models have stopped learning.

The central part of Algorithm 1 described above assumes that models are sufficiently trained and different to create varied pseudo-labels. The models are pre-trained to achieve these prerequisites. The models m_i are pre-trained on different

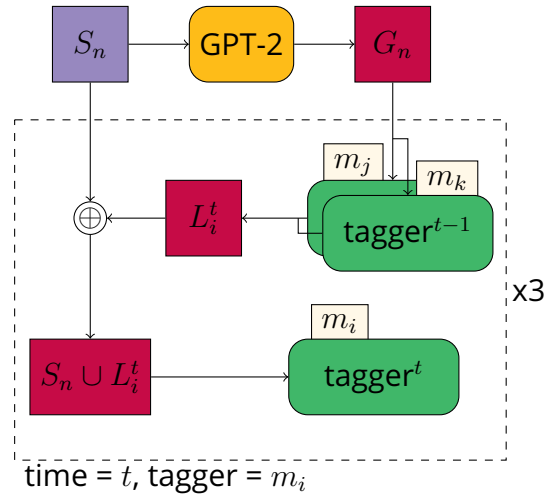


Figure 5.1: Tri-training with unlabeled data G_n generation. In rectangles are the data sets, and in rounded rectangles are the different models. The procedure is shown at episode t for model m_i . The initialization is not represented and is done by sampling with replacement from S_n .

random subsets of the labeled data S_n . These subsets are made by sampling with replacements from the training set [Ruder and Plank, 2018]. This operation is also referred to as *bootstrap sampling* in [Zhou and Li, 2005]. Sampling the pre-training data is done to introduce variety in the train sets of the three models without incurring performance losses.

For inference, the ensemble of the three different models is used together with a voting system. The labels with the highest summed score across the three models are kept.

As a semi-supervised learning algorithm, tri-training requires a great amount of unlabeled examples. The specificity of our study is the use of a generator to create unlabeled examples. Applying semi-supervised learning methods is more complicated when there is no unlabeled data. The text of the labeled data is used as the context for the generation model. The generative model is used in two different ways: (i) follow-up sentence generation, where the next sentence is generated using causal modeling, and (ii) sentence completion, where a sentence is cut short and completed with, as shown in Figure 5.2.

The first generation method used is follow-up sentence generation. Large language models like GPT-2 [Radford et al., 2019] are trained on texts containing multiple sentences. This kind of model should be able to generate the follow-up sentence from the context. Using these models out-of-the-box should work without any fine-tuning. Follow-up sentence generation is applied to generate new examples. With this method, our aim is to generate new sentences that are within the same domain but have different structures. The second method used is sentence completion. The end of the sentence is removed and completed using the language model for this method. Our aim is to generate alternative contexts to the part of the sentence that is kept with this method.

5.2.2 . Generation methods

The unlabeled set G_n is generated with GPT-2 [Radford et al., 2019]. Hugging-Face’s implementation¹ is used. In this preliminary experiment, the small version of GPT-2 is used. This packet is generated **once** before training. The text from the labeled train set is used as the context to generate entailed examples. For each labeled example, five follow-up sentences are generated. The language model is also used for sentence completion. In this case, the original text is cut arbitrarily and completed using the model. Each labeled example is cut to 75%, 50%, and 25% of its length. In each of these cases, five completed sentences are generated. This amounts to a total of 20 synthetic examples per natural example. It is, in practice, slightly less than that because sequences made exclusively of different types of whitespace, newlines, and other such noise are filtered out. Generated examples can be seen in Figure 5.3.

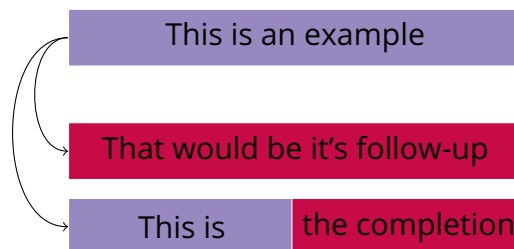


Figure 5.2: Generation methods examples. In blue is the initial example and in red is the generated text. The first generated example is from sentence follow-up, and the second is from sentence completion.

5.2.3 . Tri-training experimental settings

The main focus of this experiment is the use of tri-training without natural unlabeled data. As explained previously, the unlabeled data generated is used as the unlabeled data of tri-training. Tri-training requires one development set and one validation set: the first for the training of each model m_i , the second to validate the stagnation of the models across episodes. We chose to split the corpora’s initial development set in half to fulfill each of those purposes, as shown in Figure 5.4. As this is a preliminary experiment, the sentences without tags are excluded from the pseudo-labeled set. This is done to avoid a possible problem at very low resources where the pre-trained models are not trained enough and produce sentences with empty tag sequences where they should not. However, our results show that these precautions might not be necessary. The result of the tri-training procedure is an ensemble of three models. Inference using this ensemble is made with a simple voting system. Voting is done by summing the scores output of each tag across all models and picking the highest.

5.2.4 . Results

This section presents the results obtained on the CoNLL [Sang and De Meulder, 2003] and I2B2 [Uzuner et al., 2011] corpora. The results are a comparison of

¹<https://huggingface.co/gpt2>

ORG
MDS was founded in 1978.

PER
 And it was then that Jussi Graf's

MISC
3_x86_64.tar.gz""); // We'll add this [...]

problem
 FOLLOW US ON TWITTER!

test treatment
Disease tolerance test for benz

treatment
 -12 10:27:28] RavenQueen > she's been so [...]

Figure 5.3: Examples of generation. The first three examples are from CoNLL, and the last three are from I2B2. Each series is formed of an example of completion and two examples of sentence follow-up. The examples were cherry-picked to show both positive and negative aspects of generation, be of short length, and be labeled by the models. On CoNLL's completion example, only a full stop was added. In I2B2's completion example, the context was "FOLLOW" and was too short and generic to bring the sentence to the medical domain. The second examples for both corpora are okay. The third example for both corpora happens when short formulaic sentences are used as context. For CoNLL, it is the common -DOCSTART- and for I2B2, it is a date.

scores from a baseline against the individual performances of taggers trained in the tri-training process and the ensemble of three taggers.

Baselines are the results of models trained in a supervised manner only on the natural training data. The experiments are done using ten seeds. The results for each size n are the average and the standard deviation over the scores obtained with the ten different seeds. The results in Table 5.1 show consistent performance increases between each subset size. Seqeval [Ramshaw and Marcus, 1995, Nakayama, 2018] is used to compute the results. I2B2 F_1 ranges from 36.2 (size 50) to 77.4 (size 1000), and CoNLL F_1 ranges from 59.9 (size 50) to 87.7 (size 1000). Smaller sizes show a higher standard deviation, with 5.8 for I2B2 and 3.3 for CoNLL at size 50.

Tri-training produces three trained models supposed to be used as an ensemble of models. With constraints such as memory consumption or inference time, one might want to use a single model for inference. For such cases, the results of single models have been reported. The Δ_{unique} results show the deltas between each of the three individual models m_i and the baseline. For each subset S_n , it is an average of 30 deltas.

The purpose of tri-training is to obtain an ensemble of three models. The results of the ensembles reported are the deltas between the performances of the ensembles and their respective baselines. These results can be found within Table 5.1 at the Δ_{ensemble} line and in Figure 5.5.

Our method obtains higher results on average on all subsets and on both corpora. Generally, on I2B2, tri-training allows for a Δ_{ensemble} to range from +4.32 (S_{50}) to +1.80 (S_{1000}). On CoNLL, it otherwise ranges from +2.98 (S_{50}) to +0.71

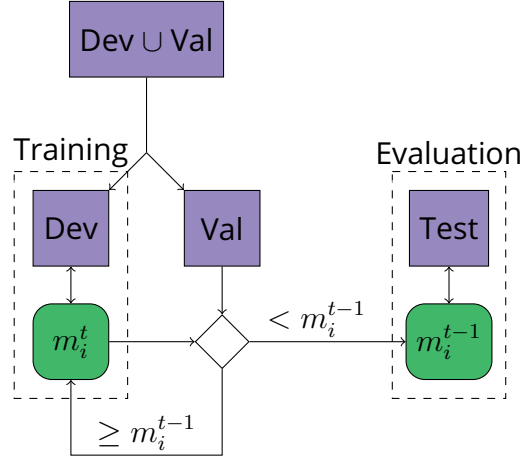


Figure 5.4: Validation mechanism for a model m_i . We split the development set in two, one to optimise the model within a training episode, the other one to optimise across episodes.

Subsets		S_{50}	S_{100}	S_{250}	S_{500}	S_{1000}
I2B2	Baseline	36.2 ± 5.8	49.2 ± 3.2	64.3 ± 1.4	71.4 ± 0.8	77.4 ± 0.6
	Δ Unique	$+3.9 \pm 1.9$	$+2.6 \pm 2.4$	$+1.9 \pm 1.3$	$+1.9 \pm 0.7$	$+1.3 \pm 0.8$
	Δ Ensemble	$+4.3 \pm 1.8$	$+3.1 \pm 2.4$	$+2.5 \pm 1.2$	$+2.5 \pm 0.7$	$+1.8 \pm 0.8$
CoNLL	Baseline	59.9 ± 3.3	69.2 ± 3.9	80.7 ± 2.0	84.7 ± 0.9	87.7 ± 0.4
	Δ Unique	$+2.3 \pm 2.0$	$+0.1 \pm 3.6$	$+1.1 \pm 1.1$	$+0.5 \pm 0.8$	$+0.3 \pm 0.4$
	Δ Ensemble	$+3.0 \pm 2.0$	$+0.8 \pm 3.7$	$+1.8 \pm 1.2$	$+1.1 \pm 0.7$	$+0.7 \pm 0.4$

Table 5.1: Average and standard deviation of F_1 score across ten seeds for the baseline results. Average and standard deviation of the deltas between the performances of each individual tri-trained tagger and their respective baselines at Δ Unique lines. Average and standard deviation of the deltas between the performances of tri-trained ensembles and their respective baselines at Δ Ensemble lines. Corpora used are I2B2 and CoNLL.

(S_{1000}). The Δ Unique shows, as expected, lower gains than Δ ensemble, ranging from $+3.93$ (S_{50}) to $+1.28$ (S_{1000}) for I2B2 and $+2.33$ (S_{50}) to $+0.27$ (S_{1000}) for CoNLL. Out of the 50 individual runs for each corpus, one is negative for I2B2, and five are negative for CoNLL. Impacts of the negative results are seen on the average results of CoNLL at subset size 100. Three seeds yield negative gains at this size, with one having extreme (-8.6 points) negative gains. Removing this extreme result in the average calculation brings the Δ ensemble score closer to expected values ($+1.89$). Performances of individual models on CoNLL are within the standard deviation of negative results. This is not the case for I2B2. These results show that using the ensemble is a more stable solution. Overall, the method is most consistent with subsets of size 250 plus, as the average performance of tri-trained ensembles is above the standard deviation of the baseline.

5.2.5 . Conclusion

Leveraging pre-trained models to improve performances on specific tasks is a common approach. With recent improvements to language modeling, recent

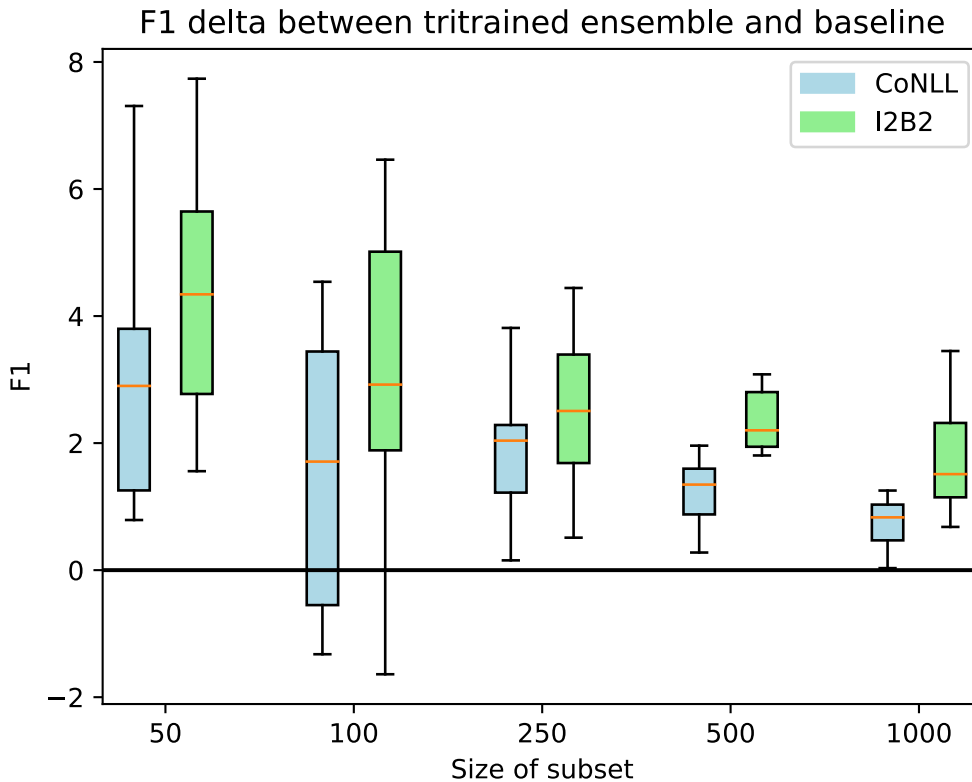


Figure 5.5: Boxplot of CoNLL and I2B2 deltas between tri-trained ensemble and baseline (Δ_{ensemble}). For each subset size, the left boxplot is CoNLL, and the right boxplot is I2B2.

models are often used directly to solve tasks. We have shown that models can be used to generate data that brings improvements to the performances of taggers on NER and Clinical Concept Extraction in a low-resource context. The gains are between 3 and 4 points of F_1 score on subsets of data of size 50. Gains are overall positive on the sizes of the subsets tested. The higher the data size, the lower the gain is. This preliminary experiment has shown that large language models are suitable tools to generate unlabeled examples for semi-supervised learning for NER. In the next section, we will improve the experimental setting and merge the generation process in the tri-training algorithm.

5.3 . Generative Tri-training

Generating a static set of unlabeled sentences has allowed us to produce improvements in performances for both the ensemble of taggers and the individual taggers trained. However, some of the generated data wasn't used. Generated data that was labeled at some point should be used as a new prompt for text generation. For that reason, we have made the generation process closer to the tri-training algorithm. The tri-training algorithm adds pseudo-labeled training examples to the training set when two out of the three taggers produce the same pseudo-label sequence. Generation is now done **at each episode** of the tri-

training process and uses the ensemble of models as a filter with the mechanism described in the previous sentence. This allows previously generated sentences to be used for generation. Our hypothesis is that this process plays to the strength of generation by producing a high quantity of varied examples.

This set of experiments uses new and improved generation methods with an independent evaluation of each method. We then present how these methods are used inside the generative tri-training process. Additional experiments with self-training using a similar generative approach were introduced to have better baseline comparisons.

5.3.1 . Generation methods

How to generate new unlabeled samples is at the core of this chapter. Language modeling is the tool used to generate new examples. The most traditional way to do language modeling is to predict tokens left-to-right. These models use the context on the left to predict the following tokens on the right. Causal modeling is used for our first two methods of generation. The model used for these generation methods is GPT-2² [Radford et al., 2019]. In particular, we have upgraded to the large version of GPT-2 as compared to the previous section.

Other methods of language modeling exist, such as masked language modeling. These models predict tokens that have been masked within the sequence using the overall context. This type of modeling is not used for generation in our experiments, but we will use T5 [Raffel et al., 2020], which was trained using span masking.

Text-to-text modeling is a recent approach to NLP brought by T5 [Raffel et al., 2020], where every task is seen as a text to which the model must produce a text response. The task on which T5 was pretrained is a span replacement task. Our last two generation methods use this span replacement capacity to edit the mention or the context of a labeled or pseudo-labeled sentence. For these methods, T5 v1.1³, which is a version of T5 with updated activations and which is not trained for the end tasks, is used. All generated methods are applied to labeled or pseudo-labeled sentences. An example of each generation method can be seen in Figure 5.6.

Follow-up

The first generation method tested is follow-up sentence generation. It is the same as in the previous setting. The principle of this method lies in the left-to-right modeling of the following text. The text from a labeled or pseudo-labeled sentence is used as the left context for modeling. As the left context is a sentence, consequently, follow-up generates the following sentence. Our hypothesis is that the new sentences brought to the dataset by this method will bring variety to the set while still being in the domain.

²<https://huggingface.co/gpt2-large>

³https://huggingface.co/google/t5-v1_1-large

Completion

In a similar manner to follow-up, completion is a method using left-to-right modeling. In this case, modeling is used to complete the sentence. The principles behind this method are the same as in the previous experiment, but with modifications. The position from which to generate is chosen by sampling between the first tokens of the mentions within the sentence. If the sentence does not contain any mention, the position is sampled between 25% and 75% of the sentence length. Our hypothesis is that this method will bring variety to the mentions while keeping the classic modeling approach.

Mention replacement

This method is the first method of text-to-text modeling. Our aim is to bring diversity directly to the objects of interest with this method. The mentions of labeled sentences are replaced using the model through span masking. Span masking is the technique used to pre-train T5. One mention from the initial sentence is replaced each time this type of generation is used. If more than one mention is present, the mention that should be replaced is sampled. Sentences without mentions are ignored.

Our hypothesis is that language models have learned information about any subject during their training. The subjects of our interest are the concepts we wish to label in sentences. Our hypothesis is that given a context that contains an instance of a given concept, the model will be able to replace this instance with other instances of the same concept. We expect this method to work best in combination with other methods.

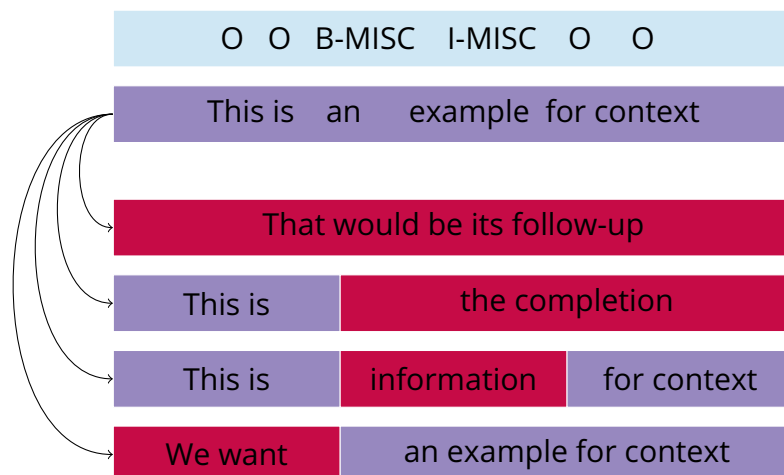


Figure 5.6: Generation methods examples. In blue is the initial example and in red is the generated text. In light blue is the label sequence, which is used to select mentions and context. The first generated example is from follow-up, the second is from sentence completion, the third from mention replacement, and the fourth from context replacement.

Context replacement

Context replacement functions in a similar way to mention replacement. It is the non-labeled text that is replaced. A list of spans containing all the context spans is created. Context spans are spans of contiguous outside tokens. The size of the spans is not limited. The span which gets replaced is sampled. Sentences without mentions are ignored. Our hypothesis is that taggers learn to predict both from the context and the mentions. From this hypothesis, we derive that learning from more varied contexts might benefit the tagger.

5.3.2 . Semi-supervised learning methods

This section describes the tri-training algorithm with on-the-fly generation. Generating a fixed-sized unlabeled dataset for semi-supervised learning seems arbitrary. There is no guarantee that the models will use the data to learn because there is no guarantee that the models will reach an agreement on the pseudo-labels. In practice, some of the data will never be used. To solve this issue, we have devised the generative tri-training algorithm. This algorithm is a modification of the tri-training algorithm to fit generation.

Tri-training requires a step of training to initialize the taggers to be able to produce pseudo-labels. This step is called the pre-training step when discussing the tri-training algorithm. It is represented between lines 1 and 4 of Algorithm 2. Any reference to pre-training refers to this step. Normally, the pre-training step is done on subsets of data sampled with replacement from the labeled set. However, we have found that adding an episode of training on the complete labeled set S_n after the sampled set improved performances of tri-training greatly. This addition is found in line 3 of Algorithm 2.

Algorithm 2 and Figure 5.7 reflect the changes made to tri-training. With tri-training, an ensemble of three models $m_i, i \in \llbracket 1 ; 3 \rrbracket$ is trained. Training is split into episodes during which all three models are trained. For a given model m_i , episodic training stops when the model's score on the validation set is lower than the previous episode; this process is described in Figure 5.4. At the beginning of each episode (line 8 of Algorithm 2), new subsets of pseudo-labeled data L_i^t are generated. These subsets are generated from the labeled and previously pseudo-labeled sets. The newly generated examples L_i^t are added without their labels to the previously generated examples U^t for the next episode. Each of these pseudo-labeled sets L_i^t is then augmented using previously generated examples U^t with the classic tri-training mechanism (lines 10 to 17 of Algorithm 2). Finally, between lines 18 and 22, the models that still require training are trained.

This algorithm solves the issue of unused data by discarding freshly generated examples on which there is no agreement, as shown in Figure 5.7. A soft disagreement modification is added to the tri-training algorithm by having a 50 percent chance for each model to keep pseudo-labeled examples on which there is a complete agreement, as shown in lines 13 and 14. Our motivation behind this change is to keep 'solved' examples seen by the models once in a while to avoid forgetting but still benefit from a lower quantity of data to improve training speed.

For tri-training and semi-supervision in general, two sets of validation data are needed to evaluate our training. The development set from our corpora is split in

Algorithm 2 Generative Tri-training

```
1: for  $i \in \llbracket 1 ; 3 \rrbracket$  do
2:    $m_i^{-2} \leftarrow \text{train\_model}(\text{sample}(S_n), \text{BERT})$ 
3:    $m_i^{-1} \leftarrow \text{train\_model}(S_n, m_i^{-2})$ 
4: end for
5:  $t \leftarrow 0$ 
6:  $U^0, L_1^{-1}, L_2^{-1}, L_3^{-1} \leftarrow \emptyset$ 
7: while Any  $m_i$  still learns do
8:    $L_1^t, L_2^t, L_3^t \leftarrow \text{gen\_with\_LM}(S_n \cup \bigcup_{i=1}^3 L_i^{t-1})$ 
9:    $U^{t+1} \leftarrow U^t \cup \text{remove\_label}(\bigcup_{i=1}^3 L_i^t)$ 
10:  for  $i \in \llbracket 1 ; 3 \rrbracket$  do
11:     $j, k \leftarrow \llbracket 1 ; 3 \rrbracket - \{i\}$ 
12:    for  $x \in U^t$  do
13:      if  $m_j^{t-1}(x) = m_k^{t-1}(x)$  and  $(m_i^{t-1}(x) \neq m_j^{t-1}(x) \text{ or } \text{drop}(p > .5))$  then
14:         $L_i^t \leftarrow L_i^t \cup \{(x, m_j^{t-1}(x))\}$ 
15:      end if
16:    end for
17:  end for
18:  for  $i \in \llbracket 1 ; 3 \rrbracket$  do
19:    if  $m_i$  still learns then
20:       $m_i^t \leftarrow \text{train\_model}(S_n \cup L_i^t, m_i^{t-1})$ 
21:    end if
22:  end for
23:   $t \leftarrow t + 1$ 
24: end while
```

two. The first half serves as a development set for the training during one episode. The second half is used to compare models across episodes and define when a model should stop learning. This mechanism is described in Figure 5.4.

5.3.3 . Experimental settings

We have devised a set of experiments to evaluate our tri-training with generation. These experiments compare our method to standard supervised training and tri-training with available task data. Generation with semi-supervision is also tested with two semi-supervised learning algorithms: tri-training and self-training.

Our baseline is supervised learning on natural data without any semi-supervision. Our **topline is tri-training** with natural data as the unlabeled data. These results will be the comparison points for generative tri-training. The topline uses part of the remaining sentences from the corpora as unlabeled data for tri-training. We consider a set of 10,000 sentences from which we extract the supervised (baseline) and the unlabeled set for tri-training (topline) corpora.

At each episode of semi-supervised learning, a fixed quantity of data is generated. Our aim is to generate 5,000 examples for each episode. This quantity is large enough to bring a good amount of information but not too large in order to avoid slowing down the training process. A lower amount of data is gen-

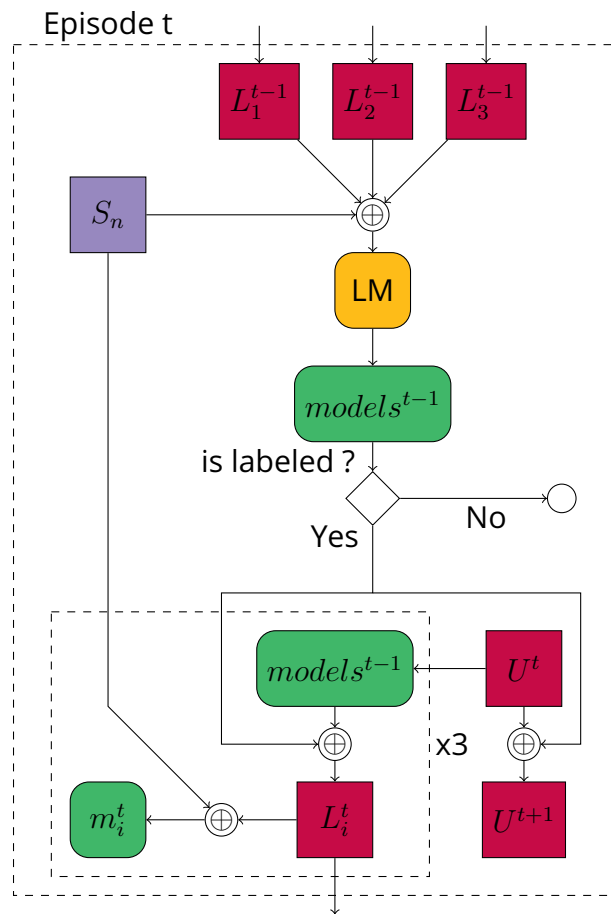


Figure 5.7: Generative tri-training step. In rectangles are the data sets, and in rounded rectangles are the different models. The initialization is not represented and is done by sampling with replacement from L . Data generated by the language model LM where $models_{t-1}$ have not reached an agreement are discarded.

erated for earlier episodes as the quantity of context available, the labeled and pseudo-labeled data, is smaller. This small quantity of data might also see its signal drowned if too much data is generated, as seen with the topline method. The first two steps are eased in by only generating 500 and 2,500 instead of 5,000 examples.

Generation is done in the context of semi-supervised learning. For tri-training, the generated examples are split among three sets, one for each model being trained within tri-training. The new sets of data are generated by applying the respective generation method to sampled examples from the natural dataset and the pseudo-labeled sets from the previous episode. The three models then label the generated text output by the generation method. If two or more models reach an agreement, the new data is saved for training. This pseudo-labeled data is sent to either the packet of the model that disagrees if two models agree or one of the sets at random if the three models are in agreement. This is done to make the model that disagrees learn the information of the two other models. If no agreement is reached, the data is discarded. New examples are generated until the expected quantity, or the maximum number of generation operations al-

Corpus	CoNLL				
Size	50	100	250	500	1000
Baseline	60.0±2.9	70.5±4.7	81.3±1.7	84.6±1.2	87.7±0.3
ΔTopline	+8.6±2.5	+5.6±2.0	+4.2±0.7	+3.8±1.0	+2.1±0.3
ΔFollow-up	+8.9±2.6	+4.4±2.8	+3.1±0.4	+2.8±1.0	+1.7±0.2
ΔCompletion	+7.8±1.9	+4.7±4.0	+2.7±1.0	+2.9±0.7	+1.7±0.2
ΔMention rep.	+9.0±2.1	+4.1±2.6	+2.8±1.1	+2.6±0.8	+1.2±0.3
ΔContext rep.	+7.3±1.9	+2.9±2.3	+1.2±0.4	+1.7±0.7	+1.3±0.4
ΔCombined	+9.3±2.7	+4.6±2.6	+3.4±1.0	+2.9±1.0	+1.6±0.4

Corpus	I2B2				
Size	50	100	250	500	1000
Baseline	39.3±3.1	50.4±1.4	63.8±0.7	71.5±1.0	77.6±0.5
ΔTopline	+8.3±2.2	+8.9±2.2	+6.7±1.2	+5.4±0.7	+3.4±0.7
ΔFollow-up	+6.3±2.6	+4.5±2.6	+4.2±1.6	+3.6±1.0	+2.3±0.8
ΔCompletion	+6.3±1.6	+5.6±2.8	+4.9±1.5	+4.3±1.7	+2.4±1.0
ΔMention rep.	+10.4±4.5	+7.4±2.9	+7.1±1.7	+4.6±1.0	+2.6±0.5
ΔContext rep.	+5.6±1.5	+7.7±3.1	+6.0±1.2	+4.2±1.0	+2.4±0.4
ΔCombined	+7.7±1.5	+6.4±1.7	+5.7±1.9	+4.4±1.2	+2.9±0.5

Table 5.2: Table of results of generative tri-training. The size row refers to the number of sentences in S_n . The results are averages and standard deviations computed on five seeds. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

lowed, is reached. This limit is put in place as a safeguard, but it is never reached in practice. This limit is set at 10,000 generation steps. Each generation method generates five new sentences for each input sentence.

5.3.4 . Results

This section presents the results of our experiments. First, the baseline and topline results are presented to set up the comparison points for our method. The results of generative tri-training and self-training are then presented.

The results of baselines and the Δ 's between topline and baseline can be seen in Table 5.2. As expected, the topline results are way above the baseline results. Baseline results range from 60.0 F_1 score with subsets of size 50 to 87.7 F_1 score at size 1000 for CoNLL. The topline gain on CoNLL ranges from 8.6 for the smallest subset to 2.1 for the largest subset. For I2B2, the baseline results are smaller than CoNLL's, ranging from 39.3 to 77.6 F_1 score. I2B2's topline gains range from 8.9 at size 100 to 3.4 at size 1000. The gains at size 50 are slightly below, with 8.3.

In this first experiment, each generation method is used independently. This experiment aims to test each of the methods to assess the impact on performance. The results related to tri-training in this section can be found in Table 5.2 for low-resource and in Table 5.4 for full-sized results. Self-training results are

Corpus	CoNLL				
Size	50	100	250	500	1000
Baseline	60.0±2.9	70.5±4.7	81.3±1.7	84.6±1.2	87.7±0.3
ΔFollow-up	+4.8±2.9	+1.4±3.6	-0.1±1.1	+0.6±1.0	-0.0±0.3
ΔCompletion	+3.9±2.7	+0.8±2.0	-0.4±1.0	+0.6±0.9	+0.1±0.5
ΔMention rep.	+6.0±2.9	+2.5±2.8	+0.7±0.6	+0.8±0.4	+0.2±0.5
ΔContext rep.	+4.3±2.0	-0.5±4.0	-1.1±1.2	-0.2±0.7	-1.0±0.4
ΔCombined	+4.9±3.5	+1.4±3.1	-0.4±0.4	+0.9±1.2	+0.1±0.4
Corpus	I2B2				
Size	50	100	250	500	1000
Baseline	39.3±3.1	50.4±1.4	63.8±0.7	71.5±1.0	77.6±0.5
ΔFollow-up	+0.8±1.7	+0.2±2.3	-0.7±1.3	-0.2±0.5	-0.8±0.9
ΔCompletion	-0.5±1.6	+0.3±2.9	-1.4±0.8	-0.9±1.8	-0.7±0.5
ΔMention rep.	+4.5±2.6	+2.9±1.8	+3.0±1.1	+2.0±1.1	-0.1±0.3
ΔContext rep.	+3.6±1.8	+3.6±2.5	+3.9±0.9	+2.2±0.7	+0.3±0.1
ΔCombined	+1.6±1.0	+0.7±1.7	-0.1±1.2	+0.6±1.6	-0.5±0.2

Table 5.3: Table of results of generative self-training. The size row refers to the number of sentences in S_n . The results are averages and standard deviations computed on five seeds. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

found in Table 5.3. For tri-training, each method displays positive gains at each subset size. Some individual generation methods even overtake the topline. This is the case for the follow-up and mention replacement generation methods for CoNLL with 8.9 and 9.0 F_1 score average gains. It is also true for the mention replacement method for I2B2 with 10.4 and 7.1 F_1 score average gains at size 50 and 250, respectively. The gains follow the same trends as the topline, with diminishing returns the higher the size of the training subset. For CoNLL, context replacement is the lowest-performing generation method overall. For I2B2, the answer is not as clear-cut. With a training subset of size 50, context replacement is the worst-performing method. Still, at higher quantities of data, it becomes the second best-performing method, and follow-up generation becomes the worst-performing method. Mention replacement is the best individual generation method on I2B2, with the best or second-best results at every subset size. Overall, GPT-2-based methods yield lower results than T5v1.1-based methods on I2B2. For self-training, as expected, the results are not as good as tri-training. Overall, all gains are lower than their tri-training counterparts, with some negative results. Mention generation remains the best individual method as it is the best method for CoNLL and is contending with context replacement depending on the subset size for I2B2.

	CoNLL	I2B2
Caseline	92.0	86.6
Follow-up	91.8	88.0
Completion	92.1	87.5
Mention rep.	92.2	87.7
Context rep.	92.2	87.6
Combined	92.3	87.5

Table 5.4: F_1 scores of generation methods on full-sized corpora. Results computed on one seed. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

The combined method is done by sampling the generation method from the available methods for each generation step. As every generation method yields positive results, they were all added to the pool of available methods. Results of the combined generation method for tri-training can be found in Table 5.2. The method displays two different behaviors on the two corpora. For CoNLL, the combined generation method is the best generation method. For I2B2, the combined generation method is more akin to an average method. It is the best method with subsets of size 1000 with a 2.9 F_1 score. It is otherwise lower or equal to the replacement methods, except for context replacement on the smallest subset. The combined generation is higher than the left-to-right modeling methods for I2B2. Overall the combined method has a higher Pearson correlation factor for follow-up and completion methods with .98 and .97 than with mention and context replacement with .92 and .80, respectively. Those coefficients were computed with the results reported in Table 5.2.

Results for generative tri-training on the full-sized corpora can be observed in Table 5.4. For CoNLL, our best results are with the combined method with a .3 improvement. There is, however, a .2 deterioration with follow-up. Overall, on full-sized CoNLL, positive results are obtained. For I2B2, all methods significantly increase the F_1 score with improvements ranging from .9 for completion and the combined method to 1.4 for follow-up.

5.3.5 . Conclusion

Our proposed method works when generic language models are available. The method was tested on a general-purpose domain with CoNLL and on a specialized domain with I2B2. GPT-2 based generation methods are less effective in the specialized domain than in the general-purpose domain. Thus the model used will have an impact on the qualitative results of the generation methods. Furthermore, our methods are not tested with a reduced validation set. There is no guarantee that the generated data is meaningful or within the domain. We know from the generated data that some examples are outside the domain. This is especially true for the completion method if the mention where the generation starts from is close to the beginning of the sentence.

Our proposed method of augmentation combining the semi-supervised learning method known as tri-training with generation using large language models is effective in the setting tested. We have proposed several methods of generation and their combination to use within this augmentation framework and a combination of those methods. These methods were compared in a low-resource setting against supervised training and tri-training with natural unlabeled data. Generating data for self-training was also tested. Overall, generation works best with tri-training and improves performances from the baseline. Having unlabeled data from the task still results in better performances in most cases. The best generation method for both CoNLL and I2B2 is mention replacement using T5 v1.1 as a span-replacement tool.

In the following section, we will evaluate the impact of the quantity of data in the development set.

5.4 . Impact of a small development set and performances of generated data

We have shown previously that language models could be used to generate the unlabeled training set for tri-training. These experiments were done using the full-sized development set of the corpora experimented on. As shown in the previous section and in Figure 5.4, the development set played a more important role in pseudo-label-based semi-supervised learning. It is divided to be used for both model training evaluation and episodic evaluation. As such, reducing the size of the development set might have a large impact on the final performances. In this section, the impact of following the methodology presented in Chapter 3 is studied. Our expectations are that the smaller size of the development set will have a large impact on generative tri-training.

5.4.1 . Small development set

Let us first take a look at the impact of the development size on the baseline. The impact is mainly felt at smaller sizes where there is a degradation of performances of a couple of points. The impact is not felt at higher quantities of data. The impact is expected to be stronger for tri-training.

The results of generative tri-training with a smaller set of data can be observed in Table 5.5 for CoNLL, Table 5.6 for I2B2 and Table 5.7 for MEDIA. Performances have overall shifted towards lower scores for both the baseline and the ensembles of models. The behavior of the different generation methods for CoNLL is not strongly affected, with the combined method being the overall best method. The combined method is not the best for subsets S_n containing 50 and 1000 sentences, but it is quite close to the best methods at these sizes, with a 0.3 difference at 50 and a 0.1 at size 1000. The GPT-2 based methods are not as clearly ahead of the T5-based methods as they were with large development sets. For I2B2, the behavior of the various methods is quite different. Below 500 sentences, causal-based methods are outperformed by span-replacement methods. They become equivalent from subsets of 500 sentences onward. The combined method is hit by the reduction of the development set at size 50, where it performs the worst.

Size	50	100	250	500	1000
Baseline	57.8±2.9	70.1±5.6	81.2±1.8	85.3±1.1	88.0±0.3
Follow-up	64.3±2.2	73.7±3.3	83.5±1.1	87.2±0.7	89.4±0.2
Completion	63.5±2.2	71.6±2.8	83.4±1.3	86.8±0.7	89.2±0.3
Mention rep.	64.5±1.2	72.7±3.1	82.8±1.9	87.2±0.7	89.1±0.4
Context rep.	62.8±3.2	72.3±2.7	82.5±1.4	86.3±0.7	89.0±0.1
Combined	64.2±3.5	74.6±3.3	83.6±1.1	87.3±0.9	89.3±0.5

Table 5.5: Table of results of the ensemble of taggers trained with generative tri-training with small development set on CoNLL. The size row refers to the size of S_n as a number of sentences. The results are averages and standard deviations computed on five seeds. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

Size	50	100	250	500	1000
Baseline	37.0±3.2	49.4±1.6	63.2±1.6	72.0±0.7	77.3±0.4
Follow-up	38.9±3.0	50.5±2.4	66.1±0.4	75.1±1.4	79.6±0.8
Completion	38.2±4.7	51.6±2.5	67.8±1.4	74.4±0.6	80.5±0.5
Mention rep.	41.4±3.0	54.9±2.0	68.9±1.9	75.3±0.8	79.9±0.7
Context rep.	42.8±2.4	55.0±3.2	68.2±1.1	74.3±0.5	79.4±0.7
Combined	37.9±4.6	52.4±2.2	68.1±0.5	74.7±0.7	79.9±0.8

Table 5.6: Table of results of the ensemble of taggers trained with generative tri-training with small development set on I2B2. The size row refers to the size of S_n as a number of sentences. The results are averages and standard deviations computed on five seeds. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

For MEDIA, this is the first tri-training experiment. All the tri-training examples produce higher than baseline results. The higher the size of natural data, the closer the results of the methods are to each other. At size 50, the results range from a 46.5 score for follow-up generation to 48.3 for both completion and context replacement against 44.3 for the baseline. At size 100, the results range from 55.6 for follow-up generation to 57.4 for mention replacement against 53.9 for the baseline. Follow-up is still behind the other generation methods at size 250, with a 67.9 score against 68.6 for completion, but it is still higher than the baseline, which garners a 66.4 score. At size 500, the results range from 75.6 for follow-up generation and the combined methods to 76.2 for completion against 74.3 for the baseline. Finally, the scores range from 81.2 to 81.4 at size 1000 against 79.1 for the baseline. Follow-up is clearly the worst method using BLOOM for the MEDIA corpus. Completion is the method that obtains the best performance the most amount of time.

The Pearson correlation factor is computed with the delta of the methods with the baseline. This factor is used to understand the correlation of the individual methods with the combined method. The correlation factors with the combined method show a similar trend to the results with a large development set

Size	50	100	250	500	1000
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
Follow up Completion	46.5±3.0 48.3±3.3	55.6±1.7 56.5±1.0	67.9±1.7 68.6±1.7	75.6±1.0 76.2±0.9	81.2±0.5 81.3±0.3
Mention rep. Context rep.	47.7±2.6 48.3±3.2	57.4±1.0 56.6±0.9	68.5±1.4 68.3±1.4	75.9±1.0 75.9±0.9	81.4±0.6 81.1±0.5
Combined	47.4±2.9	57.0±0.8	68.3±1.8	75.6±1.2	81.2±0.6

Table 5.7: Table of results of the ensemble of taggers trained with generative tri-training with small development set on MEDIA. The size row refers to the size of S_n as a number of sentences. The results are averages and standard deviations computed on five seeds. Follow-up and completion use BLOOM as a generator. Mention and context replacement use mT5.

with a Pearson of 0.83 and 0.79 for follow-up and completion with the combined method and 0.69 and 0.42 for mention and context replacement with the combined method. The performances of the combination of the method still depend more on the causal-language-modeling-based methods.

For CoNLL, combining the methods is still the best approach. For I2B2, the span-replacement techniques overall perform better. Generative tri-training yields better results than the baseline in all cases. While tri-training is a good method, the final ensemble of models is three times the size of an individual model. The ensemble trained might profit from being an ensemble against a single tagger. For this reason, a comparison to a single tagger trained on the data generated through the tri-training process is necessary. In the next section, we will test whether the performances of the ensemble be replicated by a single model.

5.4.2 . Training on the data generated

Our goal is to produce generation methods capable of creating synthetic text with tags. The way tri-training is evaluated in the previous sections does not reflect this goal. The capacity of tri-training to produce an ensemble of taggers using synthetic unlabeled datasets was evaluated. This endeavor has been successful. The capacity of the generated data to train a single tagger is evaluated in this section. The performance of these taggers to the performances of the individual taggers trained during tri-training are compared. Using the individual taggers trained during tri-training would be the simplest way to obtain another single tagger. The individual taggers behave in a similar manner to the ensembles, albeit with smaller scores, as is shown in Tables 5.8 5.9 5.10. Degradation of performances is observed for context replacement when applied to CoNLL when using subsets of size ranging from 100 to 500. Sentence completion also obtains negative results at size 50 for I2B2. For MEDIA, context replacement also obtains negative results on the single models at size 500.

Generative tri-training adds pseudo-labels to the unlabeled generated set. Each of the individual models is trained on synthetic data for which the other two models have produced equal pseudo-labels. How do we choose the data that is output by generative tri-training? We have chosen the simple approach of keeping only

Size	50	100	250	500	1000
Baseline	57.8±2.9	70.1±5.6	81.2±1.8	85.3±1.1	88.0±0.3
Follow-up	63.1 ±3.0	72.3±3.3	82.2±1.2	86.2±0.8	88.6 ±0.5
Completion	62.2±2.7	70.4±2.8	82.3 ±1.5	85.5±1.3	88.5±0.4
Mention rep.	62.9±1.9	71.0±3.3	81.3±1.9	85.8±0.9	88.2±0.5
Context rep.	60.3±4.6	69.6±2.5	80.9±1.9	84.8±1.1	88.0±0.3
Combined	62.7±3.2	72.6 ±3.6	81.5±2.0	86.3 ±1.0	88.4±0.5

Table 5.8: Table of results of the unique taggers trained with generative tri-training with small development set for CoNLL. The size row refers to the size of S_n in number of sentences. The results are averages and standard deviations computed on five seeds and the three models for each tri-training run. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

Size	50	100	250	500	1000
Baseline	37.0±3.2	49.4±1.6	63.2±1.6	72.0±0.7	77.3±0.4
Follow-up	37.7±2.8	49.5±2.5	64.7±1.2	73.9±1.6	78.6±0.8
Completion	36.9±5.0	50.4±2.5	66.5±1.7	73.1±1.0	79.5 ±0.9
Mention rep.	39.9±3.3	53.5 ±2.0	67.4 ±2.4	74.0 ±1.2	78.6±1.0
Context rep.	40.5 ±3.0	53.0±3.4	66.4±1.4	72.9±1.0	78.1±0.8
Combined	37.0±4.7	51.1±2.9	66.6±1.2	73.2±1.5	78.9±1.0

Table 5.9: Table of results of the unique taggers trained with generative tri-training with small development set for I2B2. The size row refers to the size of S_n in number of sentences. The results are averages and standard deviations computed on five seeds and the three models for each tri-training run. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

the data where the three models produce the same sequence of tags. Other ways of creating the synthetic corpus are not studied. It is a possible axis of development of this method. BERT+MLP taggers are trained on S_n with the addition of the synthetic corpus produced with generative tri-training. The F_1 scores of the taggers on the test sets can be found in Table 5.11 for CoNLL, in Table 5.12 for I2B2, and in Table 5.13. The results are, in general, better than the singular taggers, except for MEDIA, where the difference is less clear. Exceptions exist, such as context replacement for CoNLL with the degradation from the baseline score at size 1000 and the degradation from the singular taggers at size 250. Context replacement for I2B2 also performs worse than with the singular taggers at sizes 50 and 100. For MEDIA, on the 25 generation method x size settings, 13 obtain better results when taggers are trained on the data produced, versus three that obtain equivalent results and nine that obtain negative results versus the single taggers trained during tri-training. This phenomenon is accentuated at sizes 50 and 100, for which there are only three positive results against six negatives. At size 250 and above, the results are mostly positive, with ten positive comparisons against

Size	50	100	250	500	1000
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
Follow-up	45.1±3.5	54.7±1.7	66.9±1.4	74.4±1.1	80.2±0.6
Completion	47.1±3.2	55.5±1.2	67.2±1.6	74.7±1.0	80.1±0.5
Mention rep.	46.4±2.8	54.3±4.1	67.1±1.5	74.7±1.1	79.8±1.5
Context rep.	46.8±3.3	55.6±1.0	66.8±1.4	73.8±1.8	80.1±0.7
Combined	47.2±3.9	55.6±1.2	67.0±1.7	74.6±1.2	79.9±0.6

Table 5.10: Table of results of the unique taggers trained with generative tri-training with small development set for MEDIA. The size row refers to the size of S_n in number of sentences. The results are averages and standard deviations computed on five seeds and the three models for each tri-training run. Follow-up and completion use BLOOM as a generator. Mention and context replacement use mT5.

three negative. The methods of generation also play a role in the improvement of taggers trained on the outputs against the single taggers. Follow-up generation and mention replacement have four out of the five sizes tested where the taggers trained on the outputs outperform the taggers trained during tri-training. Outside of these occurrences cementing context replacement as the worst method, the scores obtained by training a tagger on the generated sets are mostly better than the individual models trained during generative tri-training. The performance of the taggers trained on the outputs rivals the performance of the ensembles of taggers on some generation methods. Follow-up obtains equivalent performances on CoNLL and better performances for the taggers trained on the outputs for I2B2. For sentence completion, the ensembles perform better at over 500 sentences on CoNLL but worse on smaller sizes. The same behavior is seen on I2B2. Mention replacement follows the same pattern, with ensemble performances being better with more than 250 sentences for CoNLL. However, for I2B2, the mention replacement method is better with a single model except at a subset of size 1000 where it is equivalent. For context replacement, the scores are clear, and the ensembles outperform the single taggers on all settings except I2B2 with size 250. The combined method is worse on all sizes for CoNLL and better on all sizes for I2B2. For MEDIA, the results are not as positive, as only the combined method at size 50 produces better results than the ensembles of models.

Generative tri-training generates data that can be used to train taggers. Mention replacement and follow-up generation seem to be the most consistent generation methods. With 100 or more sentences, the combined method is also a safe option if you want to generate synthetic text using the generative tri-training method. It must be noted that the generation methods behave differently between the two corpora. This might be inherent to the generation methods, but it can also come from the models used by these generation methods. GPT-2 was trained using WebText [Radford et al., 2019], a corpus of articles and pages linked from Reddit. T5, on the other hand, was trained using a corpus of examples extracted from Common Crawl they have named C4 [Raffel et al., 2020]. These pre-training corpora might not contain similar quantities of information related to the

Size	50	100	250	500	1000
Baseline	57.8±2.9	70.1±5.6	81.2±1.8	85.3±1.1	88.0±0.3
Follow-up	64.2±2.1	73.6±3.0	83.9±0.9	87.3±0.6	89.0±0.7
Completion	63.9±3.3	72.1±2.6	83.3±1.4	86.6±0.9	88.6±0.1
Mention rep.	65.2±1.6	72.9±3.2	82.6±1.6	86.3±0.9	88.4±0.4
Context rep.	61.2±4.7	70.4±2.8	80.2±1.8	85.3±1.4	87.3±0.3
Combined	62.9±3.2	74.4±2.6	82.8±1.3	87.0±0.6	88.6±0.5

Table 5.11: Table of results of BERT+MLP taggers trained on data generated with generative tri-training with small development set on CoNLL. The size row refers to the size of S_n in number of sentences. The results are averages and standard deviations computed on five seeds. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

Size	50	100	250	500	1000
Baseline	37.0±3.2	49.4±1.6	63.2±1.6	72.0±0.7	77.3±0.4
Follow-up	41.4±2.6	51.1±1.6	67.8±1.3	75.4±1.1	79.8±0.7
Completion	39.1±3.3	52.6±2.6	68.2±1.3	74.1±1.1	80.3±0.4
Mention rep.	43.0±2.8	56.1±1.2	70.2±1.3	75.7±1.0	79.8±0.6
Context rep.	39.8±2.0	52.5±2.6	68.5±1.1	73.7±1.1	78.9±0.8
Combined	39.1±3.6	53.5±2.6	69.1±0.8	74.8±0.9	80.3±0.5

Table 5.12: Table of results of BERT+MLP taggers trained on data generated with generative tri-training with small development set on I2B2. The size row refers to the size of S_n in number of sentences. The results are averages and standard deviations computed on five seeds. Follow-up and completion use GPT-2 as a generator. Mention and context replacement use T5v1.1.

domain of training sets used. Other models could be used instead of GPT-2 and T5, which might change the landscape of the results. Due to these various factors, we cannot recommend the best generation method if you want to use this method. However, avoid starting experimenting with context replacement as it is the least safe generation method with our settings. Our recommendation is to use follow-up or mention replacement first. While we have seen that generative tri-training creates data that improves the training of taggers, it is not the only method presented in this work. How does this method fare against pattern-filling generation?

5.4.3 . Comparison to pattern-filling generation

In the previous chapter, a pattern-filling generation method was introduced. This method produced labeled synthetic data that improved tagger performance for I2B2 at all sizes and improved tagger performance at the smaller sizes for CoNLL. The results of the pattern-filling generation method found in Table 4.1 are compared to the result of the taggers trained on the outputs of the generative tri-training algorithm found in Tables 5.11 5.12 5.13. With subsets containing 50 sentences, taggers trained with pattern-filling generation data outperform the

Size	50	100	250	500	1000
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
Follow-up Completion	46.4±2.6 47.1±2.6	54.6±2.8 55.1±0.4	67.2±1.2 67.0±1.2	74.9±0.9 75.0±1.3	80.3±0.4 80.1±0.6
Mention rep. Context rep.	46.9±3.0 46.3±3.1	56.6±1.4 55.0±1.1	67.6±1.8 67.1±0.6	74.4±0.7 75.0±1.0	80.1±0.5 79.8±0.5
Combined	47.0±2.6	55.4±1.2	67.5±1.0	74.6±0.9	80.0±0.7

Table 5.13: Table of results of BERT+MLP taggers trained on data generated with generative tri-training with small development set on MEDIA. The size row refers to the size of S_n in number of sentences. The results are averages and standard deviations computed on five seeds. Follow-up and completion use BLOOM as a generator. Mention and context replacement use mT5.

taggers trained with generative tri-training data. There is a 1.6 points difference between the best methods for CoNLL and a 2.7 points difference between the best methods for I2B2. With 100 sentences, the combined method is better than pattern-filling generation for CoNLL. For I2B2, the taggers trained on the uniform distribution pattern-based data are still 0.5 points over mention replacement. For 250 and more sentences, the generative tri-training data is overall better than pattern-filling generation data, with only context replacement being lower at 250 sentences for CoNLL and 500 and 1000 sentences for I2B2. For MEDIA, at sizes 50 and 100, using either the uniform distribution or the natural distribution in pattern-filling generation outperforms all taggers trained on the outputs of tri-training. At size 250, the uniform method of pattern-filling generation is outperformed by taggers trained on the outputs of tri-training with most methods of generation, but the taggers trained on data generated with natural distribution are still best. For sizes 500 and above, the taggers trained on the outputs of tri-training are better than the taggers trained using pattern-filling generation.

Pattern-filling generation remains the best method to use for smaller sizes of available data. However, this method does not scale as well as generative tri-training and even hinders performances on CoNLL at higher quantities of data. These methods both have their strength, and it is possible that improving the quality of the tri-trained taggers using pattern-filling generation could improve the final performances. Using pattern-filling generation on a set of synthetic data produced by the tri-training generation method is also a possible improvement.

5.5 . Conclusion

Using language models to generate unlabeled data for semi-supervised learning is a successful method. Generation was mostly tested for the tri-training algorithm. Self-training was also tested in its simplest form. An improved self-training algorithm exists, which might improve the results obtained. Improvements to the tri-training method also exist, especially on the filtering side of the algorithm. We had chosen to compare the base versions of the algorithms as they are easier

to implement when the generative process is added on top of them. Tri-training performed better than self-training with these settings.

Models that perform better, thanks to the generation of data, were produced. The tri-training algorithm trains an ensemble of three taggers that were trained using synthetic data. We have shown that a single tagger trained on the synthetic data containing all the examples where the three taggers from tri-training had an equal tagging scheme obtained better performance than the ensemble. The performance of these taggers was compared to the performances obtained with pattern-filling generation in the previous chapter, and we have observed that tri-training synthetic data is better with 100 or more sentences as the pool of initial data. Both the ensembles of models and taggers trained on the product of tri-training outperform the models produced with pattern generation at higher quantities of data. However, the combination of the ensemble method and episodic training and the use of large language models causes this method to be quite heavy in terms of computation. This might be a factor when choosing the method to use when generating data.

We have raised questions on the performances of the generation methods relative to the models used. The performance of models reflects the data they were trained on. We have used GPT-2, BLOOM, T5, and mT5, which were not pre-trained on the same dataset. The impact a model has on the quality of the results obtained by a generation method is unknown. Studying the impact of different generators is a path that can be explored in the future. For now, we have chosen to explore the use of replacement methods to produce labeled text in the next chapter.

6 - Modeling with patterns

In the previous chapters, two approaches to labeled sentence generation were presented and studied. The first approach uses pattern-based methods for labeled text generation. The second approach focuses on combining language modeling with semi-supervised learning to produce labeled sentences. This chapter looks into the possible connections that could be made between these two approaches.

Thus, the first approach is the pattern-filling generation method which works by replacing parts of the sentences associated with a concept with other parts associated with the same concept. The choice of the replacement parts is made via a sampling method. The benefit of this method is to leverage the structural knowledge from the pattern and the semantic knowledge of the lists of mention. Pattern-filling generation is the best method of the two previously shown below 100 sentences. The second approach studied is language modeling coupled with tri-training. This method uses language modeling to produce new sentences. These sentences are both filtered by the ensemble of three taggers and used as part of the training procedure. The benefit of this method is to leverage the knowledge instilled in the pre-trained large language models to produce sentences that improve tagger performances. This method is the best of the two with larger quantities of data, meaning 250 and more sentences.

Using language models to replace parts of sentences is used in NLPAug [Ma, 2019] and in a more recent study [Sutiono and Hahn-Powell, 2022]. New models exist, such as T5 [Raffel et al., 2020], which are trained on a more suitable task for replacement. T5 is a model trained on the span replacement task. This task consists of reconstructing 15% of the sentences that were replaced by a special token. This technique can be used to replace parts of a sentence. This highly resembles what the pattern-filling generation does. In the previous chapter, T5 was used to replace mentions or context from labeled sentences with semi-supervised learning. The labels associated with the sentences were, however, not kept with semi-supervised learning. In this chapter, we evaluate the use of T5 to apply span replacement while keeping the labels.

6.1 . Span-replacement with T5

In the previous chapter, T5 was used to generate text, whereas, in this section, T5 is used to generate text while keeping the label structure, which ultimately generates labeled text. Our goal with such a method is to compare the pattern-based approach to a language modeling approach that works in similar ways. The spans selected for replacement are bound by the tags. They either replace a mention or a part of the context. This works like patterns-filling generation, with the exception that patterns can be generated with this method. However, applying the original tagging structure can be done by applying the tagging scheme of the replaced part to the generated part. A mention of a certain concept c would be

replaced by a string of text that would also be labeled with the concept c , and the replaced context would retain the empty annotations. Will the plethora of information stored within a language model be useful for this sort of task? The previous work with semi-supervised learning has shown that these methods brought an improvement to baseline performances. Can they bring similar improvements without the need for the tri-training process, which has both the benefits of filtering the generated data and the cost of iteratively training the taggers?

A preliminary experiment is done in which the span-replacement method is applied without any additional parts. This first experiment obtains negative results. Data needs to be analyzed to understand the problems that cause the taggers to obtain degraded performances. From these analyses, we add filtering mechanisms to the method in order to try to improve the results.

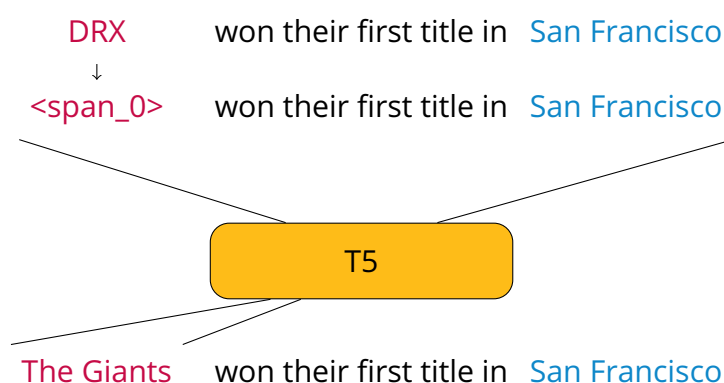


Figure 6.1: Example of augmentation using T5 with labels. The first mention is replaced by a special token indicating to T5 which part to generate. The sentence with the special token is fed to T5. T5 outputs a span that is placed instead of the special token. Information about the tags is kept. Here in red are organizations, and in blue locations.

6.1.1 . Preliminary experiment

This section presents the experimental settings for the span replacement experiment. Span replacement is applied to three corpora: CoNLL, I2B2, and MEDIA. These corpora cover two languages: English and French. Multiple versions of T5 exist, such as the multilingual version mT5¹[Xue et al., 2021], the pre-trained version T5v1.1 and the fine-tuned on downstream tasks version T5. T5v1.1 and mT5 are pre-trained on the span-replacement task, which is why they are used in this section and in the previous chapter. We have chosen to use T5v1.1² for the English corpora as it is likely the most performant model for this language. It also contains small architectural improvements over T5, such as the change of activation functions. The multilingual model mT5 is used for the French corpus MEDIA to display the feasibility of this method with multilingual resources. It follows the T5 architecture and is pre-trained on the mC4 corpus. This task is heavily reliant on T5 as it is a span-replacement model. We will focus on T5 as it is the model we have used in previous experiments.

¹<https://huggingface.co/google/mt5-large>

²https://huggingface.co/google/t5-v1_1-large

Span replacement is used within our experiments. What is it, and how is it used? The span replacement objective on which T5 was trained works by replacing parts of the sentences with special tokens. The model uses this span-replaced sentence to model its output. The output of a pre-trained T5 model is a sequence of spans delimited by the special span tokens, which are numbered to allow multiple spans to be replaced at once. Using the example in Figure 6.1, the outputs of T5 would be a sequence like “<span_0> The Giants <span_1> [...]” where the span is chosen according to the span token in the input sentence. Our goal is to use this task to replace parts of labeled sentences. The replaced spans are chosen depending on what the experiment is on: either a mention or a span of context. A list of the available spans is built, and the span is chosen through uniform sampling of this list. Modeling is done with nucleus sampling where the number of highest probability words $k = 50$ and the maximum cumulative probabilities $p = 0.95$. These numbers were chosen arbitrarily following common practices, with a human check to make sure the resulting sentences were not unusable. Twenty-five sequences are produced, from which five are kept via uniform sampling. This allows filtering to happen before the final sampling. The spans are extracted from the sequences of spans, and they replace the special span token in the initial sentence. The span is split using white space to know how many tags must be inserted into the sequence of tags. These additional tags are inserted in the right slot of the sequence of tags. This whole process is summarized in Figure 6.1 where a mention is replaced using T5.

Experiments have been run in three different configurations. The first configuration is mention replacement, a direct analogy to the pattern-filling generation method. The second configuration is when the context around mentions is replaced by the language model. For context replacement, the spans are the series of tokens between mentions. We have tried this method as the possibility of generating new patterns would be removing the limiting factor of pattern-filling generation, which is obtaining new patterns. The third configuration is when the two methods can be applied. The choice of the method is made via sampling.

For this experiment, the methodology from chapter 3 was used. Five seeds were used to obtain an indicator of the randomness of the process. The taggers are trained on data augmented with the tested generation methods. The taggers trained are the BERT + MLP taggers, with the pre-trained BERT models being BERT-large-cased³ for CoNLL and BioBERT-base-cased⁴ for I2B2. The generated set with this method contains 20,000 examples. We will take a look at the F_1 scores of the taggers trained on the augmented sets produced with the methods presented.

In this first experiment, our main objective was to create augmented sets using mention replacement, context replacement, and a set containing examples from both methods. The F_1 scores for CoNLL, I2B2 and MEDIA can be observed respectively in Table 6.1, Table 6.2 and Table 6.3. All three methods at all natural data sizes for CoNLL display degradation of performances from the baseline. Context replacement degrades less than the other settings, with the method using both replacements being second. The mention replacement method provides small

³<https://huggingface.co/bert-large-cased>

⁴<https://huggingface.co/dmis-lab/biobert-base-cased-v1.1>

Size	50	100	250	500	1000
Baseline	57.8 \pm 2.9	70.1 \pm 5.6	81.2 \pm 1.8	85.3 \pm 1.1	88.0 \pm 0.3
T5 20k mention	50.1 \pm 4.0	55.7 \pm 2.2	64.9 \pm 2.5	70.2 \pm 1.2	77.1 \pm 0.8
T5 20k context	52.0 \pm 3.5	62.3 \pm 3.2	72.6 \pm 2.2	78.8 \pm 0.6	83.6 \pm 0.5
T5 20k both	53.8 \pm 2.2	61.9 \pm 1.7	70.1 \pm 2.4	75.4 \pm 1.4	80.6 \pm 1.1

Table 6.1: CoNLL F_1 scores for taggers trained on data augmented with T5 span replacement.

Size	50	100	250	500	1000
Baseline	37.0 \pm 3.2	49.4 \pm 1.6	63.2 \pm 1.6	72.0 \pm 0.7	77.3 \pm 0.4
T5 20k mention	38.2 \pm 2.4	49.3 \pm 3.5	63.2 \pm 1.2	71.5 \pm 1.8	77.9 \pm 1.0
T5 20k context	28.8 \pm 2.4	37.9 \pm 2.0	49.9 \pm 3.0	57.9 \pm 1.8	66.8 \pm 1.3
T5 20k both	37.7 \pm 1.7	44.7 \pm 1.3	54.3 \pm 1.4	61.6 \pm 1.2	67.7 \pm 0.5

Table 6.2: I2B2 F_1 scores for taggers trained on data augmented with T5 span replacement.

improvements for I2B2 at size 1000 with 77.9 against 77.3 and larger improvements at size 50 with 38.2 against 37.0, otherwise being similar to the baseline results. The method replacing both mention and context achieves better performances at size 50 with 37.7 against 37.0. Otherwise, the other methods provide degradation at all other size points, with the method replacing both context and mention also being in the middle. For MEDIA, span-replacement degrades the performances of the baseline for all data points except two. The context replacement achieves improvements at sizes 50 and 100, with 47.9 and 55.7 against 44.3 and 53.9, respectively. At higher data sizes, the method obtains negative results with degradation of up to two points. The other methods obtain lower results. We did not expect such negative results with regard to the performances obtained with tri-training. This shows that the filtering of tri-training had some impact on the examples included in the training set. We must investigate the generated set to understand what has happened in the generation process to degrade the training of the downstream models to this extent.

6.1.2 . Data Analysis

The results of this first experiment are negative. Analyzing the generated text is the first step to understanding and fixing the potential sources of performance degradation. A first look at the data gives us a hint of why the results are as they are. Many of the mentions were replaced using numbers, punctuation, or special characters. Examples of the generated mentions can be seen in Table 6.4. The examples contain a variety of cherry-picked mentions from CoNLL. Some of these mentions are correct, but others display some problems. Codes such as "500 12 1/2 63" appear within the mentions, and other uninformative parts of speech such as "the u" or ", the" were introduced. Codes can potentially be sorted with regular expression, but the other types of errors cannot.

Size	50	100	250	500	1000
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
T5 20k mention	35.2±1.2	42.9±1.4	55.6±2.1	65.6±0.9	73.6±0.8
T5 20k context	47.9±2.0	55.7±1.1	65.5±1.3	72.4±0.8	78.0±0.8
T5 20k both	41.3±2.6	48.9±1.5	60.9±1.5	69.4±1.0	75.5±0.9

Table 6.3: MEDIA F_1 scores for taggers trained on data augmented with T5 span replacement.

Our first method is to try to evaluate how much of the augmented data contained such mentions. A simple regex matching everything except words is used to find such mentions. An example of the distribution of these problematic examples can be seen in Table 6.5. A first look at these results on seed one at size 50 shows that there is a significant quantity of noisy mentions. The interesting fact is that these mentions are not distributed equally between the different concepts. The MISC concept contains far fewer problematic mentions than the other concepts, with 16 out of the 1433 mentions being problematic. The number of these mentions can vary from simple to double for the other concepts. This ranges from 196 out of the 2030 PER mentions to 410 out of the 2141 LOC mentions. Looking at the problem in one example run is not enough to reach any conclusion.

The results across the multiple experiments of these matches can be seen in Table 6.6. A couple of mentions are matched for I2B2 at size 1000 on two seeds, but it was so few that it did not get represented in the average. We can also see that T5 does not produce the same quantity of these peculiar mentions for CoNLL and I2B2. For CoNLL, the quantity is around 15% of the unique mentions generated. For I2B2, this quantity is closer to 7%. This quantity varies with the size of S_n . This can be explained by the number of natural mentions being higher in these sets. The mention replacement also produces fewer of these mentions relative to the number of mentions generated.

ORG	LOC
over terrorism 500 12 1/2 63 Commonwealth summit	Polish Federation 26 August-29 1)
PER	MISC
Mario Gardin and the Englishman the u	traders said. The a city gov , the

Table 6.4: Examples of mentions generated by T5 for the CoNLL corpus.

6.1.3 . Filtering and prompting

We believe that this method can be improved to avoid the pitfalls of the first experiment. Solutions can be put in place to feed data that contains more and better

	ORG	LOC	PER	MISC
Baseline 50 seed 1	25	25	21	15
Baseline 50 seed 1 sp	0	0	0	0
T5 20k mention 50 seed 1	1593	2141	2030	1433
T5 20k mention 50 seed 1 sp	359	410	196	16

Table 6.5: Analysis of unique mentions in seed one at size 50 of CoNLL. These mentions can be repeated within the dataset. Sp lines represent the amount of mentions that are filtered.

Size	50	100	250	500	1000
CoNLL					
T5 mentions	15.3±3.2	14.2±1.3	13.8±1.4	13.6±1.1	13.4±0.7
T5 both	16.4±3.2	15.4±1.7	14.6±1.4	14.0±0.9	13.6±0.8
I2B2					
T5 mentions	6.5±0.7	6.6±1.3	6.6±0.7	6.1±0.5	6.0±0.5
T5 both	7.5±0.9	7.4±1.4	7.5±0.8	6.9±0.7	6.6±0.4

Table 6.6: Percentage of mentions that can be removed using a simple filter regex. Context is not shown as it cannot be matched.

information, thus enabling better generation. Filtering can also be put in place to remove obvious bad examples from the generated sentences. The experiments with semi-supervised learning have shown that these methods of generation can function as-is, but the models sort out the worse examples.

Two experiments were done. The first experiment introduces filtering on the generated spans. In the previous experiment with T5, 25 spans were generated, but five were kept. Filtering is introduced on those 25 generated spans. We apply the regular expression used in the data analysis section to remove the obviously out-of-scope spans. The second experiment adds prompting on top of the filtering. It is possible that the context provided for text generation was not sufficient. For this reason, we sample and concatenate a sentence from the natural data in front of the span-replaced sentence. The experiments were only run on the use of both context and mention replacements because it was the average method, and the performances of the methods varied greatly between the corpora, making it impossible to outline the best method.

Size	50	100	250	500	1000
Baseline	57.8±2.9	70.1±5.6	81.2±1.8	85.3±1.1	88.0±0.3
T5 20k filtered	57.1±1.9	62.1±2.3	70.2±1.7	76.2±1.3	81.1±0.6
T5 20k filtered prompt.	55.4±2.9	64.1±2.0	73.8±2.7	78.9±2.5	81.9±0.3

Table 6.7: F_1 score of filtered CoNLL. C stands for context replacement, and m stands for mention replacement.

Size	50	100	250	500	1000
Baseline	37.0±3.2	49.4±1.6	63.2±1.6	72.0±0.7	77.3±0.4
T5 20k filtered	37.8±1.6	43.3±2.5	54.3±1.5	61.7±1.4	68.5±0.6
T5 20k filtered prompt.	38.8±2.7	45.0±2.6	52.3±1.5	67.0±5.4	72.1±4.8

Table 6.8: F_1 score of filtered I2B2. C stands for context replacement, and m stands for mention replacement.

Size	50	100	250	500	1000
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
T5 20k filtered	39.8±1.7	47.5±1.0	59.7±1.7	68.1±1.2	74.6±0.9
T5 20k filtered prompt.	40.6±1.6	47.2±1.0	59.3±2.0	67.9±1.8	74.2±1.0

Table 6.9: F_1 score of filtered MEDIA. C stands for context replacement, and m stands for mention replacement.

The results from CoNLL, I2B2 and MEDIA can respectively be found in Tables 6.7, Table 6.8 and Table 6.9. The performances of the taggers trained with these methods are still below the baseline for the majority of the sizes of subsets used. Improvements over the baseline are seen for I2B2 at size 50 for both prompting and not prompting, with respectively 38.8 and 37.8 against 37.0. Prompting obtains better performances on all experiments from CoNLL and I2B2 except I2B2 at size 250. The new results decrease the performances for MEDIA. While this method has not been fruitful, it is possible that prompting could be used to improve the previous generative tri-training experiments. This would require testing that we will not be able to provide in this work. The additional filtering and prompting were not enough to bring positive performances to this method. Using the baseline models to filter the generated data as it is done for tri-training would be a potential solution to this hurdle. Unfortunately, this will have to be tested in future works.

6.2 . Combining pattern-filling and tri-training

In the previous chapters, generation techniques were presented and studied. In the previous section, two of the generation methods introduced for semi-supervised learning were used in a pattern-based fashion. Simpler ways of combining the generation methods exist. Generated sets of data are produced by the generation methods. These methods can be used in parallel to produce generated data which can be concatenated to obtain a final set of generated data that is larger and contains more various examples. These methods can also be used in sequence to iteratively improve on the previous augmented set. This section studies the impact of such uses of the generation methods.

6.2.1 . Concatenation of Pattern-filling and Tri-training data

The experiment we have conducted for parallel use of pattern-filling and tri-training is to concatenate the datasets obtained with pattern-filling generation and generative tri-training. With this experiment, our objective is to show that adding data from different sources of generation can improve the end result. In this setting, the taggers are trained on the natural subset S_n with the generated set from pattern-filling generation G_n^p and the generated set from generative tri-training G_n^t . Fixing the total quantity of data would not have been possible as the number of sentences in the outputs of generative tri-training varies from seed to seed due to the requirement that all three models must output the same tagging scheme. Performance indicators were also computed using these fully generated sets for training. Changing the sizes of G_n^p and G_n^t would add other biases that would be difficult to take into account. As such, we know that the training set is larger than in the previous experiment and expect an increase in performance. These experiments were only conducted with two generation methods from tri-training: sentence follow-up generation and mention replacement. These methods were the overall best generation methods from generative tri-training.

Size	50	100	250	500	1000
Baseline	57.8±2.9	70.1±5.6	81.2±1.8	85.3±1.1	88.0±0.3
PatGen 20k	66.8±3.3	71.7±2.8	80.3±1.6	83.6±1.8	86.7±0.7
Follow-up Outputs	64.2±2.1	73.6±3.0	83.9±0.9	87.3±0.6	89.0±0.7
Follow-up + PatGen	64.6±3.1	74.1±1.9	83.1±0.9	86.4±1.5	88.4±0.3
Mention rep. Outputs	65.2±1.6	72.9±3.2	82.6±1.6	86.3±0.9	88.4±0.4
Mention rep. + PatGen	64.7±1.2	74.1±1.9	81.4±2.1	85.8±1.2	86.8±0.5

Table 6.10: F_1 score of $S_n \cup G_n^p \cup G_n^t$ for CoNLL. Follow-up and mention rep. are the tri-training generation method from which the data is used. PatGen refers to the pattern-filling generation method in a uniform setting.

Size	50	100	250	500	1000
Baseline	37.0±3.2	49.4±1.6	63.2±1.6	72.0±0.7	77.3±0.4
PatGen 20k	45.7±4.4	56.6±2.2	67.4±0.2	73.8±0.9	79.1±0.8
Follow-up Outputs	41.4±2.6	51.1±1.6	67.8±1.3	75.4±1.1	79.8±0.7
Follow-up + PatGen	42.7±3.3	53.0±2.7	68.8±0.7	75.9±1.2	80.7±0.5
Mention rep. Outputs	43.0±2.8	56.1±1.2	70.2±1.3	75.7±1.0	79.8±0.6
Mention rep. + PatGen	42.9±2.2	56.8±3.1	70.6±1.4	75.3±0.5	80.2±0.4

Table 6.11: F_1 score of $S_n \cup G_n^p \cup G_n^t$ for I2B2. Follow-up and mention rep. are the tri-training generation method from which the data is used. PatGen refers to the pattern-filling generation method in a uniform setting.

The results of this experiment can be found in Table 6.10 for CoNLL, in Table 6.11 for I2B2, and in Table 6.12 for MEDIA. The original pattern-filling genera-

Size	50	100	250	500	1000
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
PatGen 20k	50.7 ±3.8	57.5±1.5	67.1±2.2	74.0±0.9	78.0±0.3
Follow-up Outputs	46.4±2.6	54.6±2.8	67.2±1.2	74.9 ±0.9	80.3 ±0.4
Follow-up + Patgen	49.5±2.4	57.6±1.0	67.7±2.4	74.4±0.8	79.0±0.5
Mention rep. Outputs	46.9±3.0	56.6±1.4	67.6±1.8	74.4±0.7	80.1±0.5
Mention rep. + Patgen	50.0±3.3	58.6 ±1.6	68.4 ±1.2	74.8±1.0	79.1±0.6

Table 6.12: F_1 score of $S_n \cup G_n^p \cup G_n^t$ for MEDIA. Follow-up and mention rep. are the tri-training generation method from which the data is used. PatGen refers to the pattern-filling generation method in a uniform setting.

tion method is still the best method at size 50, with a 66.8 F_1 score. The models trained on the data from tri-training with follow-up from previous experiments are still the best methods for sizes 250 to 1000, with 83.9, 87.3, and 89.0 scores. Follow-up with pattern-filling generation improves the scores of the baseline at each tested size for both corpora and is the best method alongside with mention replacement with pattern generation at size 100 with a 74.1 score. Mention replacement with pattern-filling generation improves the scores of the baseline for subsets of sizes under 500 and degrades performances at size 1000 for CoNLL. The performance of pattern-filling generation for CoNLL is degraded for subsets of size 250 and more. This impact is felt more strongly for mention replacement as the size 1000 shows performance degradation. For I2B2, the performances of the taggers trained are lower than pattern-generation at size 50 for both methods and are otherwise higher, or equivalent to the performances of pattern-generation and tri-training generated data with the respective generation method. The best method at sizes 100 and 250 is the new mention replacement with the pattern-filling generation with 56.8 against 56.6 and 56.1 for the pattern-filling generation method and the mention replacement method, and 70.6 against 67.4 for pattern-filling and 70.2 for mention replacement. Mention replacement with pattern-filling also improves performances over the initial methods at size 1000 with 80.2 against 79.1 for pattern-filling and 79.8 for mention replacement. The best method at sizes 500 and 1000 is the follow-up with pattern-filling generation method, with 75.9 against 73.8 and 75.4 for the pattern-filling generation method and the follow-up method, and 80.7 against 79.1 for pattern-filling and 79.8 for follow-up generation. This method also outperforms both the initial pattern-filling method and follow-up generation on size 250 with 68.8 against 67.4 and 67.8, respectively. For MEDIA, adding pattern-generated data to the tri-training-generated data improves the results at sizes up to 250. Pattern-filling generation remains the best method at size 50 with 50.7 against 49.5 for the addition of follow-up data and 50.0 for the addition of the mention replacement data. The best method for 100 and 250 is mention replacement with pattern-filling generation with 58.6 against 57.5 and 58.6 for the pattern-filling generation method and the mention replacement method and 68.5 against 67.1 and 67.6, respectively. Mention replacement with pattern generation is also better than mention

replacement alone at size 500 with 74.8 against 74.4. Follow-up with pattern-filling generation is worse than follow-up alone at sizes 500 and 1000.

The addition of both generated sets has an averaging effect on the performance of the taggers trained for CoNLL. For I2B2 and MEDIA, adding the data from pattern-filling generation with mention replacement is better at sizes 100 and 250. It is only better at size 100 for CoNLL. Adding pattern-generated data to the mention replacement improves performances at size 100 consistently. Adding more data seems to not be the answer to tagger improvement with the quantities we can now generate overall. This raises the question of how to create a better dataset from multiple generation techniques that have been run in parallel. This question is, however, not answered in this work. In the next section, we will study the effect of applying the generation techniques sequentially.

6.2.2 . Applying pattern-filling on semi-supervised data

This section presents the application of the generation methods in sequence. Our hypothesis is that since the generation methods improve the performances of taggers through the augmentation of the training data, augmenting an augmented set can potentially improve performances further. We, however, acknowledge that this could also be degenerative and produce degradation to the performances. That is why this process must be investigated. The experiments presented contain the pattern-filling generation method applied to data augmented with generative tri-training.

In this experiment, pattern-filling generation is applied to the augmented set of data from generative tri-training $S_n \cup G_n^t$. The experiments were conducted using the data from the follow-up sentence generation method. It is one of the two best methods of generative tri-training. The other method is mention replacement. As mention replacement is similar to the pattern-filling generation method in nature, we have chosen not to use it in this set of experiments. The methodology from chapter 3 was applied. Twenty thousand examples were generated using pattern-filling generation. For CoNLL, for some seeds, there are slightly less than 20,000 examples that were generated due to patterns or mentions containing problematic sequences of characters for the regular expression library that we have chosen to ignore and have not implemented replacement mechanisms.

Size	50	100	250	500	1000
Baseline	57.8±2.9	70.1±5.6	81.2±1.8	85.3±1.1	88.0±0.3
PatGen 20k	66.8±3.3	71.7±2.8	80.3±1.6	83.6±1.8	86.7±0.7
Follow-up Outputs	64.2±2.1	73.6±3.0	83.9±0.9	87.3±0.6	89.0±0.7
PatGen on Follow-up	63.2±3.7	72.8±4.0	80.8±1.7	85.4±0.8	87.9±0.3

Table 6.13: F_1 score of BERT + MLP taggers trained on an augmented set of data generated with pattern-filling generation on an augmented set produced by generative tri-training for CoNLL.

The F_1 scores obtained with this method for CoNLL, I2B2 and MEDIA can be observed respectively in Table 6.13, Table 6.14 and Table 6.15. Performances of the taggers have been degraded from the taggers trained on the generative

Size	50	100	250	500	1000
Baseline	37.0±3.2	49.4±1.6	63.2±1.6	72.0±0.7	77.3±0.4
PatGen 20k	45.7±4.4	56.6±2.2	67.4±0.2	73.8±0.9	79.1±0.8
Follow-up Outputs	41.4±2.6	51.1±1.6	67.8±1.3	75.4±1.1	79.8±0.7
PatGen on Follow-up	39.8±3.5	50.3±2.1	65.9±1.2	74.3±1.4	79.5±0.2

Table 6.14: F_1 score of BERT + MLP taggers trained on an augmented set of data generated with pattern-filling generation on an augmented set produced by generative tri-training for I2B2.

Size	50	100	250	500	1000
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
PatGen 20k	50.7±3.8	57.5±1.5	67.1±2.2	74.0±0.9	78.0±0.3
Follow-up Outputs	46.4±2.6	54.6±2.8	67.2±1.2	74.9±0.9	80.3±0.4
PatGen on Follow-up	47.8±1.9	55.7±1.9	66.9±1.6	73.6±1.3	79.1±0.6

Table 6.15: F_1 score of BERT + MLP taggers trained on an augmented set of data generated with pattern-filling generation on an augmented set produced by generative tri-training for MEDIA.

tri-training data for CoNLL and I2B2. At 250 sentences and 1000 sentences, the performances of the taggers drop below the baseline results for CoNLL. These consequences were expected for CoNLL, as pattern-filling generation degraded the baseline performances overall. The results of this experiment are not as negative for I2B2 as the augmented set is still obtaining better performances than the baseline. For MEDIA, the F_1 scores at size 50 and 100 are higher than generative tri-training with 47.8 and 55.7 against 46.4 and 54.6, but it is still below the pattern-filling generation results that are 50.7 and 57.5. The results drop below the baseline at size 500 with 73.6 against 74.3.

Pattern-filling generation cannot be used to improve augmented data. This method performed well at low quantities of data but is now hindered by the additional data provided by generative tri-training. This shows that the filtering mechanism that generative tri-training uses could still be improved. This improvement to the filtering mechanism will most likely come from improving the initial performances of the ensemble of taggers. Pattern-filling generation could serve as a way of improving the initial quality of the model, as it has shown its potential for all corpora at low resources. This experiment will, however, not be treated in this work due to time and computing constraints.

We did not have the time to do any experiment where the generative tri-training algorithm is applied to a set of data augmented with pattern-filling generation. These experiments would likely result in better performances for multiple reasons: wrong annotations are not spread by the pattern-filling generation method, and tri-training depends on initial tagger performances, which would be increased by pattern-filling in most cases. This remains to be tested.

6.3 . Conclusion

In this chapter, the combination of pattern-based generation and language modeling with semi-supervised learning was studied. Our hypothesis was that it would be able to bring the best of both methods and improve them by making them work together. Using the span-replacement methods presented with tri-training in a pattern-filling manner has proven unfruitful. This has highlighted the requirement of the filtering brought by tri-training. A future experiment with these methods is needed where the baseline would be used in a similar fashion as the ensemble is used to filter the generated examples. The generation methods from the previous chapters were also combined in a more pragmatic manner. Running the generation methods in parallel has shown an averaging effect on the performances of the taggers. Sequential use of the methods has not shown promising results. However, this is mainly due to pattern generation being applied after tri-training generation for CoNLL. This method did not work well with natural CoNLL data, and it also did not work well with augmented data. Reversing the order of the generation methods is a possible improvement, especially for I2B2, because both methods bring improvements. Combining the methods has not been a successful endeavor. Adding more data is not the issue at this point. Experimentation needs to be done on how to filter and extract the most useful examples for training from these synthetic sets. This thesis does not provide these experiments. It is one of the axes of improvement of these experiments for future works.

7 - Conclusions

In this thesis, we studied generation techniques with the purpose of generating or augmenting datasets of labeled sequences for the sequence labeling task.

The work from this thesis is aimed at improving the pipeline of the AIDA project on which it was financed. One of the objectives of the application of the methods studied was to augment data provided by the AIDA project. The data was not available in due time to be used and presented in this thesis. The corpora chosen to apply our methods on in the experiments cover many of the aspects of the AIDA corpus, such as the language with MEDIA or the heavy influence of the domain with I2B2.

The contributions made during this thesis are first described. This process has raised questions that we will discuss. The experiments we have done also procured findings. These studies had blind spots that still need to be explored.

7.1 . Contributions

We have made contributions in the pursuit of improving performance for the sequence labeling task.

Pattern-filling generation: We have proposed our implementation¹ of the pattern-filling generation method. We have also provided an analysis of the performance of the pattern-filling generation method [Boulanger, 2020]. This analysis covers multiple settings, which include a generation setting and augmentation setting, and the study of the number of patterns necessary to apply this generation method in a real-life setting. We also propose the study of the method used with recent models.

Tri-training and language modeling: The second approach for labeled text generation presented in this thesis is the use of language modeling coupled with a semi-supervised learning method, tri-training. Our first contribution with this approach is a study of the behavior of the tri-training algorithm with data generated with language models [Boulanger et al., 2022] and its implementation². Our second contribution to this approach is generative tri-training, a modified tri-training algorithm including the language modeling text generation process, which is currently in the publication process. Multiple methods of generation were used in this contribution which made use of causal language modeling and of span-replacement. The implementation will be available in the same repository as the other tri-training implementation at the time of publication. Our goal is to produce labeled data, and as such, our final contribution to this method is to test the quality of the labeled data generated by the generative tri-training process.

¹<https://github.com/HugoBoulanger/Pattern-Filling-Generation>

²<https://github.com/HugoBoulanger/Tritraining-Gen>

Combining language modeling and patterns: Our third approach is to combine the approaches of the previous contributions. To improve upon the lack of additional mention or patterns in the pattern-filling generation method, we proposed span-replacement modeling as a means to produce new labeled sentences with a process similar to pattern-filling generation, which did not produce positive results. This contribution uses the span-replacement methods introduced with generative tri-training while keeping the tagging scheme of the natural sentences used. As each generation method introduced in previous contributions improved upon the baseline, we proposed using them in parallel and in sequence.

7.2 . Discussions

The experiments in this thesis have raised discussions on various subjects. These discussions are focused on the low-resource setting. The first idea would be that this setting would not consume much carbon. We will, however, see that it is not that straightforward. The second subject of discussion is the impact of the pre-trained language models.

The question of the carbon cost of the thesis was not a question that we had considered at the inception of the thesis. As such, we have not used any carbon cost measurement tool. We will do our maximum to measure and mitigate carbon emissions considering the minimum threshold of the carbon emitted that will be presented and the importance of energy efficiency in the current context of the world. We can, however, give a minimum threshold for this thesis based on computing resource consumption that was clocked in by the data centers on which they were run. Over the year 2022, we have consumed about 30,000 GPU hours of Nvidia V100 32Go at Jean Zay, mostly tri-training related. We consumed 5,000 GPU hours at the end of 2021, mostly VAE-related. Previous work was done mostly on Lab-IA or directly on a laptop on which the consumptions were not recorded. These 35,000 hours of GPU usage amount to $35,000 \times 0.482 = 16,870$ kWh or 956 kgCO₂e using France's 2021 emission factor. This is equivalent to 3000 km of planes, which is a third of the total plane usage going to NAACL. This duality of working in a low-resource setting is something to have in mind when willing to evaluate low-resource methods. Due to the reduction of data required to simulate the low-resource setting, tempering sampling bias becomes an even more important part of the experiments. Thus experiments need to be multiplied, which results in high computing time and a high carbon cost. Code optimization will play a part in reducing the carbon footprint, but the biggest impact is probably model optimization. Using lighter models is something we have tried and will continue to pursue our experiments by limiting ourselves to the 1B parameters models for inference and BERT-large for training. Using even lighter models such as ALBERT [Lan et al., 2020] and their derivatives was on our radar, but we did not have the time for additional experiments. But the multiplicity of training, especially with methods such as tri-training, can be addressed. A hypothetical way of addressing the issue would be to use Adapters [Houlsby et al., 2019] instead of fine-tuning the full models. This requires further investigation.

The other topic of discussion for the low-resource setting is the use of pre-trained large language models, not for their ecological impact but due to the sheer amount of data used to train them. We have shown that our methods work using pre-trained models as the centerpiece of our taggers. The pattern-filling generation method worked when used with a tagger that used less pre-trained information, but not none, as the embeddings were pre-trained on Wikipedia. The effect of the pre-training data for the taggers can be seen in the performances obtained on the full corpora. The effect of the models used for generation is much harder to quantify. First, this would require to have done experiments with different models for the same corpora. This was not done in this thesis, but the same models, GPT-2 and T5, were used for two corpora, CoNLL and I2B2. They have shown different behaviors, with GPT-2 producing better results on CoNLL and T5 producing better results on I2B2. Is this a result of the downstream task of the data used to train the large language models, or both? This work doesn't aim to tackle such a question, but it is clear that any data found in any part of the training process can have an impact on the end result.

7.3 . Summary of Findings

Pattern-filling generation: In our domain of studies and publications, rule-based approaches have led the stage to machine learning-based approaches. Language modeling using large language models has become the predominant approach to solving many problems. We believed that building synthetic labeled data with patterns and mentions could be a means to use the best of both worlds. The first contribution to this thesis is the pattern-filling generation method. We have found that pattern-filling generation was a viable option for labeled sentence augmentation when 50 or 100 sentences are available. The results at higher quantities of data vary from corpora to corpora, sometimes degrading the performances of the tagger. This raises the question of how to improve this method further, especially when more data is available.

Tri-training and language modeling: We studied the use of semi-supervised learning as a means to add tags to sentences generated with language modeling methods. Positive results were obtained for tri-training using a static set of data which validated the approach of generating data. The generated data was not completely used by the taggers during this experiment which led us to try to include the generation process during the tri-training process. By integrating the generation process in the tri-training algorithm, we aimed to use the ensemble of models trained and the pseudo-labeling mechanism as a filtering mechanism for the text generator. The generation methods used in tri-training are language modeling methods applied to sentences. Generative tri-training allows the methods to be applied to previously generated sentences. The ensemble of models trained with this method is the best-performing taggers in our arsenal at 500 and 1000 sentences. The quality of the data generated was tested, and it showed improvements over the performances of individual models trained during tri-training, and for some corpora, it even brought improvements over the ensemble.

Combining the previous techniques: The final findings relate to the combination of the techniques of generation through language modeling with the information available in patterns and in the available labeled natural data. Tri-training introduced mention replacement and context replacement methods which used T5 for its span replacement capacities and used the information from the tagging scheme to choose the span to be replaced but did not fit the tagging scheme to the generated data, and it was the ensemble's role to add the labels. We have tested to keep the tagging scheme from the natural data to no avail, and it degraded the performances of the baseline. The data produced requires more filtering to be useful, as the same techniques used with tri-training have shown better results. Filtering and prompting were used to improve the results, and they did, but it still degraded baseline performances. Additional techniques could be employed to improve generative tri-training. This also shows that the filtering capacity of the ensemble of models in tri-training and the iterative application of the generation methods might be a huge help, and a better understanding of the interactions of these functionalities of tri-training to replicate or improve them is a work left for the future. The generation methods presented in previous chapters were also used in parallel and in sequence. The performance of taggers decreases when trained on data from both tri-training and pattern-filling generation against the best of both methods. This shows that adding more generated data is not an improvement venue at this point, and the priority should now be to improve the quality of the generated data. Sequentially applying the generation methods has potential, but only one order of application was tested due to constraints: pattern-filling applied to tri-training data. This did not produce good results, and it is possible that the order in which the generation processes are used has importance for tri-training.

7.4 . Perspectives

This work has proposed methods to generate additional tagged sentences for the sequence labeling task. We have mostly explored pattern-based generation and semi-supervised learning used for the addition of tags over language-modeled sentences. The methods explored have not permitted taggers to reach full-sized corpus performance when trained on a small set of data with augmentation, but they have improved the performances of the taggers at low quantities of data. This improvement can allow these taggers to be used as a first step in an application setting or a helping tool for labeling, but they definitely need further improvements to be used as is. These methods have opened trails of potential experiments and applications that we have not explored.

Pattern-filling generation can degrade performances at higher quantities of data depending on the corpus but also depending on the tagger architecture. Further exploration of these limitations is needed to obtain a more reliable method of generation at higher quantities of data.

The problem with tri-training is computation, as it requires multiple training sessions and multiplies the size of the tagger as it is an ensemble. Exploring architectural improvements such as parameter sharing could help reduce this com-

putational issue and might allow more taggers to be included in the ensemble. Tri-training has also raised interrogations upon the filtering operated by the ensemble, which allows models trained on the augmented data to obtain the best performances. If the performance of this mechanism can be replicated without the need for the ensemble of taggers, it could be a good tool to improve other generation methods and allow iterative modifications with the generation methods.

Using the span-replacement methods introduced with tri-training in a standalone setting has not provided improvements. We have explored some methods, such as filtering and prompting, which have improved the results but have not redeemed the approach. These improvements should be tested in tri-training as they could also improve this process. Finding a cheaper and more effective filtering mechanism than tri-training is a goal that needs to be pursued as both a means to improve existing working methods and allow other methods, such as the T5 augmentation method, to work outside of tri-training.

Applying the generation methods consecutively did not bring the expected improvements. Running pattern-filling generation on data augmented with tri-training did not provide improvements over using only the tri-trained data. The potential of applying first the pattern-filling generation method is greater as the better the taggers from within the tri-training process, the better their filtering and annotations of the generated text.

This thesis aimed to produce data for the AIDA project. While the data was not available in a quality sufficient for application during the thesis, it is one of the tasks that will be done after the redaction of this thesis.

In this thesis, we aimed to develop techniques to augment data for sequence labeling. We have shown that pattern-filling generation improves tagger performances for all the tested corpora at the smallest quantities of data, whereas the performance can degrade on a corpus basis at higher quantities of data. Using large language models with tri-training has proved to be a more reliable solution, with the caveat of the higher computation cost. The use of large language models to replace parts of labeled sentences has not produced positive results. As a whole, we have shown that generating additional data can improve the performance of models on the downstream task. It is, however, not enough to reach performances of full-sized datasets with only 1000 or lower natural examples as of the current state of our research. All of the code used for experimentation is or will be available on this repository: <https://github.com/HugoBoulangier/NERAug>.

Bibliography

- [Alemi et al., 2018] Alemi, A., Poole, B., Fischer, I., Dillon, J., Saurous, R. A., and Murphy, K. (2018). Fixing a broken elbo. In *International Conference on Machine Learning*, pages 159–168. PMLR.
- [Béchet and Raymond, 2019] Béchet, F. and Raymond, C. (2019). Benchmarking Benchmarks: Introducing New Automatic Indicators for Benchmarking Spoken Language Understanding Corpora. In Kubin, G. and Kacic, Z., editors, *Inter-speech 2019, 20th Annual Conference of the International Speech Communication Association, Graz, Austria, 15-19 September 2019*, pages 4145–4149. ISCA.
- [Bengio et al., 2000] Bengio, Y., Ducharme, R., and Vincent, P. (2000). A neural probabilistic language model. *Advances in neural information processing systems*, 13.
- [Bengio et al., 2013] Bengio, Y., Léonard, N., and Courville, A. (2013). Estimating or propagating gradients through stochastic neurons for conditional computation. *arXiv e-prints*, pages arXiv–1308.
- [Blum and Mitchell, 1998] Blum, A. and Mitchell, T. (1998). Combining labeled and unlabeled data with co-training. In *Proceedings of the eleventh annual conference on Computational learning theory*, pages 92–100.
- [Bojanowski et al., 2017] Bojanowski, P., Grave, E., Joulin, A., and Mikolov, T. (2017). Enriching word vectors with subword information. *Transactions of the association for computational linguistics*, 5:135–146.
- [Bonneau-Maynard et al., 2005] Bonneau-Maynard, H., Rosset, S., Ayache, C., Kuhn, A., and Mostefa, D. (2005). Semantic annotation of the french media dialog corpus. In *Ninth European Conference on Speech Communication and Technology*.
- [Bordes et al., 2017] Bordes, A., Boureau, Y.-L., and Weston, J. (2017). Learning end-to-end goal-oriented dialog. In *International Conference on Learning Representations*.
- [Boulanger, 2020] Boulanger, H. (2020). Évaluation systématique d’une méthode commune de génération. In *6e conférence conjointe Journées d’Études sur la Parole (JEP, 33e édition)*, pages 43–56. ATALA; AFCEP.
- [Boulanger et al., 2022] Boulanger, H., Lavergne, T., and Rosset, S. (2022). Generating unlabelled data for a tri-training approach in a low resourced NER task. In *Proceedings of the Third Workshop on Deep Learning for Low-Resource Natural Language Processing*, pages 30–37, Hybrid. Association for Computational Linguistics.

- [Brown et al., 2020] Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J. D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., et al. (2020). Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.
- [Cattan et al., 2022] Cattan, O., Ghannay, S., Servan, C., and Rosset, S. (2022). Benchmarking transformers-based models on french spoken language understanding tasks. In *INTERSPEECH 2022*.
- [Chen et al., 2019a] Chen, Q., Zhuo, Z., and Wang, W. (2019a). Bert for joint intent classification and slot filling. *arXiv e-prints*, pages arXiv-1902.
- [Chen et al., 2019b] Chen, Y., Zhou, C., Li, T., Wu, H., Zhao, X., Ye, K., and Liao, J. (2019b). Named entity recognition from chinese adverse drug event reports with lexical feature based bilstm-crf and tri-training. *Journal of Biomedical Informatics*, 96:103252–103252.
- [Cheng et al., 2016] Cheng, J., Dong, L., and Lapata, M. (2016). Long short-term memory-networks for machine reading. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 551–561.
- [Child et al., 2019] Child, R., Gray, S., Radford, A., and Sutskever, I. (2019). Generating long sequences with sparse transformers. *arXiv e-prints*, pages arXiv-1904.
- [Chiu and Nichols, 2016] Chiu, J. P. and Nichols, E. (2016). Named entity recognition with bidirectional lstm-cnns. *Transactions of the association for computational linguistics*, 4:357–370.
- [Cho et al., 2014a] Cho, K., van Merriënboer, B., Bahdanau, D., and Bengio, Y. (2014a). On the properties of neural machine translation: Encoder–decoder approaches. In *Proceedings of SSST-8, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation*, pages 103–111.
- [Cho et al., 2014b] Cho, K., van Merriënboer, B., Gülçehre, Ç., Bahdanau, D., Bougares, F., Schwenk, H., and Bengio, Y. (2014b). Learning phrase representations using rnn encoder-decoder for statistical machine translation. In *EMNLP*.
- [Chowdhery et al., 2022] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., Barham, P., Chung, H. W., Sutton, C., Gehrmann, S., et al. (2022). Palm: Scaling language modeling with pathways. *arXiv preprint arXiv:2204.02311*.
- [Conneau and Lample, 2019] Conneau, A. and Lample, G. (2019). Cross-lingual language model pretraining. *Advances in neural information processing systems*, 32.
- [Coucke et al., 2018] Coucke, A., Saade, A., Ball, A., Bluche, T., Caulier, A., Leroy, D., Doumouro, C., Gisselbrecht, T., Caltagirone, F., Lavril, T., et al. (2018). Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*.

- [Dahl et al., 1994] Dahl, D. A., Bates, M., Brown, M., Fisher, W., Hunicke-Smith, K., Pallett, D., Pao, C., Rudnicky, A., and Shriberg, E. (1994). Expanding the scope of the ATIS task: The ATIS-3 corpus. In *Proceedings of the workshop on Human Language Technology*, pages 43–48. Association for Computational Linguistics.
- [Dai et al., 2019] Dai, Z., Yang, Z., Yang, Y., Carbonell, J. G., Le, Q., and Salakhutdinov, R. (2019). Transformer-xl: Attentive language models beyond a fixed-length context. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2978–2988.
- [Dauphin et al., 2017] Dauphin, Y. N., Fan, A., Auli, M., and Grangier, D. (2017). Language modeling with gated convolutional networks. In *International conference on machine learning*, pages 933–941. PMLR.
- [Davis and Arel, 2013] Davis, A. and Arel, I. (2013). Low-rank approximations for conditional feedforward computation in deep neural networks. *arXiv e-prints*, pages arXiv–1312.
- [Devlin et al., 2018] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.
- [Ding et al., 2020] Ding, B., Liu, L., Bing, L., Kruengkrai, C., Nguyen, T. H., Joty, S., Si, L., and Miao, C. (2020). Daga: Data augmentation with a generation approach for low-resource tagging tasks. *arXiv preprint arXiv:2011.01549*.
- [Dopierre et al., 2021] Dopierre, T., Gravier, C., and Logerais, W. (2021). Protagument: Intent detection meta-learning through unsupervised diverse paraphrasing. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2454–2466.
- [Fan et al., 2018] Fan, A., Lewis, M., and Dauphin, Y. (2018). Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898.
- [Fedus et al., 2022] Fedus, W., Zoph, B., and Shazeer, N. (2022). Switch transformers: Scaling to trillion parameter models with simple and efficient sparsity. *Journal of Machine Learning Research*, 23(120):1–39.
- [GAGE, 1994] GAGE, P. (1994). A new algorithm for data compression. *The C Users Journal*, 12(2):23–38.
- [Ghannay et al., 2020] Ghannay, S., Neuraz, A., and Rosset, S. (2020). What is best for spoken language understanding: small but task-dependant embeddings or huge but out-of-domain embeddings? In *ICASSP 2020-2020 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 8114–8118. IEEE.

- [Goodfellow et al., 2014] Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A., and Bengio, Y. (2014). Generative adversarial nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N., and Weinberger, K., editors, *Advances in Neural Information Processing Systems*, volume 27. Curran Associates, Inc.
- [Halle, 1962] Halle, M. (1962). Phonology in generative grammar. *WORD*, 18(1-3):54–72.
- [Higgins et al., 2017] Higgins, I., Matthey, L., Pal, A., Burgess, C., Glorot, X., Botvinick, M., Mohamed, S., and Lerchner, A. (2017). beta-VAE: Learning basic visual concepts with a constrained variational framework. In *International Conference on Learning Representations*.
- [Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- [Holtzman et al., 2019] Holtzman, A., Buys, J., Du, L., Forbes, M., and Choi, Y. (2019). The curious case of neural text degeneration. In *International Conference on Learning Representations*.
- [Houlsby et al., 2019] Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for nlp. In *International Conference on Machine Learning*, pages 2790–2799. PMLR.
- [Johnson et al., 2016] Johnson, A. E., Pollard, T. J., Shen, L., Lehman, L.-w. H., Feng, M., Ghassemi, M., Moody, B., Szolovits, P., Anthony Celi, L., and Mark, R. G. (2016). Mimic-iii, a freely accessible critical care database. *Scientific data*, 3(1):1–9.
- [Joshi et al., 2020] Joshi, P., Santy, S., Budhiraja, A., Bali, K., and Choudhury, M. (2020). The state and fate of linguistic diversity and inclusion in the nlp world. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6282–6293.
- [Kingma and Welling, 2014] Kingma, D. P. and Welling, M. (2014). Auto-Encoding Variational Bayes. In *2nd International Conference on Learning Representations, ICLR 2014, Banff, AB, Canada, April 14-16*.
- [Kudo, 2018] Kudo, T. (2018). Subword regularization: Improving neural network translation models with multiple subword candidates. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 66–75.
- [Kudo and Richardson, 2018] Kudo, T. and Richardson, J. (2018). Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 66–71.

- [Kullback, 1968] Kullback, S. (1968). *Information theory and statistics*. New York: Dover.
- [Lafferty, 2001] Lafferty, J. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proc. 18th International Conference on Machine Learning, 2001*, pages 282–289.
- [Lan et al., 2020] Lan, Z., Chen, M., Goodman, S., Gimpel, K., Sharma, P., and Soricut, R. (2020). Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.
- [Lavergne et al., 2011] Lavergne, T., Allauzen, A., Crego, J. M., and Yvon, F. (2011). From n-gram-based to crf-based translation models. In *Proceedings of the sixth workshop on statistical machine translation*, pages 542–553.
- [Le et al., 2011] Le, H.-S., Oparin, I., Allauzen, A., Gauvain, J.-L., and Yvon, F. (2011). Structured output layer neural network language model. In *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5524–5527. IEEE.
- [Le Cun, 1989] Le Cun, Y. (1989). Generalization and network design strategies. *Connectionism in perspective*.
- [Lee et al., 2020] Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C. H., and Kang, J. (2020). Biobert: a pre-trained biomedical language representation model for biomedical text mining. *Bioinformatics*, 36(4):1234–1240.
- [Lei Ba et al., 2016] Lei Ba, J., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization. *ArXiv e-prints*, pages arXiv-1607.
- [Lewis et al., 2020] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2020). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880.
- [Li et al., 2017] Li, J., Monroe, W., Shi, T., Jean, S., Ritter, A., and Jurafsky, D. (2017). Adversarial learning for neural dialogue generation. In *Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing*, pages 2157–2169.
- [Liu et al., 2019] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019). Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.
- [Ma, 2019] Ma, E. (2019). Nlp augmentation. <https://github.com/makcedward/nlpaug>.
- [Marcus et al., 1993] Marcus, M., Santorini, B., and Marcinkiewicz, M. A. (1993). Building a large annotated corpus of english: The penn treebank. *Computational Linguistics*, 19(2):313–330.

- [Martin et al., 2020] Martin, L., Muller, B., Suárez, P. J. O., Dupont, Y., Romary, L., De La Clergerie, É. V., Seddah, D., and Sagot, B. (2020). Camembert: a tasty french language model. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7203–7219.
- [Merity et al., 2017] Merity, S., Xiong, C., Bradbury, J., and Socher, R. (2017). Pointer sentinel mixture models. In *International Conference on Learning Representations*.
- [Micallef et al., 2022] Micallef, K., Gatt, A., Tanti, M., van der Plas, L., and Borg, C. (2022). Pre-training data quality and quantity for a low-resource language: New corpus and BERT models for Maltese. In *Proceedings of the Third Workshop on Deep Learning for Low-Resource Natural Language Processing*, pages 90–101, Hybrid. Association for Computational Linguistics.
- [Mikolov et al., 2013] Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space in 1st international conference on learning representations, iclr 2013, scottsdale, arizona, usa, may 2-4, 2013. In *Workshop Track Proceedings*.
- [Nakayama, 2018] Nakayama, H. (2018). seqeval: A python framework for sequence labeling evaluation. Software available from <https://github.com/chakkiworks/seqeval>.
- [Neuraz et al., 2018] Neuraz, A., Llanos, L. C., Burgun, A., and Rosset, S. (2018). Natural language understanding for task oriented dialog in the biomedical domain in a low resources context. *CoRR*, abs/1811.09417.
- [Oshikawa et al., 2020] Oshikawa, R., Qian, J., and Wang, W. Y. (2020). A survey on natural language processing for fake news detection. In *Proceedings of the 12th Language Resources and Evaluation Conference*, pages 6086–6093.
- [Paulus et al., 2018] Paulus, R., Xiong, C., and Socher, R. (2018). A deep reinforced model for abstractive summarization. In *International Conference on Learning Representations*.
- [Pennington et al., 2014] Pennington, J., Socher, R., and Manning, C. D. (2014). Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543.
- [Peters et al., 2018] Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 2227–2237, New Orleans, Louisiana. Association for Computational Linguistics.

- [Pham et al., 2016] Pham, N.-Q., Kruszewski, G., and Boleda, G. (2016). Convolutional neural network language models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1153–1162.
- [Radford et al., 2018] Radford, A., Narasimhan, K., Salimans, T., and Sutskever, I. (2018). Improving language understanding by generative pre-training.
- [Radford et al., 2019] Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners. *OpenAI Blog*, 1(8):9.
- [Raffel et al., 2020] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research*, 21:1–67.
- [Ramshaw and Marcus, 1995] Ramshaw, L. and Marcus, M. (1995). Text chunking using transformation-based learning. In *Third Workshop on Very Large Corpora*.
- [Roca et al., 2020] Roca, S., Rosset, S., García, J., and Alesanco, Á. (2020). Evaluation of embeddings in medication domain for spanish language using joint natural language understanding. In *European Medical and Biological Engineering Conference*, pages 510–517. Springer.
- [Ruder and Plank, 2018] Ruder, S. and Plank, B. (2018). Strong baselines for neural semi-supervised learning under domain shift. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1044–1054.
- [RUMELHART, 1986] RUMELHART, D. (1986). Learning internal representations by error propagation. *Parallel Distributed Processing*, 1:318–362.
- [Sang and De Meulder, 2003] Sang, E. T. K. and De Meulder, F. (2003). Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003*, pages 142–147.
- [Scao et al., 2022] Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., Hesslow, D., Castagné, R., Luccioni, A. S., Yvon, F., Gallé, M., et al. (2022). Bloom: A 176b-parameter open-access multilingual language model. *arXiv preprint arXiv:2211.05100*.
- [Schuster and Nakajima, 2012] Schuster, M. and Nakajima, K. (2012). Japanese and korean voice search. In *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 5149–5152. IEEE.
- [Schwenk and Gauvain, 2005] Schwenk, H. and Gauvain, J.-L. (2005). Training neural network language models on very large corpora. In *Proceedings of human language technology conference and conference on empirical methods in natural language processing*, pages 201–208.

- [Sennrich et al., 2016a] Sennrich, R., Haddow, B., and Birch, A. (2016a). Improving neural machine translation models with monolingual data. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 86–96.
- [Sennrich et al., 2016b] Sennrich, R., Haddow, B., and Birch, A. (2016b). Neural machine translation of rare words with subword units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725.
- [Shazeer et al., 2017] Shazeer, N., Mirhoseini, A., Maziarz, K., Davis, A., Le, Q., Hinton, G., and Dean, J. (2017). Outrageously large neural networks: The sparsely-gated mixture-of-experts layer. In *International Conference on Learning Representations*.
- [Si et al., 2019] Si, Y., Wang, J., Xu, H., and Roberts, K. (2019). Enhancing clinical concept extraction with contextual embeddings. *Journal of the American Medical Informatics Association*, 26(11):1297–1304.
- [Smith et al., 2022] Smith, S., Patwary, M., Norick, B., LeGresley, P., Rajbhandari, S., Casper, J., Liu, Z., Prabhunoye, S., Zerveas, G., Korthikanti, V., et al. (2022). Using deepspeed and megatron to train megatron-turing nlg 530b, a large-scale generative language model. *arXiv e-prints*, pages arXiv–2201.
- [Søgaard, 2010] Søgaard, A. (2010). Simple semi-supervised training of part-of-speech taggers. In *Proceedings of the ACL 2010 Conference Short Papers*, pages 205–208.
- [Su et al., 2021] Su, J., Lu, Y., Pan, S., Wen, B., and Liu, Y. (2021). Roformer: Enhanced transformer with rotary position embedding. *arXiv preprint arXiv:2104.09864*.
- [Sutiono and Hahn-Powell, 2022] Sutiono, A. and Hahn-Powell, G. (2022). Syntax-driven data augmentation for named entity recognition. In *Proceedings of the First Workshop on Pattern-based Approaches to NLP in the Age of Deep Learning*, pages 56–60, Gyeongju, Republic of Korea. International Conference on Computational Linguistics.
- [Sutton et al., 1999] Sutton, R. S., McAllester, D., Singh, S., and Mansour, Y. (1999). Policy gradient methods for reinforcement learning with function approximation. *Advances in neural information processing systems*, 12.
- [Tjong Kim Sang, 2002] Tjong Kim Sang, E. F. (2002). Introduction to the CoNLL-2002 shared task: Language-independent named entity recognition. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*.
- [Tolstikhin et al., 2017] Tolstikhin, I., Bousquet, O., Gelly, S., and Schoelkopf, B. (2017). Wasserstein auto-encoders. *arXiv preprint arXiv:1711.01558*.

- [Uzuner et al., 2011] Uzuner, Ö., South, B. R., Shen, S., and DuVall, S. L. (2011). 2010 i2b2/va challenge on concepts, assertions, and relations in clinical text. *Journal of the American Medical Informatics Association: JAMIA*, 18(5):552.
- [Van Engelen and Hoos, 2020] Van Engelen, J. E. and Hoos, H. H. (2020). A survey on semi-supervised learning. *Machine Learning*, 109(2):373–440.
- [van Schooten et al., 2007] van Schooten, B. W., Rosset, S., Galibert, O., Max, A., op den Akker, R., and Illouz, G. (2007). Handling speech input in the ritel qa dialogue system. In *INTERSPEECH*, pages 126–129.
- [Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. (2017). Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.
- [Vijayakumar et al., 2016] Vijayakumar, A. K., Cogswell, M., Selvaraju, R. R., Sun, Q., Lee, S., Crandall, D., and Batra, D. (2016). Diverse beam search: Decoding diverse solutions from neural sequence models. *arXiv preprint arXiv:1610.02424*.
- [Wang et al., 2018a] Wang, A., Singh, A., Michael, J., Hill, F., Levy, O., and Bowman, S. R. (2018a). Glue: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.
- [Wang et al., 2021a] Wang, X., Jiang, Y., Bach, N., Wang, T., Huang, Z., Huang, F., and Tu, K. (2021a). Automated concatenation of embeddings for structured prediction. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2643–2660.
- [Wang et al., 2021b] Wang, Y., Mukherjee, S., Chu, H., Tu, Y., Wu, M., Gao, J., and Awadallah, A. H. (2021b). Meta self-training for few-shot neural sequence labeling. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pages 1737–1747.
- [Wang et al., 2018b] Wang, Y., Shen, Y., and Jin, H. (2018b). A bi-model based rnn semantic frame parsing model for intent detection and slot filling. In *Proceedings of NAACL-HLT*, pages 309–314.
- [Wei and Zou, 2019] Wei, J. and Zou, K. (2019). EDA: Easy data augmentation techniques for boosting performance on text classification tasks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 6382–6388, Hong Kong, China. Association for Computational Linguistics.
- [Weston et al., 2016] Weston, J., Bordes, A., Chopra, S., and Mikolov, T. (2016). Towards ai-complete question answering: A set of prerequisite toy tasks. In Bengio, Y. and LeCun, Y., editors, *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings*.

- [Wu et al., 2016] Wu, Y., Schuster, M., Chen, Z., Le, Q. V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al. (2016). Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*.
- [Xue et al., 2021] Xue, L., Constant, N., Roberts, A., Kale, M., Al-Rfou, R., Siddhant, A., Barua, A., and Raffel, C. (2021). mt5: A massively multilingual pre-trained text-to-text transformer. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 483–498.
- [Yang et al., 2019] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R. R., and Le, Q. V. (2019). Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems*, 32.
- [Yarowsky, 1995] Yarowsky, D. (1995). Unsupervised word sense disambiguation rivaling supervised methods. In *33rd annual meeting of the association for computational linguistics*, pages 189–196.
- [Yu et al., 2017] Yu, L., Zhang, W., Wang, J., and Yu, Y. (2017). Seqgan: Sequence generative adversarial nets with policy gradient. In *Proceedings of the AAAI conference on artificial intelligence*, volume 31.
- [Zhang et al., 2017a] Zhang, H., Xu, T., Li, H., Zhang, S., Wang, X., Huang, X., and Metaxas, D. N. (2017a). Stackgan: Text to photo-realistic image synthesis with stacked generative adversarial networks. In *Proceedings of the IEEE international conference on computer vision*, pages 5907–5915.
- [Zhang et al., 2017b] Zhang, Y., Gan, Z., Fan, K., Chen, Z., Hénao, R., Shen, D., and Carin, L. (2017b). Adversarial feature matching for text generation. In *International Conference on Machine Learning*, pages 4006–4015. PMLR.
- [Zhou and Li, 2005] Zhou, Z.-H. and Li, M. (2005). Tri-training: Exploiting unlabeled data using three classifiers. *IEEE Transactions on knowledge and Data Engineering*, 17(11):1529–1541.
- [Zhu et al., 2015] Zhu, Y., Kiros, R., Zemel, R., Salakhutdinov, R., Urtasun, R., Torralba, A., and Fidler, S. (2015). Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *Proceedings of the IEEE international conference on computer vision*, pages 19–27.

A - ChatGPT conversation

This chapter contains the whole exchange with ChatGPT¹. This conversation used the January 9th 2023 release of ChatGPT. The intention of this conversation was to generate a factice introduction and a sequence labeling example. The prompts are as they were used, which includes grammatical and spelling mistakes.

Me: "Can you please write the introduction to a Ph.D. thesis about nlp. In this thesis, text generation is used as a means to produce additionnal data to improve sequence labeling."

ChatGPT: "Natural Language Processing (NLP) is a rapidly growing field that deals with the interaction between computers and human language. In this thesis, we focus on the use of text generation as a means to produce additional data for improving sequence labeling tasks. Sequence labeling is a fundamental task in NLP that involves assigning labels to individual elements in a sequence, such as words in a sentence or tokens in a text. The goal of this research is to explore the potential of text generation to enhance the performance of sequence labeling models by augmenting the available training data. We investigate various text generation techniques and evaluate their effectiveness on different sequence labeling tasks. The contributions of this thesis include a novel approach for using text generation to improve sequence labeling and an in-depth analysis of the impact of text generation on the performance of sequence labeling models."

Me: "Can you try again. But this time add the constraint of low-resource on the available data."

ChatGPT: "Natural Language Processing (NLP) is a rapidly growing field that deals with the interaction between computers and human language. One of the major challenges in NLP is dealing with low-resource scenarios, where limited amounts of labeled data are available. In this thesis, we focus on the use of text generation as a means to produce additional data for improving sequence labeling tasks in low-resource settings. Sequence labeling is a fundamental task in NLP that involves assigning labels to individual elements in a sequence, such as words in a sentence or tokens in a text. The goal of this research is to explore the potential of text generation to enhance the performance of sequence labeling models by augmenting the available low-resource training data. We investigate various text generation techniques and evaluate their effectiveness on different sequence labeling tasks in low-resource settings. The contributions of this thesis include a novel approach for using text generation to improve sequence labeling in low-resource settings and an in-depth analysis of the impact of text generation on the performance of sequence labeling models in such scenarios."

Me: "Write a text that can be used for sequence labeling containing the following concepts : locations, organizations, and persons."

ChatGPT: "The United Nations headquarters is located in New York City, where representatives from various countries and organizations gather to discuss global

¹<https://chat.openai.com>

issues. The Secretary-General, António Guterres, opened the meeting by addressing the ongoing crisis in Syria and the need for humanitarian aid. He was joined by representatives from the International Committee of the Red Cross and the World Health Organization, who provided updates on their efforts to provide aid to affected areas. In addition, representatives from the governments of Russia and the United States also participated in the meeting, discussing potential solutions to the crisis.”

Me: “Now write a text that can be used for sequence labeling containing medical concepts such as: problems, treatments, and tests.”

ChatGPT: “John Smith, a 54-year-old patient, visited his primary care physician complaining of chest pain and shortness of breath. The physician ordered an ECG test, which revealed that the patient was experiencing an acute coronary syndrome. The doctor prescribed aspirin as a treatment and referred the patient to a cardiologist for further evaluation. The cardiologist recommended a cardiac catheterization procedure to assess the extent of the damage to the coronary artery. Based on the results, the patient was diagnosed with atherosclerosis, and the cardiologist recommended a treatment of angioplasty and stenting. The patient underwent the procedure and was discharged from the hospital the following day. The patient was advised to follow a healthy lifestyle and to take medications to lower his cholesterol levels to prevent future cardiac events.”

B - Pattern Generation Appendix

Size	100	250	500	1000	3000	7000	11483
Baseline	35.8	46.6	56.5	62.4	71.3	76.5	79.1
PatGen 20k	42.6	46.8	59.0	63.6	72.9	75.7	77.8
PatGen 20k m	42.1	47.8	59.2	64.1	73.4	77.4	78.6
PatGen 20k p	43.1	47.8	58.6	63.3	72.2	77.6	77.9
PatGen 20k (m + p)	37.0	47.9	58.2	64.8	73.1	76.2	75.4
PatGen 20k aug. m	41.4	47.1	58.4	65.1	72.4	77.1	78.7
PatGen 20k aug. (m+p)	43.5	49.1	58.6	65.0	74.0	78.2	80.0

Table B.1: BiLSTM results of I2B2.

Size	100	250	500	1000	2000	3000	4478
Baseline	67.8	83.6	87.6	91.1	94.3	95.3	95.3
PatGen 20k	71.7	86.0	90.5	92.9	94.9	95.4	95.8
PatGen 20k m	70.6	87.3	91.0	93.4	95.1	95.3	95.6
PatGen 20k p	70.6	85.7	90.7	93.2	95.3	95.8	95.5
PatGen 20k (m + p)	72.5	87.1	91.1	93.7	94.9	95.1	95.6
PatGen 20k aug. m	70.4	86.1	91.3	93.5	95.0	95.5	95.6
PatGen 20k aug. (m+p)	71.2	86.4	90.4	93.5	95.5	95.9	96.0

Table B.2: BiLSTM results of ATIS.

Size	100	250	500	1000	3000	7000	12916
Baseline	34.5	45.2	55.7	68.0	77.5	81.4	84.6
PatGen 20k	44.0	58.3	60.9	66.3	70.4	71.7	69.6
PatGen 20k m	49.0	57.0	65.2	72.2	79.0	80.8	82.7
PatGen 20k p	47.6	57.2	64.5	69.6	71.0	70.6	68.9
PatGen 20k (m + p)	49.2	57.3	65.7	72.8	79.0	80.6	81.8
PatGen 20k aug. m	48.8	56.9	65.6	73.5	79.9	82.3	84.5
PatGen 20k aug. (m+p)	48.7	56.4	65.9	73.5	79.6	82.5	84.7

Table B.3: BiLSTM results of MEDIA.

Size	100	250	500	1000	3000	7000	13284
Baseline	25.3	46.3	54.8	65.1	78.5	87.5	91.8
PatGen 20k	46.4	58.7	65.6	73.6	82.4	88.6	89.5
PatGen 20k m	45.4	58.4	66.6	74.0	83.6	88.4	90.8
PatGen 20k p	46.7	56.9	66.5	74.7	83.2	88.4	89.8
PatGen 20k (m + p)	46.9	59.2	67.3	73.6	83.7	88.2	90.9
PatGen 20k aug. m	44.7	56.2	67.1	75.1	83.6	89.0	93.1
PatGen 20k aug. (m+p)	45.9	59.1	68.7	74.5	83.8	89.5	91.6

Table B.4: BiLSTM results of SNIPS.

C - BERT tagger architecture results

Size	50	100	250	500	1000
Baseline	57.8±2.9	70.1±5.6	81.2±1.8	85.3±1.1	88.0±0.3
Pattern Generation					
PatGen 20k	66.8 ±3.3	71.7±2.8	80.3±1.6	83.6±1.8	86.7±0.7
PatGen 20k all dist	66.3±2.6	74.3±2.1	80.8±1.3	84.5±1.7	86.8±0.7
Generative Tritraining					
Follow-up	64.3±2.2	73.7±3.3	83.5±1.1	87.2±0.7	89.4 ±0.2
Completion	63.5±2.2	71.6±2.8	83.4±1.3	86.8±0.7	89.2±0.3
Mention rep.	64.5±1.2	72.7±3.1	82.8±1.9	87.2±0.7	89.1±0.4
Context rep.	62.8±3.2	72.3±2.7	82.5±1.4	86.3±0.7	89.0±0.1
Combined	64.2±3.5	74.6 ±3.3	83.6±1.1	87.3 ±0.9	89.3±0.5
Follow-up Individual	63.1±3.0	72.3±3.3	82.2±1.2	86.2±0.8	88.6±0.5
Completion Individual	62.2±2.7	70.4±2.8	82.3±1.5	85.5±1.3	88.5±0.4
Mention rep. Individual	62.9±1.9	71.0±3.3	81.3±1.9	85.8±0.9	88.2±0.5
Context rep. Individual	60.3±4.6	69.6±2.5	80.9±1.9	84.8±1.1	88.0±0.3
Combined Individual	62.7±3.2	72.6±3.6	81.5±2.0	86.3±1.0	88.4±0.5
Follow-up Outputs	64.2±2.1	73.6±3.0	83.9 ±0.9	87.3 ±0.6	89.0±0.7
Completion Outputs	63.9±3.3	72.1±2.6	83.3±1.4	86.6±0.9	88.6±0.1
Mention rep. Outputs	65.2±1.6	72.9±3.2	82.6±1.6	86.3±0.9	88.4±0.4
Context rep. Outputs	61.2±4.7	70.4±2.8	80.2±1.8	85.3±1.4	87.3±0.3
Combined Outputs	62.9±3.2	74.4±2.6	82.8±1.3	87.0±0.6	88.6±0.5
T5 patterns					
T5 20k mention	50.1±4.0	55.7±2.2	64.9±2.5	70.2±1.2	77.1±0.8
T5 20k context	52.0±3.5	62.3±3.2	72.6±2.2	78.8±0.6	83.6±0.5
T5 20k both	53.8±2.2	61.9±1.7	70.1±2.4	75.4±1.4	80.6±1.1
T5 20k filtered	57.1±1.9	62.1±2.3	70.2±1.7	76.2±1.3	81.1±0.6
T5 20k filtered prompt.	55.4±2.9	64.1±2.0	73.8±2.7	78.9±2.5	81.9±0.3
Patterns with Tri-training					
Follow-up + PatGen	64.6±3.1	74.1±1.9	83.1±0.9	86.4±1.5	88.4±0.3
Mention rep. + PatGen	64.7±1.2	74.1±1.9	81.4±2.1	85.8±1.2	86.8±0.5
PatGen on Follow-up	63.2±3.7	72.8±4.0	80.8±1.7	85.4±0.8	87.9±0.3

Table C.1: All results of BERT experiments trained on CoNLL following chapter 3 methodology.

Size	50	100	250	500	1000
Baseline	37.0±3.2	49.4±1.6	63.2±1.6	72.0±0.7	77.3±0.4
Pattern Generation					
PatGen 20k	45.7 ±4.4	56.6±2.2	67.4±0.2	73.8±0.9	79.1±0.8
PatGen 20k (p+m) dist	44.0±6.7	56.0±1.6	67.8±1.2	73.5±1.3	79.1±0.8
Generative Tritraining (small dev)					
Follow-up	38.9±3.0	50.5±2.4	66.1±0.4	75.1±1.4	79.6±0.8
Completion	38.2±4.7	51.6±2.5	67.8±1.4	74.4±0.6	80.5±0.5
Mention rep.	41.4±3.0	54.9±2.0	68.9±1.9	75.3±0.8	79.9±0.7
Context rep.	42.8±2.4	55.0±3.2	68.2±1.1	74.3±0.5	79.4±0.7
Combined	37.9±4.6	52.4±2.2	68.1±0.5	74.7±0.7	79.9±0.8
Follow-up Individual	37.7±2.8	49.5±2.5	64.7±1.2	73.9±1.6	78.6±0.8
Completion Individual	36.9±5.0	50.4±2.5	66.5±1.7	73.1±1.0	79.5±0.9
Mention rep. Individual	39.9±3.3	53.5±2.0	67.4±2.4	74.0±1.2	78.6±1.0
Context rep. Individual	40.5±3.0	53.0±3.4	66.4±1.4	72.9±1.0	78.1±0.8
Combined Individual	37.0±4.7	51.1±2.9	66.6±1.2	73.2±1.5	78.9±1.0
Follow-up Outputs	41.4±2.6	51.1±1.6	67.8±1.3	75.4±1.1	79.8±0.7
Completion Outputs	39.1±3.3	52.6±2.6	68.2±1.3	74.1±1.1	80.3±0.4
Mention rep. Outputs	43.0±2.8	56.1±1.2	70.2±1.3	75.7±1.0	79.8±0.6
Context rep. Outputs	39.8±2.0	52.5±2.6	68.5±1.1	73.7±1.1	78.9±0.8
Combined Outputs	39.1±3.6	53.5±2.6	69.1±0.8	74.8±0.9	80.3±0.5
T5 patterns					
T5 20k mention	38.2±2.4	49.3±3.5	63.2±1.2	71.5±1.8	77.9±1.0
T5 20k context	28.8±2.4	37.9±2.0	49.9±3.0	57.9±1.8	66.8±1.3
T5 20k both	37.7±1.7	44.7±1.3	54.3±1.4	61.6±1.2	67.7±0.5
T5 20k filtered	37.8±1.6	43.3±2.5	54.3±1.5	61.7±1.4	68.5±0.6
T5 20k filtered prompt.	38.8±2.7	45.0±2.6	52.3±1.5	67.0±5.4	72.1±4.8
Patterns with Tri-training					
Follow-up + PatGen	42.7±3.3	53.0±2.7	68.8±0.7	75.9 ±1.2	80.7 ±0.5
Mention rep. + PatGen	42.9±2.2	56.8 ±3.1	70.6 ±1.4	75.3±0.5	80.2±0.4
PatGen on Follow-up	39.8±3.5	50.3±2.1	65.9±1.2	74.3±1.4	79.5±0.2

Table C.2: All results of BioBERT experiments trained on I2B2 following chapter 3 methodology.

Size	50	100	250	500	1000
Baseline	44.3±3.0	53.9±1.3	66.4±1.1	74.3±1.4	79.1±0.8
Pattern Generation					
PatGen 20k	50.7 ±3.8	57.5±1.5	67.1±2.2	74.0±0.9	78.0±0.3
PatGen 20k (p+m) dist	50.0±3.9	57.9±2.1	68.3±1.1	74.3±0.8	78.8±0.3
Generative Tritraining					
Follow-up	46.5±3.0	55.6±1.7	67.9±1.7	75.6±1.0	81.2±0.5
Completion	48.3±3.3	56.5±1.0	68.6 ±1.7	76.2 ±0.9	81.3±0.3
Mention rep.	47.7±2.6	57.4±1.0	68.5±1.4	75.9±1.0	81.4 ±0.6
Context rep.	48.3±3.2	56.6±0.9	68.3±1.4	75.9±0.9	81.1±0.5
Combined	47.4±2.9	57.0±0.8	68.3±1.8	75.6±1.2	81.2±0.6
Follow-up Individual	45.1±3.5	54.7±1.7	66.9±1.4	74.4±1.1	80.2±0.6
Completion Individual	47.1±3.2	55.5±1.2	67.2±1.6	74.7±1.0	80.1±0.5
Mention rep. Individual	46.4±2.8	54.3±4.1	67.1±1.5	74.7±1.1	79.8±1.5
Context rep. Individual	46.8±3.3	55.6±1.0	66.8±1.4	73.8±1.8	80.1±0.7
Combined Individual	47.2±3.9	55.6±1.2	67.0±1.7	74.6±1.2	79.9±0.6
Follow-up Outputs	46.4±2.6	54.6±2.8	67.2±1.2	74.9±0.9	80.3±0.4
Completion Outputs	47.1±2.6	55.1±0.4	67.0±1.2	75.0±1.3	80.1±0.6
Mention rep. Outputs	46.9±3.0	56.6±1.4	67.6±1.8	74.4±0.7	80.1±0.5
Context rep. Outputs	46.3±3.1	55.0±1.1	67.1±0.6	75.0±1.0	79.8±0.5
Combined Outputs	47.0±2.6	55.4±1.2	67.5±1.0	74.6±0.9	80.0±0.7
T5 patterns					
T5 20k mention	35.1±1.3	42.6±1.4	55.7±2.1	64.7±0.0	74.1±0.1
T5 20k context	47.9±2.0	55.7±1.1	65.6±0.8	72.1±0.8	78.7±0.5
T5 20k both	41.3±2.6	48.9±1.5	60.1±2.0	69.5±1.4	75.8±0.1
T5 20k filtered	39.5±1.7	47.5±1.0	59.5±1.7	68.1±1.0	75.3±0.0
T5 20k filtered prompt	40.6±1.6	47.2±1.0	59.4±2.4	68.4±0.6	75.0±0.2
Patterns with Tri-training					
Follow-up + Patgen	49.5±2.4	57.6±1.0	67.7±2.4	74.4±0.8	79.0±0.5
Mention rep. + Patgen	50.0±3.3	58.6 ±1.6	68.4±1.2	74.8±1.0	79.1±0.6
PatGen on Follow-up	47.8±1.9	55.7±1.9	66.9±1.6	73.6±1.3	79.1±0.6

Table C.3: All results of mBERT experiments trained on MEDIA following chapter 3 methodology.

Titre: Augmentation et génération de données d'apprentissage en traitement automatique des langues.
Mots clés: Apprentissage, génération, TAL, extraction d'information.

Résumé: De plus en plus de domaines cherchent à automatiser une partie de leur processus. Le traitement automatique des langues contient des méthodes permettant d'extraire des informations dans des textes. Ces méthodes peuvent utiliser de l'apprentissage automatique. L'apprentissage automatique nécessite des données annotées pour faire de l'extraction d'information de manière optimale. L'application de ces méthodes à de nouveaux domaines nécessite d'obtenir des données annotées liées à la tâche. Le problème que nous souhaitons résoudre est de proposer et d'étudier des méthodes de génération pour améliorer les performances de modèles appris à basse quantité de données. Nous explorons différentes méthodes avec et sans apprentissage pour générer les données nécessaires à l'apprentissage de modèles d'étiquetage.

La première méthode que nous explorons est le remplissage de patrons. Cette méthode de génération de données permet de générer des données annotées en combinant des phrases à trous, les patrons, et des mentions. Nous avons montré que cette méthode permet d'améliorer les performances des modèles d'étiquetage à très petite quantité de données. Nous avons aussi étudié la quantité de données nécessaire pour l'utilisation optimale de cette méthode.

La deuxième approche de génération que nous avons testé est l'utilisation de modèles de langue pour la génération couplée à l'utilisation de méthode d'apprentissage semi-supervisé. La méthode d'apprentissage semi-supervisé utilisé est le tri-training et sert à ajouter les étiquettes aux données générées. Le tri-training est testé sur plusieurs méthodes de génération utilisant différents modèles

de langue pré-entraînés. Nous avons proposé une version du tri-training appelé tri-training génératif, où la génération n'est pas faite en amont, mais durant le processus de tri-training et profite de celui-ci. Nous avons testé les performances des modèles entraînés durant le processus de semi-supervision et des modèles entraînés sur les données produites par celui-ci. Dans la majeure partie des cas, les données produites permettent d'égaliser les performances des modèles entraînés avec la semi-supervision. Cette méthode permet l'amélioration des performances à tous les niveaux de données testés vis-à-vis des modèles sans augmentation.

La troisième piste d'étude vise à combiner certains aspects des approches précédentes. Pour cela, nous avons testé différentes approches. L'utilisation de modèles de langues pour faire du remplacement de bouts de phrase à la manière de la méthode de remplissage de patrons fut infructueuse. Nous avons testé l'addition de données générées par différentes méthodes qui ne permet pas de surpasser la meilleure des méthodes. Enfin, nous avons testé l'application de la méthode de remplissage de patrons sur les données générées avec le tri-training qui n'a pas amélioré les résultats obtenu avec le tri-training.

S'il reste encore beaucoup à étudier, nous avons cependant mis en évidence des méthodes simples, comme le remplissage de patrons, et plus complexe, comme l'utilisation d'apprentissage supervisé avec des phrases générées par un modèle de langue, permettant d'améliorer les performances de modèles d'étiquetage grâce à la génération de données annotées.

Title: Data augmentation and generation for natural language processing.

Keywords: Machine learning, generation, NLP, information extraction.

Abstract: More and more fields are looking to automate part of their process. Automatic language processing contains methods for extracting information from texts. These methods can use machine learning. Machine learning requires annotated data to perform information extraction. Applying these methods to new domains requires obtaining annotated data related to the task. In this thesis, our goal is to study generation methods to improve the performance of learned models with low amounts of data. Different methods of generation are explored that either contain machine learning or do not, which are used to generate the data needed to learn sequence labeling models.

The first method explored is pattern filling. This data generation method generates annotated data by combining sentences with slots, or patterns, with mentions. We have shown that this method improves the performance of labeling models with tiny amounts of data. The amount of data needed to use this method is also studied.

The second approach tested is the use of language models for text generation alongside a semi-supervised learning method for tagging. The semi-supervised learning method used is tri-training and is used to add labels to the generated data. The tri-training is tested on several generation methods using different pre-trained language models. We proposed a version of tri-training called generative

tri-training, where the generation is not done in advance but during the tri-training process and takes advantage of it. The performance of the models trained during the semi-supervision process and of the models trained on the data generated by it are tested. In most cases, the data produced match the performance of the models trained with the semi-supervision. This method allows the improvement of the performances at all the tested data levels with respect to the models without augmentation.

The third avenue of study combines some aspects of the previous approaches. For this purpose, different approaches are tested. The use of language models to do sentence replacement in the manner of the pattern-filling generation method is unsuccessful. Using a set of data coming from the different generation methods is tested, which does not outperform the best method. Finally, applying the pattern-filling method to the data generated with the tri-training is tested and does not improve the results obtained with the tri-training.

While much remains to be studied, we have highlighted simple methods, such as pattern filling, and more complex ones, such as the use of supervised learning with sentences generated by a language model, to improve the performance of labeling models through the generation of annotated data.