FEW-SHOT INTENT CLASSIFICATION IN USER-GENERATED SHORT TEXTS: APPLICATION TO CONVERSATIONAL AGENTS

A DISSERTATION
SUBMITTED TO THE DEPARTMENT OF COMPUTER SCIENCE
AND THE COMMITTEE ON GRADUATE STUDIES
OF UNIVERSITE JEAN MONNET
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

Thomas Dopierre
October 2021

THESE de DOCTORAT DE L'UNIVERSITE DE LYON

opérée au sein de

Université Jean Monnet

Ecole Doctorale ED SIS 488

(**Ecole Doctorale Science, Ingénierie et Santé**)

Spécialité de doctorat:

Informatique

Soutenue publiquement le 26/11/2021 par

Thomas Dopierre

Few-Shot Intent Classification in User-Generated Short Texts: Application to Conversational Agents

Devant le jury composé de :

Sophie ROSSET	Directrice de recherche	LIMSI, CNRS	Rapporteure
Jian-Yun NIE	Professeur	Université de Montréal	Rapporteur
Laure SOULIER	Maître de conférences	LIP6, Sorbonne Université	Examinatrice
Frederique LAFOREST	Professeure	LIRIS, INSA Lyon	Examinatrice
Christophe GRAVIER	Professeur	Université Jean Monnet	Directeur de thèse
Wilfried LOGERAIS	Head of Data Science	Meetic	Encadrant Professionel

Contents

List of Tables

List of Figures

Few-Shot Intent Classification in User-Generated Short Texts: Application to Conversational Agents

Thomas Dopierre

October 19, 2021

Chapter 1

Introduction

1.1 Conversational Agents

Natural language is used every day by billions of humans who intend to communicate between each other. Such languages usually evolve with time with the introduction of new concepts and ways of formulating such concepts or words disappearing from the common tongue as they are used less and less often. While communication can be achieved using words, there are also other ways to exchange knowledge without using classical dictionaries. For example, emojis are ways of expressing an idea which are constantly growing in usage [1]. Every year, new emojis are introduced, creating new ways of expressing an idea.

While languages from different countries share some common aspects, centuries passed and languages all evolved differently. Even inside the same country, dialects are different depending on the region. People with different ages also use different words, as dialects evolved as they are passed from generation to generation. Such fluctuations in the way people express themselves make languages hard to pin down to a given set of rules.

Nonetheless, in our digital area, there is an urge to process contents expressed in those ever-evolving and multiple languages. The most obvious way to answer this need is to use algorithms running on computers so that contents can be classified, sorted, clustered or even computer-generated. This field of Computer Science research is known as Natural Language Processing (NLP). In this thesis, we will take

[1] https://blog.emojipedia.org/emoji-trends-that-defined-2020/

Natural Language Processing (NLP) in a wide sense describing all computer manipulations of natural languages.

Given the increasing amount of available online services, technologies using NLP are becoming more and more widespread. This ranges from scanning emails for spams up to, for instance, more complex tasks such as computer-generated news generation or news feeds summarization. Not only are NLP technologies accessible on a personal computer but they are also fingers away from the end users. People using modern mobile phones are exposed with such technologies on a daily basis, on a different level of awareness depending on the technology. For example, autocorrection and next word prediction on a mobile phone's keyboard is considered as a technology based on NLP, as it analyses correctly written sentences in order to make the life of the end user easier.

One of the biggest challenges that NLP pursue is to provide comprehensive conversational agents that would be able to discuss with humans in order to assist them in their activities – this work belongs to that frame of research. This thesis takes place in an era of research in which there is a renewed interests and confidence to build such agents thanks to neural architectures [52]. In this thesis, we will focus on the application of NLP to Conversational Agents, one sub-domain of NLP. The race for building an "intelligent" conversational agent started very early: in 1964, ELIZA, the first conversational computed program was created by Joseph Weizenbaum. This computer program was based on a fixed set of rules, which gave the user an illusion of being understood, an illusion of having a real conversation. In the context of current digital economy, businesses are processing more than 265 billion [2] customer requests per year, representing businesses expenditures in a $1.3 trillion ballpark. Using chatbots (another way to denote conversational agents in this work) to process such requests could help save up to 30% of this cost[2].

From an application point-of-view, chatbots are made both to understand queries and to provide appropriate answers. One way to address the former is to perform intent classification, which is trying to figure out what is the intent expressed in the user's query. For example, the classification model would have to separate "I want

[2]https://www.ibm.com/blogs/watson/2017/10/how-chatbots-reduce-customer-service-costs-by-30-percent/

to order pizza" from "I need to book a flight". For the answering part, the conversational agent can either generate a response (generation-based model), or retrieve an answer from a set of possible answers. In this thesis, we focus on the intent detection task, which attempts to understand and categorize user queries into intents classes. It is a special case of the standard text classification task, in which textual contents are user-generated (which imply issues such as length heterogeneity, grammatical and construct errors, slang, ...) and classes are usually counted by dozens in practical settings (where a spam detection algorithm is a binary classifier for instance). To build intent detection classifier, it is therefore expected to need a large amount of training data since modern NLP approaches rely on Deep Artificial Neural Networks [52, 137, 33], which are data-hungry. With the fluctuations of the human language discussed above, the variety of the different languages as well as their proper dialects, and the fact that chatbots usually evolves with time as new features are introduced, constructing a robust training dataset is not a simple task. On the contrary, given annotation cost and the fact that such annotations may be irrelevant in a close future due to the chatbot evolution, chatbots often face the case where not much labeled data is available to train on. This thesis therefore investigates how to effectively train intent detection for chatbots with a limited amount of data, which is a training regime named as few-shot learning as described in Section 2.4. As few-shot learning does always mean no training data (this special scenario being referred to as zero-shot learning [149] in the literature), we will briefly discuss in the next Section the stakes and issues of data annotation.

1.2 Data Annotation

One of the first step when building a conversational agent is to gather a dataset. To understand user queries, chatbots need samples of such queries associated to one or more intent labels, in order to learn to classify user requests it will receive in the future. The worldwide data volume increasing exponentially[3] is good news for this, as it means plenty of data is available. In the conversational agent context, acquiring unlabeled data is quite cheap, as we just have to collect logs from past conversations between users and the conversational agent. Unfortunately, this data

[3]https://www.statista.com/statistics/871513/worldwide-data-created/

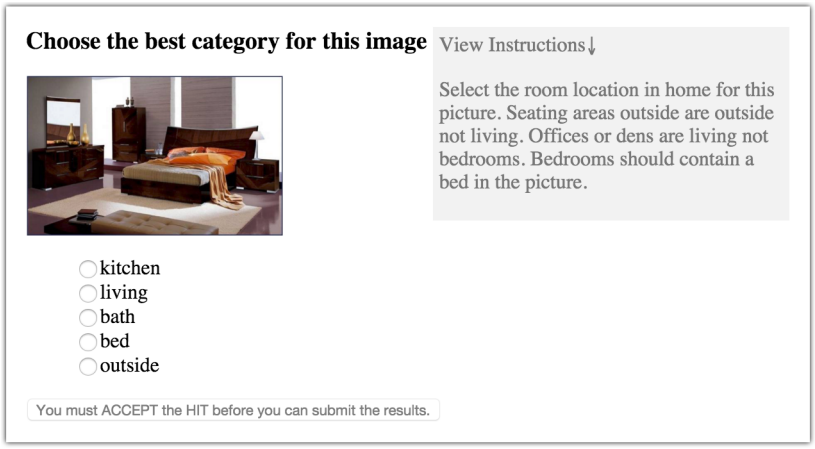


Figure 1.1: Example of an image labeling task on a crowd-sourcing website.

is not annotated, and need a human judgement in order to be leveraged as labeled data. To annotate such data, crowd-sourcing websites are mostly used [108, 153], where people around the globe are paid to perform annotation tasks.

However, a recent study [4] on the Amazon Mechanical Turk [5] platform shows that the number of annotators around the globe is at best stable, at worst decreasing between 2017 and 2018. With the increasing volume of unlabeled data, as well as increasing number of applications based on Machine Learning which require labeled data, the decreasing number of annotators leads to a dead end. To make things worse, as NLP tasks are getting more complex, the annotation process time increases, and adds up very quickly money-wise. An example is machine comprehension [114]: it requires time for annotators to read and to process a paragraph and to select the answer to a given question whose answer may appear in this paragraph. For more standard text classification tasks, as we are looking to more and more challenging datasets in order to train classifiers able to support more sophisticated scenarios as we progress research-wise, we are expecting more subtlety, hence efforts, in the annotation process. This increased complexity for the human annotators is significant, and it is magnified by the requirement to have several humans tagging the same samples in order to both obtain more reliable labels but also identify malicious workers.

To overcome this issue, we are looking for algorithms which are not very data

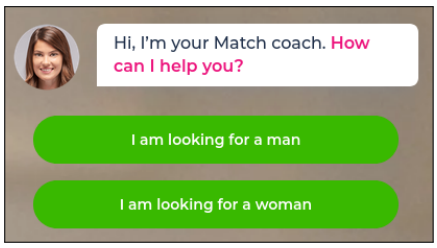
[4] https://www.cloudresearch.com/resources/blog/how-many-amazon-mturk-workers-are-there/
[5] https://www.mturk.com/

hungry, or to say the less, which performs "well enough" even if they do not have a large amount of labeled data at hand. The appreciation of their performance gap is task and context dependent. In this thesis, we are providing directions in order to tackle the time and cost associated to data annotation by trying to explore relevant and efficient ways to limit the need for human annotators. Not only does it aim at lowering the cost of building intent detection module for chatbots, but the more we alleviate using human in this process, the faster is the construction and iteration of a learning algorithm dedicated to intent detection.

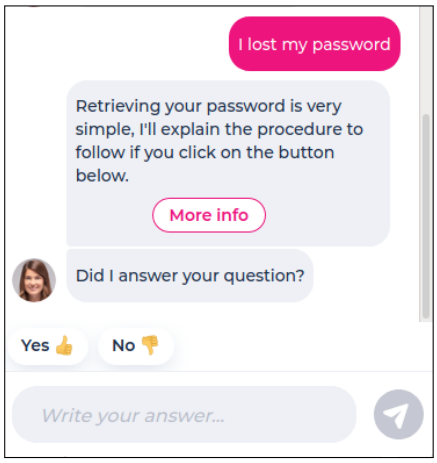
Since this thesis takes place in both an academic (Laboratoire Hubert Curien UMR CNRS 5516) and a company (Meetic from Match Group) context, the next section introduces some insights on the practical chatbot in which our contributions can be instantiated. While none of our contributions are specific to this chatbot, it provides an insights on the practical issues that motivates our research.

1.3 Lara, Meetic's Chatbot

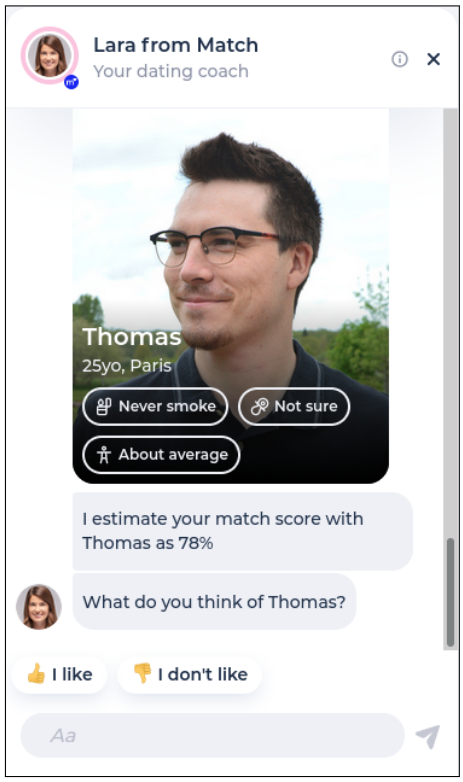
Meetic is a french dating website founded in 2001. With hundreds of thousands of users using it daily, it is one of the most popular dating websites in France. Created in 2016, Lara is Meetic's conversational agent. As she interacts with users on the mobile app, she covers multiple goals (Figure 1.2), including profile creation, answering customer care related questions, or even profile recommendation. From the very beginning of the user experience, she helps creating one's profile. When users encounter a problem, for example when they lose their password, Lara is here to help them fix the issue. However, the core of her job is to act as a dating coach. In this role, she helps user improve the quality of their profile (e.g. fill in the blanks, choose the best picture), does profile recommendation (more than $70,000$ profiles recommended per day), as well as provide customized dating tips. Overall, she recognizes more than 300 user intents spanning those different use cases (this indicates the number of classes we consider in our intent detection algorithms). Those intents evolve in number and scope with time. At the beginning of my thesis, there were less than 30. As the website's features evolved, and the user behavior too, we gradually added more and more intents to better capture what users were talking about. This led to scenarios where we had the need for intent classification methods which were



(a) Profile creation



(b) Customer care



(c) Profile recommendation

Figure 1.2: Different use-cases for Lara

quick to adapt: such release cycles are hardly compatible with going over a new full data annotation process each time when classes are added, removed, or partially merged.

Lara also speaks 6 languages, as depicted in Figure 1.3. While collecting and annotating data in English and French was possible – yet painful – for us, doing the same in every languages is one order of magnitude more costly. Moreover, annotated data in English are more frequent on the Web, yet it is much harder to find datasets which might help in the other languages. While one can argue that Neural Machine Translation [7] systems can provide solution to this problem, but we will later demonstrate (Section 5.4.3) that it is hardly a solution for user intent detection given user-generated text specific features. Overall, multilingualism increases the need for method which were good at quickly adapting to a new intent detection setting given only few labeled data.

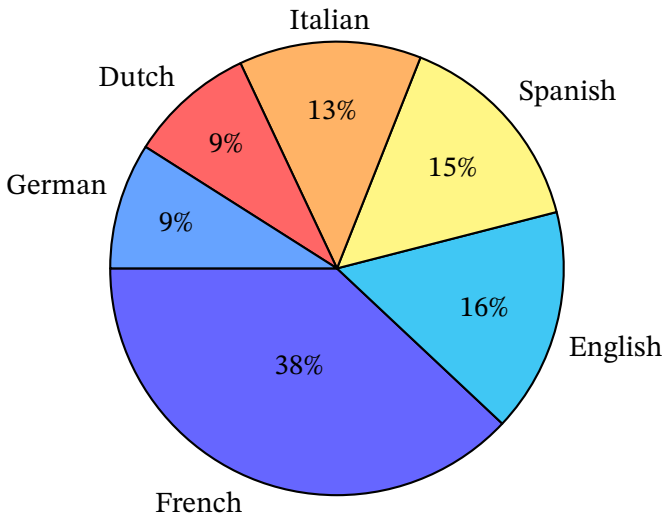


Figure 1.3: Language repartition of users interacting with Lara

1.4 Thesis Contributions

This section provides an overview of the contributions proposed in this thesis.

1.4.1 On the importance of sentence embeddings for Few-Shot Learners

As a first contribution, we revisit the State-of-the-Art for the Few-Shot Intent Detection task, with surprising results. Approaches to learn representations for text, that is learning to embed pieces of text into vectors, have quickly evolved in the past few years 2.2. This embedding step is the primary resource for downstream NLP neural architectures that tackles specific application, and this includes intent detection in a few-shot setting. In the meantime, the existing literature about few-shot intent detection have been developed while text encoders have progressed. Consequently, when a new few-shots intent detection method becomes the new state-of-the-art, it is left unsaid if this progress is due to the new few-shot neural architecture or simply introduced using of a more modern and efficient text encoder with respect to the previous best performing few-shot intent detection algorithm. To avoid this biased comparison, we first measure the impact fine-tuning a state-of-the-art text encoder

on intent detection datasets in terms of embeddings quality. Then, we run experiments to compare the various few-shot learning methods proposed in the last decade using the fine-tuned text encoder as input and we show that progress in this field is mostly, if not entirely, due to progress by the text encoders. This shows that few-shot intent detection remains a difficult task to tackle. This contribution has been the subject of a long paper [37]. published at EACL'21.

1.4.2 A Folding/Unfolding method for Pseudo-Labeling

In a labeling framework, it is hard for the annotator to select the correct label if the number of classes is high, and this is amplified if the available volume of data one can annotate is high. An possible approach therefore consists in hypothesising a label to an unlabeled data given its proximity in the representation space. This process is called pseudo-labeling [5]. The resulting so-called pseudo-labels can then be used for training, leaving the learning algorithm to deal with the associated uncertainty, or the annotator can decide whether the given pseudo-label is appropriate – it fosters the annotation process in this latter case. In this context, a second contribution we introduce is a new two-step pseudo-labeling algorithm that is stat-of-the-art for intent detection. This method not only surpasses other baselines, it is also complementary to existing approach so that aggregating existing pseudo-labeling techniques with ours yield even better pseudo-labeling accuracy. This contribution has been the subject of a long paper [38] published at COLING'20.

1.4.3 Extending Prototypical Networks using unlabeled data and diverse paraphrasing

As a last contribution, we introduce a novel method for end-to-end few-shot intent detection. In the conversational agent domain, it is cheap and easy to gather a lot of unlabeled user queries, as more and more users interact with the agent. Given a few labeled data and some unlabeled data, we want to find a way to combine both in order to get a robust model. First, we fine-tune a paraphrase generation model. Then, we use this fine-tuned model to generate paraphrases, with different strategies to enforce the diversity among the generated sentences. Finally, we introduce a novel extension of Prototypical Networks leveraging both the knowledge from labeled data

and the unsupervised knowledge from unlabeled data and their diverse paraphrases. Our novel techniques show great results in the few-shot intent detection task, even out-performing the labeled baselines using less annotated data. This contribution has been the subject of a long paper [36] published at ACL'21.

1.5 Conclusion

In this thesis, we approach the Intent Detection task in a few-shot setting. Chapter 2 first starts by introducing all background knowledge required to understand the contributions presented in the next chapters. As a first contribution, we will measure the importance of the text encoder when dealing with Few-Shot text classification model Along with a revisited state-of-the-art on the matter (Chapter 3). Then, we introduce a novel method of producing pseudo-labels on unlabeled data, which helps the model generalize (Chapter 4). Finally, Chapter 5 presents a novel approach to few-shot intent detection, using an unsupervised paraphrase generation model, which yields competitive results even in an extremely low annotated data regime. Chapter 6 concludes and provides insights on open direction for research.

Chapter 2

Background

2.1 Introduction

In this section, we will dive into all the background knowledge required to contextualise this thesis' contributions. First, we will describe the various sentence representation methods and how they quickly evolved in the last few years. Secondly, we will describe the intent detection task that we want to solve. Then, we will have a look on the various few-shot learning methods which have been used in the past to solve this problem. Finally, we will introduce all the intent detection datasets that are used in the experiments of our contributions.

2.2 Word & Sentence Representations

Texts, in their most raw format (a string), cannot be directly processed by a computer. Instead, we need a way to convert such texts to numerical values so that a computer can accomplish a task using those texts. Sentence Representations [78, 68] – or sentence embeddings, those terms are interchangeable –, are representations of sentences in an n-dimensional vector space, such that two sentences sharing some semantic similarities are close in such space. Embeddings are not just used for texts – images, products, users can be embedded into vectors [147, 71] –, however in this thesis we are only interested in text embedding techniques. The embedding of a sentence is derived depending on the words it contains. From one-hot encoding to Transformers, sentence embedding techniques quickly evolved in the recent years.

In this section, we will describe those different techniques, how they came to be, as well as their strengths and shortcomings. In what follow, we will refer to **vocabulary** as the set of unique words that compose the language model we want to build.

2.2.1 One-hot encoding

The simplest way to represent words using numerical values is to one-hot-encode [70] them. Using this method, each word is represented by a large vector of the size of the vocabulary. In this vector, all values are 0 except a 1 at the word's index in the vocabulary. Given those individual representations of words, we can derive a sentence representation as the average or all word representations it contains. One-hot encoding treats all words equally, would they be very significant or only serve a grammatical function.

An extension of this method, called TF-IDF [65] (Term Frequency - Inverse Document Frequency), assigns weights to words. The more frequent a word is in a document, the higher the weight. However, if this word is present in many documents, then the weight increase is penalized (it is most likely not a discriminator of the documents). Consequently, this inverse document frequency mechanism avoids putting too much weight on words which are too common (e.g. stop-words), and emphasises rare – and hopefully, more important – words. Sentence representations obtained by this extensions brought some great results on the text classification task [64], or even sentiment analysis [94]. Still, while being superior than assigning boolean values, this trick does not correct the following flaws of those vocabulary-based encoding methods.

Word Similarity These one-encoding methods do not model the similarity between two words. According to these embedding methods, there is no shared knowledge between any two pair of words. Two given words will just have many 0's in common, and a different index at which they have a non-null value, no matter the similarity between the two words. This means that, according to this representation technique, there will not be more similarity between `telephone` and `smartphone` than between `telephone` and `snake`.

Scalability The size of vectors is the size of the vocabulary. According to a study [1], there could be around 170,000 words currently in use in the English language. If we are to represent each word with a 170,000-dimension vector, the computer memory tax is prohibitive.

Sparsity Vectors produced by this method are extremely sparse, hence models will have a hard time learning anything from such sparse data [14]. Moreover, if the model sees a new word it has not previously seen, it will be unable to infer anything from it.

Word Order Those methods do not take into account the words order in the sentence. Using basic TF-IDF, both sentences `` `You know I do not like it' `` and `` `I do not know it like you' `` will have the same representation, while being very dissimilar in terms of meaning.

To overcome those issues, word embeddings were introduced, and will be covered by the next section.

2.2.2 Word Embeddings

Word embeddings map each words to a dense N-dimensional space. Depending on the implementation, N can vary from 50 to 1,000. This dimension reduction compared to the vocabulary size in the previous methods solves the scalability issue. The sparsity issue is also addressed as the target space is dense. Finally, the word similarity issue is tackled by the training method of word embeddings, which aims at bringing closer words which share a common meaning.

Word embeddings were first introduced by the Word2Vec [100] method. The concept of this method was based on the distributional hypothesis, suggesting that word occurring in the same linguistic context have a similar semantic meaning. Following this hypothesis, Word2Vec aims at finding a representation of words in the N-dimensional space having similar words close to each other. Word2Vec introduces two training methods: **Skip-Gram** and **C-Bow**. The former, tries to retrieve a word given its surrounding words, defined as its *context*. On the opposite, **C-Bow** aims at

[1] https://englishlive.ef.com/blog/language-lab/many-words-english-language/

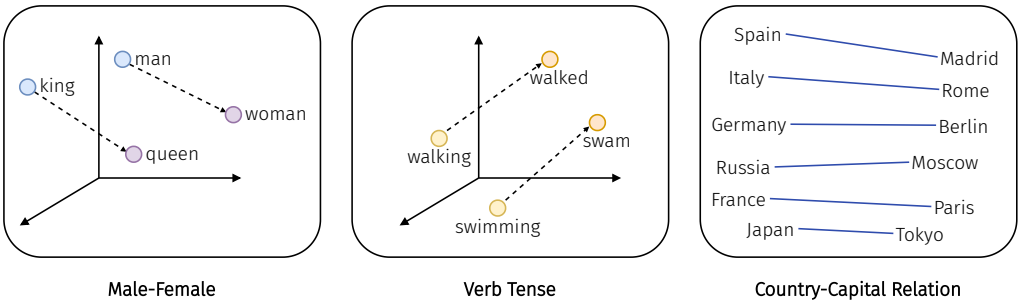


Figure 2.1: Examples of similarities captured by dense word embeddings

predicting a context given a word as an input. The two models were composed of a shallow 2-layered Neural Network to model the probability of a word existing in a context.

In the dense space of word embeddings, two words which are close in this space share some kind of similarity. Additionally, word embeddings can capture relations between words, as shown by the examples of Figure 2.1. Words embeddings were a major breakthrough, as they enabled the ability to represent and compare words and sentences based on their meaning. Many other extensions of word embeddings were proposed to enrich the quality of the derived vectors, like Dict2Vec [136] which leverage word definition in the Word2Vec objective and which was developed in my laboratory. Word embeddings can also be enriched using sub-word information, as in FastText [17]. Using this method, sub-words, which are chunks of words, also have their own embedding. The embedding of a word is computed using the average embedding of its sub-word.

No matter the quality of the produced word embeddings, an issue was still to be addressed: a given word may have a different meaning depending on the context. For example, consider the following sentences:

- *I'm into blond girls.*
- *Can you serve me a blond beer, please?*

In both sentences, the word "blond" appears, but the context is very different. Hence, if would not be appropriate to use the same vector representing "blond" in these two contexts. To overcome this fixed embeddings issue, we would like to find a way to enrich or modify the value of a word embedding depending on the context in which

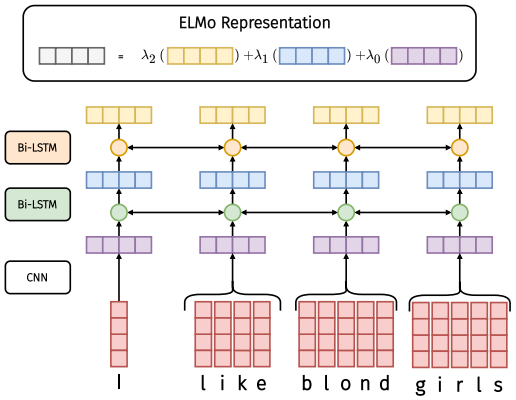


Figure 2.2: Overview of the ELMo architecture to produce contextualised word embeddings

the word appears. Such occurrence-aware embeddings are called contextualized embeddings and are presented in what follows.

2.2.3 Contextualized Embeddings

Contextual word embeddings were introduced by CoVe [95], then popularised by ELMo [110], a Language Model based on a deep neural network architecture. The goal of such models is to make the embedding of a word dependent of the context of the sentence it appears in. The first objective of the ELMo architecture addresses the need of embeddings for out-of-vocabulary words. To overcome this issue, words are first broken down into characters, which will be processed by the model. As an example, instead of processing the word "desorganising" directly, the word will be considered as the sequence: **desorganising = d+e+s+o+r+g+a+n+i+s+i+n+g**. As such, words which have never been seen by the model can still be processed. After assigning a fixed embedding to each character, the model uses Convolutional [79] Layers on top of those character to derive word representations.

The second issued addressed by ELMo is the lack of context in the word embeddings. Given a sentence, instead of embedding each word independently, it will take the entire input sentence in order to derive each word's embedding. Those early word representations obtained with character-level convolutional layers are not contextual, which means that adjacent words have no impact on each other. Then, two

Bi-directional LSTMs [57] networks are used in order to produce contextualised representations. This bi-directional approach allows the model to learn how to read a text in both ways, either predicting the next or the previous word based on context. Finally, a linear combination of hidden layers – from both LSTMs as well as the output of the convolutional layer – in order to have a robust and contextualised ELMo representation of words within a sentence. The linear combination of the various hidden layer is **task-specific**: this is where ELMo breaks away from the different architectures. Indeed, downstream tasks are all different, and they all need to capture specific knowledge in a given sentence. After pre-training ELMo on a large text corpus, only the weights assigned to the various layers need to be fine-tuned on a specific task.

This novel way to embed words and sentences paved the way to major improvements for many NLP tasks, like Question Answering [118], Sentiment Analysis [154], or Entity Disambiguation [125]. The main shortcomings of ELMo is dependent to the two LSTMs (left to right and right to left word predictions): those recurrent networks are slow to train and they have a tendency to "forget" (although they are equipped with a forget gate which makes them better at capturing long-term dependencies than standard Recurrent Neural Networks, they still struggle to model long tokens dependencies). ELMo was later replaced by a novel architecture called the Transformer which try to tackle this shortcomings.

2.2.4 Transformers

Attention Mechanism Until the Transformers era, sequence models were based on complex recurrent and/or convolutional neural networks, often paired with an attention mechanism. This attention mechanism was first introduced in [7] for the Neural Machine Translation (NMT) task. NMT use an encoder to embed the original piece of text in a given language (e.g. sentence) in a latent representation space, and then a decoder that predicts the sequence of tokens in the target language with respect to that latent representation used as an input. The two main issues for NMT models is the need to encode variable-size sentences into fixed-width vectors and the observation that the target language sentence may not be structured or may not contain the same kind of tokens than the original one (from a vocabulary or grammatical point of view). An overview of a NMT architecture is represented in Figure 2.3. To

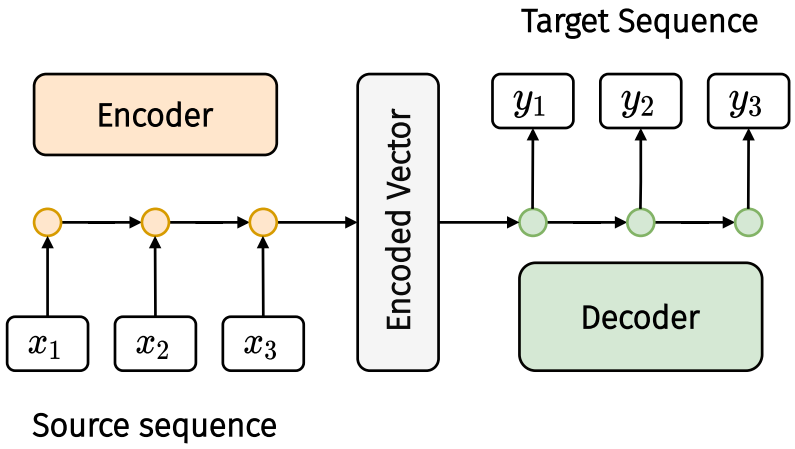


Figure 2.3: A simplified overview of the encoder-decoder recurrent neural network architecture. A bottleneck happens as all the knowledge from the input sentence is hopefully represented into a fixed-width vector.

tackle this issue, [7] proposed to extend the decoding step by allowing the model to automatically select which parts of the input sentence it needs to look at in order to decode the next word at each decoding step. In other word, they allow the model to look back into the original sentence tokens, and learns how to weight the latent representation based on both the original tokens and the decoded token so far. By providing a direct path to the inputs, this so-called attention mechanism also helps to alleviate the vanishing gradient problem (these connections act like residual connections).

The transformer [137] is a novel encoder/decoder architecture introduced in 2017 and it builds upon the attention mechanism. This new architecture was a major breakthrough in the NLP field, as plugging any tasks on top of this novel architecture yielded significant improvements over the state-of-the-art. Up until now, the go-to method was to use Recurrent Neural Networks, like the LSTM or the GRU [34, 54] (**G**ated **R**ecurrent **U**nit). Instead, as the title of the associated paper suggests – *Attention is all you need* –, the transformer is solely based on the attention mechanism. As shown in Figure 2.4, the transformer is made of successive modules, each one composed of Multi-Head Attention (MHA) and Feed Forward (FF)

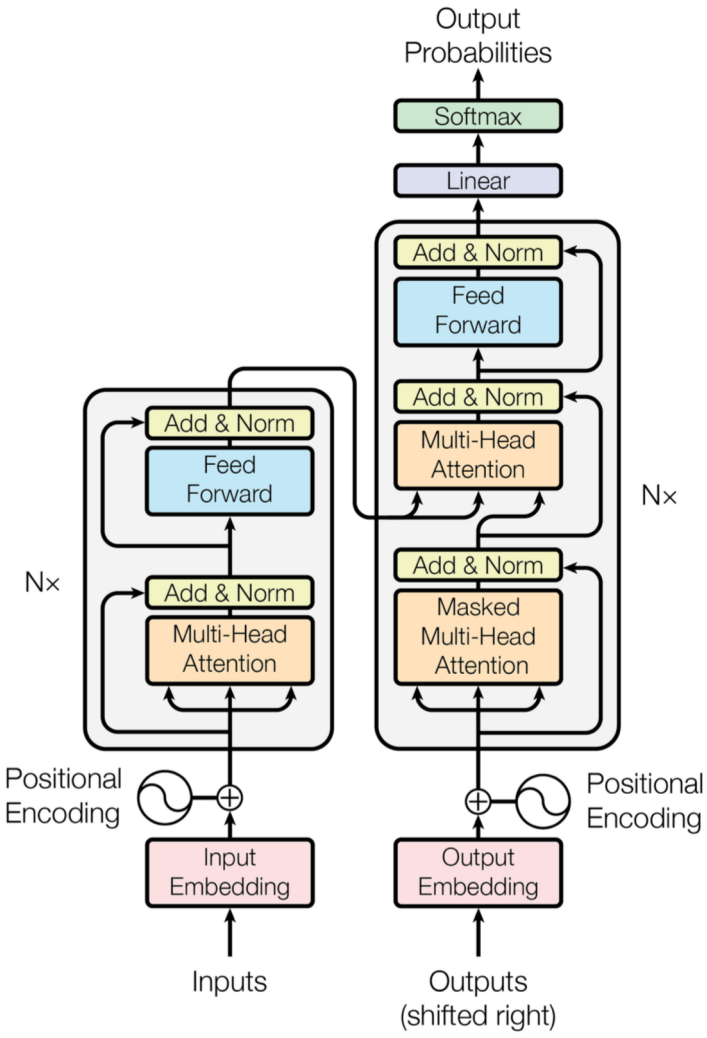


Figure 2.4: The transformer architecture, taken from [137]

.

layers. FF layers are very standard layers known since decades [103]. MHA is an attention module similar to the attention mechanism described previously, only that several attention are combined. More precisely, instead of using a single way to attend to the inputs at each decoding steps, the Transformer learns several attention modules (coined as "attention heads") that each provides a different looking back gaze on the original sentence. The reasons for having multiple attention heads are as follows. First, each head tends to attend to the inputs in a different manner (e.g. verbs vs nouns, close or long range dependencies, etc.), conditioned by their parameters random initialization [119]. Second, the training of each head is independent of each other: it can be done in parallel, therefore with a limited training overhead. Note that all attention heads are ultimately concatenated and transformed to fit the required latent representation dimension through a linear combination.

By only using an attention mechanism, this architecture does not require to read a text sequentially, as RNNs would do. Nonetheless, as previously discussed in Section 2.2.1, two sentences composed of the same words but in a different order might have a completely different meaning. In order to take into account the token rank in the original sentence, the authors introduce "positional encodings", which are used along with embeddings, so that the model gets a sense of where which word occur in the sentence. Having no need to process a sentence sequentially provides some benefits. Recurrent Neural Networks read a piece of text sequentially (from left to right and/or right to left). This means that a given word cannot be processed before previous words are. This is a problem when pieces of text are very long, as processing a full sentence might take a lot of time. Replacing LSTMs with CNNs [67] already improved the processing time. In the Transformer architecture, each attention head among a given layer can be processed in parallel. On a modern GPU, which is capable of running a lot of operations in parallel, this makes the training of transformer easier. Because the training part can now be done "efficiently" using parallel computing, transformers are constantly pushed to the limit, with a number of parameters growing exponentially (see Figure 2.5).

BERT One famous transformer is BERT [33]. When it was released, its performances yielded an outstanding 7.7% increase of accuracy on the General Language Understanding Evaluation (GLUE [142]) benchmark. Unlike language models which

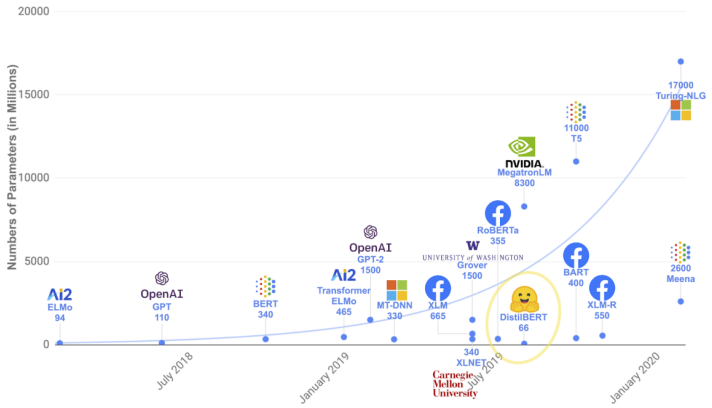


Figure 2.5: The exponentially increasing size of transformer models, taken from a blog post: `https://blog.tensorflow.org/2020/05/how-hugging-face-achieved-2x-performance-boost-question-answering.html`.

are trained to generate next tokens, BERT only uses an encoder. It was trained using large textual corpus on two unsupervised task: Masked Language Modeling and Next Sentence Prediction. In the former, the algorithm must learn to retrieve words of a given sentence which have been masked. The latter is a task where it is given two sentences A and B, and must decide whether or not B is the next sentence of A. Architecture-wise, it is composed of 12 MHA layers, instead of 8 in the original transformer paper [137]. The popularity of BERT hinted researchers to look for variants of this architecture, by optimizing the training part [88, 74], making a compressed version of the model [63], or even by training a language-specific variant [93, 77].

Other applications While transformers were originally designed for NLP tasks, their unique self-attention mechanism can be used in many other machine learning tasks. For example, transformers have been adapted to the Computer Vision field, where they have shown promising results in object detection [20], image generation [109], or image classification [39]. In speech recognition, transformer-based solutions [90, 35, 166] not only found application in both hybrid and end-to-end systems but also turned out to be better than many other modern solutions!

With the NLP field shifting to the transformers era, we will later study how

Figure 2.6: Example of user utterances labeled into intents (colors)

switching to this kind of models affect the performance of Few-Shot learning methods 2.4.2. For now, let's define the task that we are interested in solving.

2.3 Intent Detection

Intent detection – or intent classification, intent recognition – is a central part for task-oriented dialogue systems. Given a user query, it consists in finding the intent the user expresses in his query. This task can be seen as a sub-case of the more general task of text classification, where the goal is to assign one or more categories to a given piece of text. For example, new articles can be organized by topics [75, 4], tweets can be organised by user location [60] or other socioeconomic status [1], mentions can be organized by sentiment [160, 107], and so on.

While each one of those tasks might be related to text classification, they are quite different in practice. In this section, we will describe the case of intent detection, which is the task we will focus on in this thesis.

2.3.1 Intent detection vs slot filling

Intent detection is the task of finding a user's intent based on a query from this user. For example, if the user says "What's the earliest flight from dallas to houston?", then the associated intent might be `ask_flight_info`. However, using just this intent, we do not have enough information to answer the user: we also need to find

the associated **slots**. Indeed, in order to answer the user accordingly, we need to understand he is looking for flights from Dallas to Houston, and that he is looking for the earliest flight as possible. Once understood, those slots are used to query a database and find the flight the user is asking about. This example is exposed in Figure 2.7.

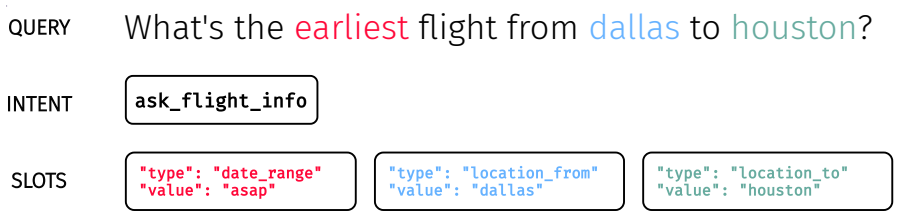


Figure 2.7: Intents and slots of a given user utterance

Slots are specific to each intent and might not be considered for some other intents. For example, if the user asks "What's the weather like in Paris?" and we correctly categorise its query as `ask_weather`, then the model which finds the according slots will not be looking for the `location_from` and `location_to` slots, as those are not relevant in that case. Instead, the model will be looking for location and date-related slots, for example. This makes intent detection and slot-filling two closely related tasks. While both tasks can be solved in a two-way process (first finding the intent, then filling the slots), best performances are reached when both tasks are solves simultaneously [55, 85, 159]. As slots depends on intents, we also have intents which depend on slots. This simultaneous dependency between the two tasks which we are trying to solve makes a joint model prone to such task.

While intent detection and slot filling are closely related, the content of this thesis will only be focused on the intent detection part.

2.3.2 A special text classification task

Typos Intent detection is about assigning a label to a user's query. Because the query comes from a user, some difficulties might be involved. For example, typos occur more frequently in a tweet than in a Wikipedia article (even a former president of the United States makes typos on Twitter[2]). If a given word, written with

[2]https://www.businessinsider.com/trump-typos-spelling-tweets-unpresidented-2017-4

typos, was not seen during training, then it might be difficult to get a reasonable representation of such word. Solution to overcome this problem have emerged, either by analysing sub-words [66], using WordPiece [123, 33, 28], or even a more recent study using a token-free model relying on bytes [152].

Informal language User-generated content, like tweets, is usually less formal then other pieces of text which are often used in pre-training of language models, like news articles. As an example, FastText [66] authors have pre-trained word vectors for 294 different languages on Wikipedia[3], which is a platform where content is more rich and way more formal than standard user-generated content like tweets. Overall, if the domain used for training a language model differs too much from the domain of a given task, then the pre-trained representations might not be suited the task. Several studies on domain adaptation [115, 41] have shown that the impact of domain shifting can be quite high. To account for this domain shift loss, we will study how further fine-tuning language model on a given task's domain helps at solving task 3.2.2.

Texts are short One other particularity of intent detection is that it deals with very short texts. In practice, user queries contain about 10 words on average 2.5. This is very different from other text classification tasks like news article classification, where pieces of texts are much longer – usually a few hundred words long. In practice, the more information you have about a piece of text, the easier it is to put a label on it. When one have only a few words to work with, it really limits the amount of information that the algorithms can exploit.

Labels In intent detection, labels might be very close as well as very far to one another. For example, Table 2.1 presents different samples for two close labels of the BANKING77 [21] dataset, `top_up_failed` and `top_up_reverted`. The former represents a general issue for an attempted top-up, while the latter is about the bank reverting a successful top-up. While this slight difference might be exposed when comparing examples A{1,2} to B{1,2}, it is very subtle, and examples {A, B}3 are even more difficult. For A3, one could say that it belongs to the right class, at it exposes

[3] https://fasttext.cc/docs/en/pretrained-vectors.html

Label	Id	Samples
top_up_failed	A1	Can you tell me if my pop-up went through?
	A2	My top up is giving me problems. What is wrong with it?
	A3	Why was my top-up rejected?
top_up_reverted	B1	Do you know if my top-up has been cancelled?
	B2	Why was my top up cancelled?
	B3	Why didn't my top up work?

Table 2.1: Sample with close labels, extracted for the BANKING77 dataset

a *rejection*, not a *reversion* of the action. As for B3, one could argue that it could belong to either of those classes. Text classification tasks like sentiment analysis, question classification or even topic modeling rely on datasets which contain very few classes (see Table 1 of [58]). As demonstrated in Section 2.5, intent detection datasets usually contain a number of labels in a different order or magnitude.

2.4 Few-Shot Learning

In this section, we will describe the few-shot learning problem and the different ways to tackle it. Fist, we will introduce the meta-learning framework, where models learn on training tasks, and are evaluated on testing tasks, where testing labels have never been by the model during training. Then, we will look at the semi-supervised learning approach, where both labeled and unlabeled data are used in order to improve the model's performances. Finally, we will talk about different data augmentation techniques, which can be used to have a more robust model.

Few-shot learning is a particular case of machine learning when the amount of labeled data is scarce. As exposed in Figure 2.8, this particular case has gained a lot of traction in the last few years, and is still increasing exponentially. One reason behind this recent interest is the cost of acquiring labeled data. Indeed, it is both time and money consuming to annotate raw data from scratch. Ideally, we would like models to learn from very few samples, given the pain of the labeling task (as discussed in Chapter 1). In a way, we would like computers to be able to learn from few information, as human do. Deep learning models are often compared to a human brain [56, 3]: the artificial neural network architecture was specifically designed to

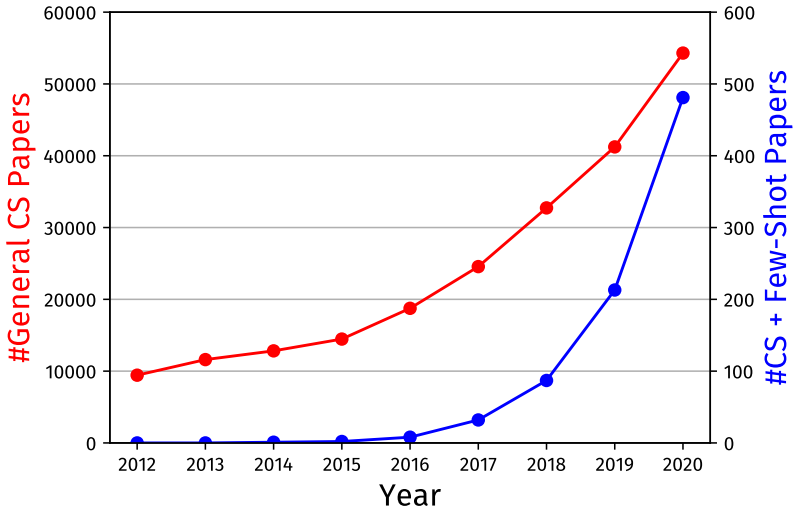


Figure 2.8: Total amount of "Computer Science" papers on ArXiv vs the same amount of "few-shot learning" related papers by year. Data has been gathered using ArXiv's advanced search [4] on the "Computer Science" category, using either no search term (red) or "few shot learning" (blue) as a search term.

mimic a human brain. However, machines do not learn the same way as humans do. Consider the following example: a person has never seen a horse in his entire life, and nobody has ever described it to him. There will be one day when this person will see a horse for the very first time in his life. After that, he will not need to see thousands of other horses to recognise this animal - he probably already seen other animals and his brain quickly transfer this knowledge for the horse he had never seen until that day. We can then say that the human brain does very well at adapting to a new task, and does not need a lot of examples for perform well on such task. As such, humans have *learnt to learn* – a process we will henceforth refer as meta-learning.

There are various ways of doing few-shot learning. The first one is about training a model using small amounts of labeled samples. At test time, the model is evaluated on its ability to predict the correct label among the classes it has seen during training, even though the model only has poor knowledge of such classes due to the low amount of data. Another way to look at the problem is by using a meta-learning framework, where the model "learns to learn". this is the framework we will describe in the next section. We will introduce this framework and its notations in the

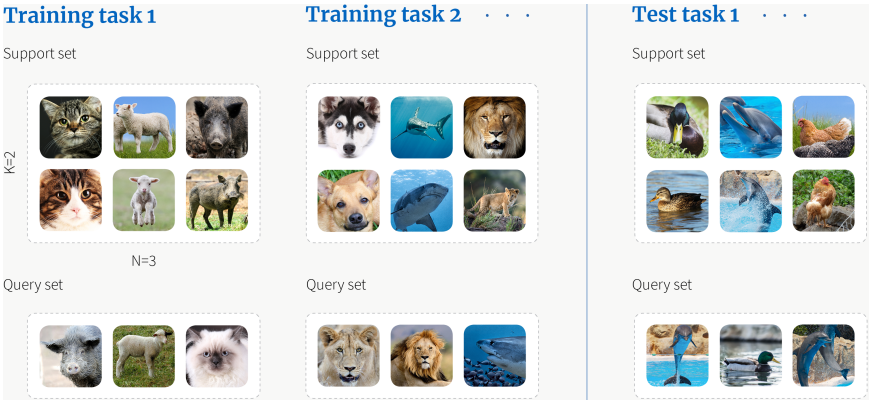


Figure 2.9: The meta-learning framework, taken from a blog post: `https://www.borealisai.com/en/blog/tutorial-2-few-shot-learning-and-meta-learning-i/`.

following section.

2.4.1 The Meta-Learning framework

In few-shot learning, the amount of training data is usually insufficient to perform a task. As such, one possible solution to overcome this problem is to gain experience from other similar tasks. By doing so, most few-shot learning approaches cast this problem as a meta-learning problem. Few-shot learning aims at optimising a model on training data, then evaluating it on testing data. In meta-learning, the model *learns to learn* on training tasks. Then, its learning capabilities are evaluated on testing tasks. Training and testing tasks do not contain the same labels, and might not even be in the same domain. The motivation behind this approach is that if we are able to discriminate between some classes at train time, then it might be easier to classify samples from previously unseen classes at train time. In other words, using a more intuitive example, the hypothesis is that if you can correctly differentiate dogs from cats, then it will be easier for you to make the difference between horses and cows after a couple of examples for those animals.

The meta-learning framework is illustrated in Figure 2.9[5]. First, labels are separated into train labels C_{train} and test labels C_{test}. To make the situation as realistic as possible, such sets are disjoint ($C_{\text{train}} \cap C_{\text{test}} = \varnothing$). Then, the experiments goes on as

[5]Note that some examples in this thesis will be illustrated in the image classification task. The reason is just that figures with images are much more intuitive than with text.

an episode-based process, with each time step corresponding to one episode. Here are the steps for creating a training episode:

- Pick C classes among C_{train}
- For each class $c \in C$:
  - sample K points belonging to this class. This will be the **support** set for this class,
  - sample Q points belonging to this class. This will be the **query** set for this class.

The episode will hence be composed of $C \times (K + Q)$ labeled points. Each episode will be used as a task to train the model, and mimics the few-shot scenario, including C classes with K examples for each. For this given episode, the model will have to predict the label of each query points based on the knowledge it can extract from support points. This framework is often referred to as a C-way, K-shot problem: we need to classify query points into C classes, given only K support points for each class. In practice, K is really important, and the larger K is, the better the model is expected to perform. Indeed, a model does better when it has more labeled samples to work with. In practice, K is usually picked in the $\{1, 5, 10\}$ set. When $K = 1$, this process is called one-shot learning, as there is only one sample for each class. By increasing the value of K, we can evaluate how the model would behave if it had a little bit more labeled data to work with. The number of query samples per class Q does not matter as much as K, as it only corresponds to the amount of points we are optimizing the model with at each step.

At each step of meta-learning, we update the parameters of the model based on its performances of a randomly generated episode. The loss value of this model will depend on its classification error on query points. The model is prompted with a different task at each time step, hence it must learn how to discriminate data classes in general, rather than a particular subset of classes. At test time, we want to evaluate the performances of the model, and this time we will select test tasks based on C_{test}, the set of test classes. Again, for each test task, we will evaluate the classification performance of query points, given the knowledge from support points. In the following, V^s (resp. V^q) will represent the vector representation of support points

(resp. query points) of a given episode. When unlabeled data will be used, V^u will denote vector representations of such unlabeled points.

2.4.2 Few-Shot Learning methods

In this section, we will introduce the various few-shot learning methods which will be used in this thesis. The different methods will be presented chronologically, from the oldest to the newest. Funnily, this chronological order is also the complexity order, as most recent methods are also more complex – in terms of number of parameters – than older ones. In practice, few-shot learning methods are directly plugged on top of embeddings. These methods actually learns in different fashion a point representative for each class – in a sense they quantifies the support points of an episode support set into a single representative point that minimizes the error on the episode query set. As such, when an episode ends, the encoder weights are updated.

2.4.2.1 Matching Networks

Matching Networks [139] were first introduced in 2016. While they were first applied in the computer vision field, they have also been successfully extended to NLP tasks, such as entity alignment [148], reading comprehension [161], or chatbot response selection [168]. An overview of how matching networks work is exposed in Figure 2.10. The idea of matching networks is to "match" points from the query set with points for the support set. To classify a query point x_q, we first compute its distances to all support points. The predicted label of $x_q \in V^q$ is the label of the support point $x_s \in V^s$ which is closest to x_q. In the original paper, the distance function used for matching networks was the cosine distance. However, this distance function can easily be swapped with another one, depending on which distance function better suits the task.

As the original paper's name indicates –*Matching Networks for One Shot Learning*–, Matching Networks were originally designed for one-shot learning. However, they can easily be extended to cases where there are multiple shots per class (i.e. $K > 1$). Such an extension was used in other works [26] where, instead of looking in a point-wise manner, authors think class-wise. To do so, they average the distance to all supports points of a given class to obtain an "average distance" to that class.

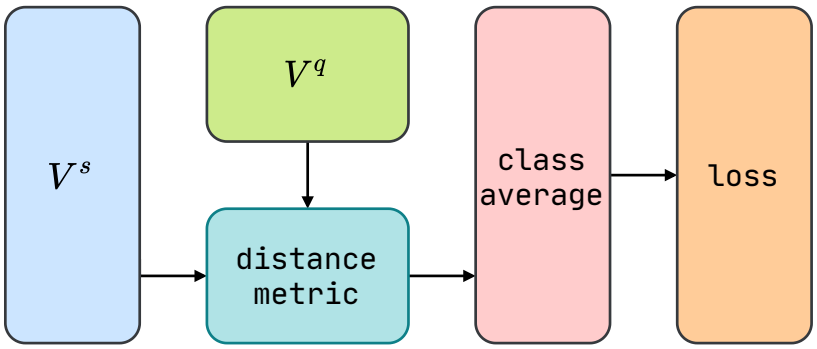


Figure 2.10: Matching Network

The class with the lowest average distance is the predicted label.

Apart from selecting the relevant distance function, matching networks do not introduce any hyper-parameter in their approach. Additionally, their method do not use any additional layer or learnable parameters. This simplicity makes them easy to understand and easy to implement. Now, we are going to describe the Prototypical Networks.

2.4.2.2 Prototypical Networks

Prototypical networks [129] were introduced in 2017, one year after matching networks. While also originally tested on computer vision tasks, they quickly made their way to the NLP domain, and were extended to tasks like text classification [132], or relation classification [48]. Prototypical networks, as the name indicates, is about creating "prototypes". For each class c of the current episode, a prototype p_c is be computed as the average embedding of all support points of this class. Then, to label a query point, we compute distances between this point and the various prototypes. The prediction of the query point is the label for which the prototype is the closest to the point. This process is illustrated in Figure 2.11

In the case of one-shot learning, the prototype of each class is just the one and only sample we have for this class. As a result, prototypical and matching networks are equivalent in that case. However, when we have more than one-shot, both methods differ. Indeed, in matching networks, we first compute distances and then average those distances to predict a query point's label. In prototypical networks, it's

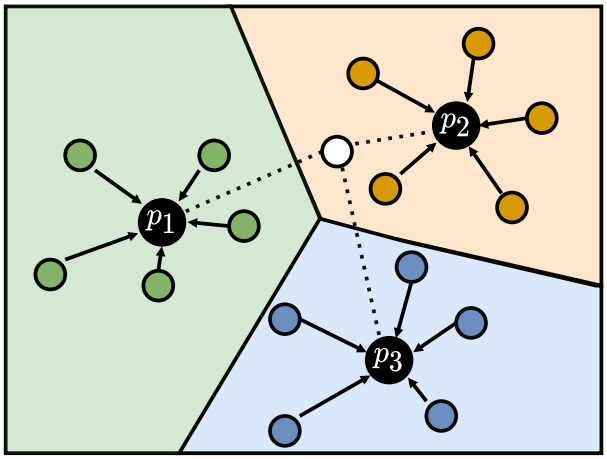


Figure 2.11: Prototypes $\{p_i, i \in \{1, 2, 3\}\}$ are computed as the average representations. To predict the label of a query point (white), we compare it to prototypes.

the reverse: classes are first averaged into a representative, then distances are computed. Another notable difference brought by prototypical networks is the use of the euclidean distance instead of the cosine one. While this change might seem small, authors have shown that the distance function plays a vital role and greatly impacts the performance of those few-shot models. We will later study the impact of such a choice in the Intent Detection task (see Chapter 3).

By construction, few-shot learning methods are struggling if the domain of the testing set – tasks the model is evaluated on – is too distant from the training domain – tasks the model is trained on. This is the reason why the proposed approach in [106] is to extend prototypical networks such that the prototypes for each class in source and target domains are close in the embedding space. Another approach [117] is to use unlabeled data along with labeled data in each episode to improve the model's robustness. To do so, after computing the prototypes, unlabeled data points are introduced in the embedding space. Soft-labels are predicted for those unlabeled data points. Then, using a soft k-means technique, prototypes are refined using both labeled data and soft labels. In other words, if an unlabeled data is close to a prototype, then we will train the model to bring closer this data point and the corresponding prototype. The idea behind this is that if an unlabeled data point is very close to a prototype, then it might be corresponding to this prototype's class. The flowchart for prototypical networks and this optional extension – which will be

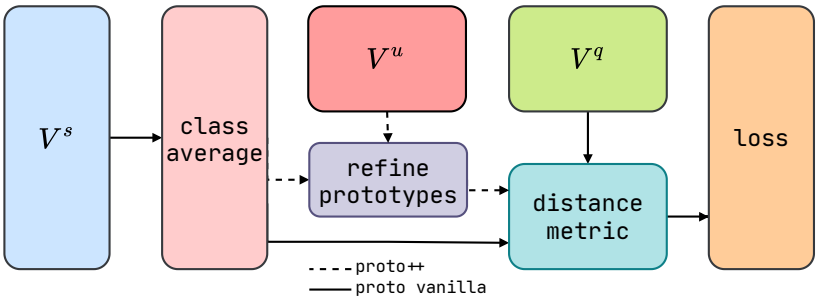


Figure 2.12: Prototypical Network

referred to as `proto++` in this thesis – is showcased in Figure 2.12. We will further study the improvements of this extension over the vanilla prototypical networks in Chapter 3

2.4.2.3 Relation Network

After both matching and prototypical networks were introduced, relation networks [133] came out in 2018. Like the previously introduced few-shot learning methods, it was originally designed for computer vision tasks, and was quickly adapted to NLP-related tasks, like sentiment classification [112], or entity recognition [24]

While other approaches [129, 139] focus on learning an embedder using a pre-define fixed metric (e.g. euclidean, cosine), relation networks do not base their concept on a pre-defined fixed metric. Instead, they aim at learning a relation module, which is capable of providing a relevant relation score between two data points. First, the embedder computes representation for both support and query points – this is the case for all few-shot methods we are studying here. Then, as in prototypical networks, representation of support points are averaged into a prototype for each class of the current episode. Finally, embeddings of query points are compared to prototypes using a *relation module*. The relation network's architecture is represented in Figure 2.13.

The whole contribution of relation networks lies within the relation module block. This relation module takes as input two representations which we need to compare, and outputs a score between 0 and 1. If the score is high, it means that both representations belong to the same category. This modules aims at replacing the pre-defined distance function which was used in previous works [139, 129]. Indeed,

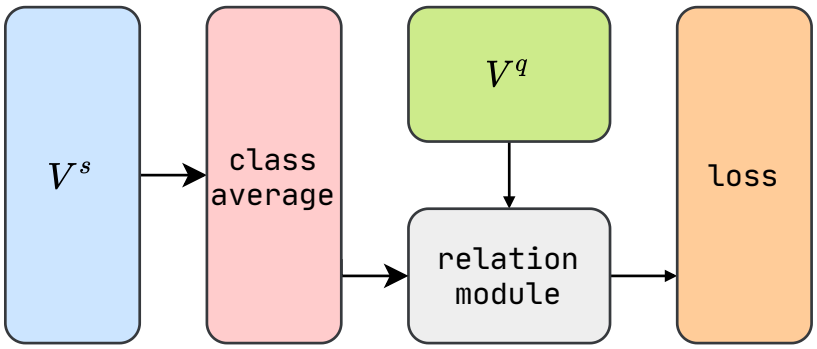


Figure 2.13: Relation Network

why would one use a pre-defined distance function when one could train its own? By using a flexible distance function approximator to learn similarity, relation networks learn a good metric in a data driven way and avoid having to manually choose the right metric (euclidean, cosine, or even Mahalanobis [32]). Fixed metric used in [139, 129] make the hypothesis that comparing representations element-wise is enough to solve the task. This hypothesis is quite strong, and make the whole model completely rely on the capacities of the embedder model to produce high-quality representations. With the recent advances in text processing using transformers, we will see how this hypothesis stands and how such models do when equipped with transformers.

The relation module's architecture is quite flexible. The main rule it must respect is taking as input two vectors, and giving a single score as the output. We will consider two relation module variants: a **base** one, used in the original paper, and the **NTL** one, introduced by other works [130]. In the following, $s_{i,c}$ denotes the relation score between query sample x_i (with embedding v_i) and class c (with, as an embedding, prototype p_c).

base In the original relation networks paper, they introduce a simple relation module. To compare a given query point x_i and a prototype p_c, they concatenate both vectors, and apply a small feed-forward neural network composed of two linear layers, with a ReLU activation function in between. The formula for this given relation module is described in Equation 2.1, where $C(\cdot,\cdot)$ denotes the concatenation operator, $f(\cdot)$ denotes the ReLU activation function, and w, M_1, M_2 are learnable

parameters. In this equation,

$$s_{i,c}^{\text{rel-base}} = \langle w, M_2 \left(f \left(M_1 \left(C \left(v_i, p_c \right) \right) \right) \right) \rangle \tag{2.1}$$

NTL Introduced by [130], the **N**eural **T**ensor **L**ayer relation module uses intermediate learnable matrices $M_t \in \mathbb{R}^{d,d}$ to model the relation between support vectors and prototypes. The idea is that different matrices may catch different similarities between the vectors. This relation module computes one relation score for each matrix M_t, then aggregates those scores into a final score representing the relation between sample x_i and class c. Computation of this relation score using this particular relation module is obtained using Equation 2.2, where w is also learnable parameter.

$$s_{i,c}^{\text{rel-ntl}} = \left\langle w, z_{i,c}^{\text{rel-ntl}} \right\rangle \quad , \quad w \in \mathbb{R}^h \tag{2.2}$$

$$z_{i,c,t}^{\text{rel-ntl}} = f\left(\left(v_i \right)^T M_t p_c \right) \quad , \quad t \in [\![1, h]\!] \tag{2.3}$$

Using this relation module, increasing the number of matrices h also increases the amount of information the model is able to capture in order to model the relation score. However, it also introduces a lot of additional learnable parameters. In a context of few-shot learning, where the amount of training data is quite limited, this might become a problem. Indeed, as it is given more chances to fine-tune itself, the model could quickly over-fit on the training task, leading to poor results on the testing tasks. We will see how this few-shot learning approach will perform on the intent detection task in Chapter 3

2.4.2.4 Induction Network

Induction network [49] were introduced in 2019. This time, it was first introduced in the NLP field, on a text classification task. Because methods in NLP and computer vision are often borrowed from one another, they were also adapted to image processing tasks like hyperspectral image classification [47] or abstract visual reasoning [59]. In the original paper, authors challenge the idea of representing a given class as the average representation of its points, which was the de-facto method [129, 133] until then. In their contribution, they introduce an induction module, which has the role

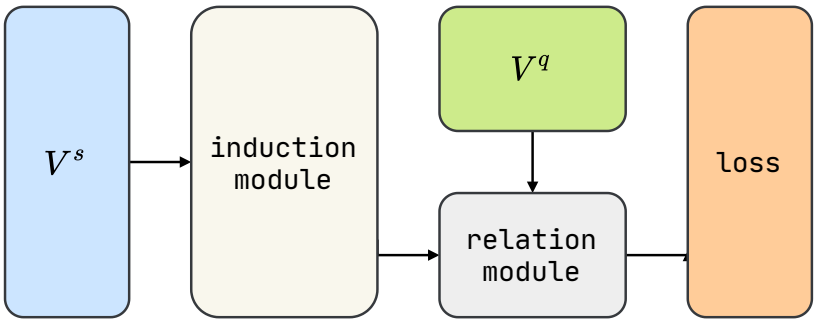


Figure 2.14: Induction Network

of computing a class representation from a set of representations of points of this class. The architecture of induction networks is displayed in Figure 2.14

To derive class representations, the induction modules uses the dynamic routing algorithm [121], explained in Algorithm 1. The squash function, defined in Equation 2.4, is a non-linear squashing function, which decreases the magnitude of a given vector. In the dynamic routing algorithms, learnable parameters W and b are introduced. Those parameters are shared across all classes, so that the model is flexible enough to handle new classes at test time.

Algorithm 1 Dynamic Routing Induction

Input: Support vectors $\{v_{i,c} \mid i = 1, \dots, K\}$ of class c
Output: Class representation p_c

Initialise logits of coupling coefficients $b_{i,c} = 0 \ \ \forall i = 1, \dots, K$
for $i = 1, \dots, K$ **do**
    $\hat{v}_{i,c} = \text{squash}(W v_{i,c} + b)$
for *iter* iterations **do**
    $d_c = \text{softmax}(b)$
    $\hat{p}_c = \sum_i d_{ci} \cdot \hat{v}_{i,c}$
    $p_c = \text{squash}(\hat{a}_c)$
    for $i = 1, \dots, K$ **do**
        $b_{i,c} = b_{i,c} + \hat{v}_{i,c} \cdot p_c$
return p_c

$$\text{squash}(x) = \frac{\|x\|^2}{1 + \|x\|^2} \frac{x}{\|x\|^2} \tag{2.4}$$

This dynamic routing algorithm was first proposed in Capsule Networks [121],

and allowed the network to learn the invariants within each one of the different classes. In a few-shot scenario, this dynamic routing allows to learn generalised class-level representation from the few available samples, hopefully in a better way than using the average representation.

The relation module used in induction networks is the NTL one, previously described in the relation network section. Compared to all other previously described approaches, induction networks are the having the most learnable parameters. As we mentioned earlier, adding too much learnable parameters could lead to problems especially in the few-shot case, as we are giving more and more over-fitting chance to the model.

Even though few-shot learning models are designed for tasks where the amount of labeled data is limited, other methods also come handy in such situations. In the following section, we will describe some semi-supervised learning approaches which can be used to overcome the data scarcity.

2.4.3 Semi-Supervised Learning

Unsupervised learning is the case where we want to train models using only unlabeled data. It remains one of the main challenges in machine learning, both in computer vision [113] and NLP [29]. If models could learn by themselves, needing little to no human supervision – this supervision can be in the form of labeled data –, then it would be a huge milestone towards solving general artificial intelligence. While unsupervised learning and general artificial intelligence are far from being solved, researchers have made a great progress in the semi-supervised learning [22, 170] field. This particular area of research lies in between supervised learning, where models are trained on top of labeled datasets, and unsupervised learning, where no labeled data are available.

Semi-supervised learning, that can be applicable to intent detection, can be approached in two different ways. The first one is to use some predictor to make assumptions on unlabeled data in order to associate them to a pseudo-label that can be later use in a supervised training scenario. An simple example of such an approach is as follows: train a model using the available labeled data, then use that model to predict pseudo-labels for the available unlabeled data, and ultimately use both labeled and pseudo-labeled data to train a better model in a supervised way [16, 167].

Note that in the literature, pseudo labels are also referred to as weak labels, noisy labels, or even proxy labels. The previous semi-supervision learning example is not straightforward as it contains multiple steps (that is in this case, training several models sequentially). Therefore, the second way to perform supervised learning is to aim at solving the task in an end-to-end manner [150], using both labeled and unlabeled data at the same time to train a robust model. In this section, we first review the different approaches using pseudo-labels (Section 2.4.3.1), then we focus on UDA [150], an end-to-end approach for training a model using semi-supervision (Section 2.4.3.2).

2.4.3.1 Pseudo-labeling

Pseudo-labeling is the action of using a trained model to produce pseudo-labels on unlabeled data. Pseudo-labels are not real labels: they are just a prediction of a given model on unlabeled data. As such, they might be wrong, hence a pseudo-labeled sentence is not as useful as labeled one. However, depending on the scenario, acquiring unlabeled data may be cheap, and therefore a large amount of unlabeled data with pseudo-labels might help at solving the task. In the end, the ultimate goal is to make a positive use of unlabeled data. While this sounds easy to do, mind that such a technique is a double-edge sword. If pseudo-labels are wrong, first they make the training much harder for the model since (pseudo)-labeled data may be incoherent and a good hyper-plane to separate data harder to find, and second, the resulting accuracy of the model can be much lower than the one trained only on labeled data without the (wrong) pseudo-labels. It is therefore important that this process brings significantly more correct pseudo-labels than wrong ones. In a sense, we want to train a model using both labeled and unlabeled data, and we want this model to be better than the model which is just using labeled data to train on (otherwise the unlabeled data is just useless).

Self-training One approach to using unlabeled data along with labeled data is self-training [157, 96]. Being one of the earliest approaches to semi-supervised learning, it is also the most straightforward method where a model learns from its own predictions. In self-training, a model is first trained on top of labeled data. Then,

it computes pseudo labels for each unlabeled data point. Finally, the model is retrained using both labeled and pseudo-labeled data. Ideally, only the most confident pseudo labels will be used. If a pseudo-label is not confident enough, then training on top of it might introduce too much noise and hence, confuse the model. To overcome this issue, a threshold parameter t is introduced, and pseudo-label are retained only if their predicted probability is higher than this threshold. An overview of how self-training works is available in Algorithm 2.

Algorithm 2 Self-training

Input:
Labeled data $L \leftarrow \{(x, y) \in L\}$
Unlabeled data $U \leftarrow \{x \in U\}$
threshold $t \in (0, 1)$
Output: Trained models m
repeat
Train model m on L
for $x \in U$ **do**
if $\max_i \ [m(x)]_i > t$ **then**
$\hat{y} = \text{argmax}_i \ [m(x)]_i$
$L \leftarrow L \cup \{(x, \hat{y})\}$
until L stops increasing

While this self-training approach is easy to understand and quite straightforward, studies have shown it did not perform very well, for example on the sentiment classification task [6]. The main drawback of this method is that once pseudo-labels are integrated into L, the set of labeled samples, then the model cannot correct such labels at any point in time. This means that if, at some point, the model has assigned an incorrect pseudo-label with a confidence higher than threshold t, then it will have to deal with it until the end of the process. This can lead to a model completely shifting from the original objective, as it will learn of top of wrong labels, and things will get worse as the process goes on. To overcome this issue, variants like co-training [16] and tri-training [167] were introduced. We will develop those methods in the next paragraphs.

Co-training In co-training [16], two models are jointly used. In traditional self-training, the model learns from its own predictions in an interactive manner. In co-training, two models are trained, each one on half of the labeled data (those data sets do not intersect). Then, each model computes its predictions on the unlabeled data. Pseudo-labels are retained if and only if one of the model is confident for this prediction, and the other one is not. Then, those pseudo-labels are added to the labeled dataset of the other model. This co-training algorithm is detailed in Algorithm 3

Algorithm 3 Co-training

Input:
Labeled data $L = \{(x, y) \in L_1 \cup L_2\}$ with $L_1 \cap L_2 = \emptyset$
Unlabeled data $U = \{x \in U\}$
threshold $t \in (0, 1)$
Output: Trained models m_1, m_2
repeat
Train model m_1 on L_1
Train model m_2 on L_2
for $x \in U$ **do**
if $\max_i \ [m_1(x)]_i > t$ **and** $\max_i \ [m_2(x)]_i < t$ **then**
$\hat{y}_1 \leftarrow \text{argmax}_i \ [m_1(x)]_i$
$L_2 \leftarrow L_2 \cup \{(x, \hat{y}_1)\}$
else if $\max_i \ [m_1(x)]_i < t$ **and** $\max_i \ [m_2(x)]_i > t$ **then**
$\hat{y}_2 \leftarrow \text{argmax}_i \ [m_2(x)]_i$
$L_1 \leftarrow L_1 \cup \{(x, \hat{y}_2)\}$
else
skip x
until L_1 and L_2 stop increasing

By using this mechanism of two models with separate training sets, model m_1 is not trained on top of its own predictions, but on predictions for which it is not confident and the other model (m_2) is. Co-training has been extended to democratic co-training [169], where multiple models are trained in parallel. In this extension, pseudo-labels are integrated in the training data only if a majority of models agree on the pseudo-label. The goal here is to use models which are the most complementary to each other.

Tri-training As its name suggest, tri-training [167] uses three different models. Additionally, to create diversity, each model m_j is trained on a subset S_j of the labeled

data. The idea is the same as in co-training: the three models must be as diverse as possible, in order to infer treat data in different ways. The procedure of tri-training is described in Algorithm 4. After models are first trained once on their respective samples, pseudo-labels are added to the training set of a given model if and only if the two other models agree on such pseudo-labels. Like other procedures described earlier, this process stops when models stop evolving. In the end, we are left with 3 trained models. To make a prediction for a new data point, we predict its label according to the 3 different models, and a majority vote is operated to derive the final prediction. Unlike co-training, tri-training does not rely on a threshold t to decide whether a pseudo-label should be considered. This is a nice improvement as having to find the correct values of hyper-parameters is hard, especially when the amount of labeled data is scarce.

Algorithm 4 Tri-training

Input:
Labeled data $L \leftarrow \{(x, y) \in L\}$
Unlabeled data $U \leftarrow \{x \in U\}$
Output: Trained models m_1, m_2
for $j \in [\![1, 3]\!]$ **do**
Sample S_j from L
Train model m_j on S_j
repeat
for $j \in [\![1, 3]\!]$ **do**
$L_j \leftarrow \emptyset$
for $x \in U$ **do**
if $\text{argmax}_i\ [m_k(x)]_i = \text{argmax}_i\ [m_l(x)]_i\ (k, l \neq j)$ **then**
$\hat{y} \leftarrow \text{argmax}_i\ [m_k(x)]_i$
$L_j \leftarrow L_j \cup \{(x, \text{argmax}_i\ [m_k(x)]_i)\}$
Train model m_j on $L \cup L_j$
until none of the pseudo-labeled sets $\{L_j \mid j \in [\![1, 3]\!]\}$ change

Tri-training was introduced in 2005, a long time ago in the field of Computer Science. Still, recent studies [120] have shown that it still remains a very strong baseline in the semi-supervised learning domain. This strong baseline was further extended in many forms: adding model disagreement [105] for part-of-speech tagging, asymmetric tri-training [122] for unsupervised domain adaptation, or adapting tri-training to the multi-task case [120].

Those different processes aim at solving the semi-supervised learning case using multiple models which interacts between each other. However, as a completely different approach, one would want to solve this problem in an end-to-end manner. In the next section, we will introduce UDA, an end-to-end system to solve semi-supervised text classification.

2.4.3.2 UDA

UDA [150], standing for **U**nsupervised **D**ata **A**ugmentation, is a recent end-to-end semi-supervised approach to the few-shot learning problem. The first step of UDA is to generate variations – or augmentations, those words are inter-changeable – of unlabeled data using data augmentation techniques. Then, the model will predict the label of each unlabeled data point and their augmentations. An unsupervised consistency loss is computed as the KL-Divergence between original data predictions and predictions for augmentations. This way, the model is trained on its ability to give the same predictions to a given point and its augmentations. This unsupervised loss is combined with a regular supervised cross-entropy loss into a final loss which is optimised by the model. An overview of the UDA framework is showcased on Figure 2.15

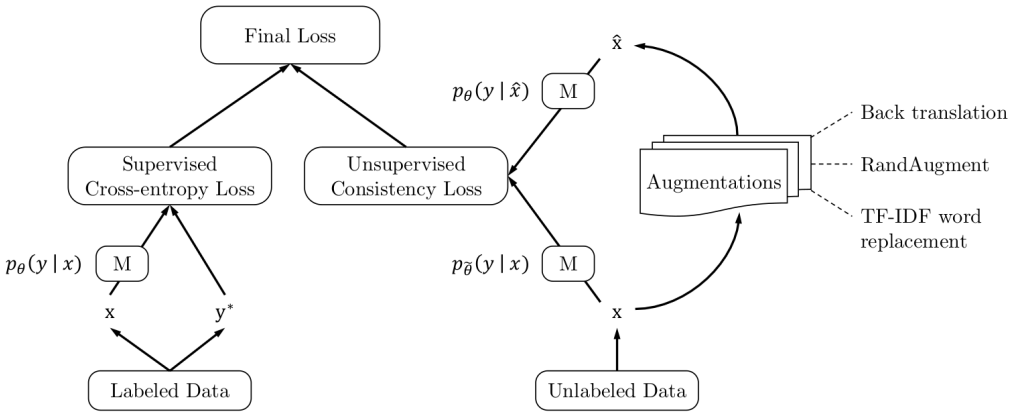


Figure 2.15: The UDA framework. M is a model that predicts a distribution of y given x. The figure is taken from the original paper [150].

By integrated unlabeled examples in the training process, UDA increases the smoothness of the model. In a sense, UDA trains the model to be insensitive to

noise, making it more robust to changes in the input sentences. In the few-shot scenario, as the knowledge from labeled data point is highly limited, using methods to increase the model's robustness is of paramount importance. Indeed, the smaller the amount of labeled data, the more prone the model will be to overfit. As this framework is not model-specific, authors of this semi-supervised method experimented on both text and image classification tasks, leading to very good results on both. For the text classification tasks, authors have experimented with two different data augmentation techniques: Back-translation and Word replacing. We will further detail the different existing data augmentation techniques in Section 2.4.4.

2.4.4 Data Augmentation Techniques

Data augmentation has recently seen a surging interest in the NLP domain. With the increasing easiness to use off-the-shelf pretrained models (highly due to the HuggingFace [146] python library), more and more use cases for such models are explored on a daily basis. However, many of those tasks face the problem of acquiring a significant labeled dataset. In such scenarios, data augmentation can have a huge impact, as it improves performances of models [128]. Recent surveys [44, 10] exposed the various data augmentation approaches specific to the NLP domain. An overview of the taxonomy of such data augmentation methods, directly taken from one of those surveys, is displayed in Figure 2.16. In this hierarchical grouping, we separate data augmentation methods in two main categories: feature space and data space methods. Methods operating in the feature space aim at transforming the feature representation of data. In [73], the authors add random multiplicative and additive noise to the hidden space of the transformer. By artificially adding small perturbations to the hidden representation of a given data point, the authors train the model at making abstraction of such small noise. Instead of operating at the feature level, another family of data augmentation method aim at adding perturbations in the data space. Such methods range operating at the document level, and up to the most fine-grained level (character level). Several approaches [43, 11] to character-level apply operation like randomly swapping letters, complete randomization of a word, or even character insertion. Those operations were also applied to token-level

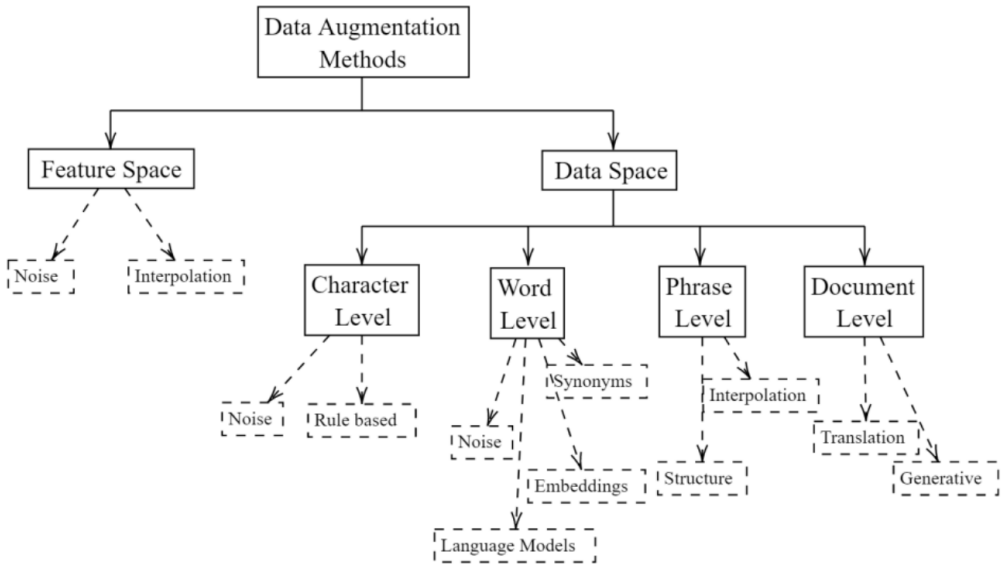


Figure 2.16: Taxonomy of different data augmentation methods. Taken from [10]

data augmentation [145], along with other token-specific data augmentation techniques like synonym replacement [69, 8]. At the document level, one familiar approach is to use back-translation [124] (while "round-trip translation" is the correct term for this approach, we will name it "back-translation" as it is most commonly used in the literature).

In what follows, we describe the two major and popular data augmentation techniques so far: Back-translation [124] (Section 2.4.4.1) and **E**asy **D**ata **A**ugmentation [145], which is usually referred to as EDA (Section 2.4.4.2).

2.4.4.1 Back-translation

Back-translation [124] is a popular translation-based data augmentation technique. This method translates a given piece of text into another language – which we will refer to as *pivot* language –, then back into the original language. To perform such task, one need to have two mirror translation models at hand. While originally used for machine translation tasks [124, 83], this method was also used in text classification [150], question answering [89], text summarization [42]. By using different pivot languages, we can generate different augmentations for a given data point.

Recently, NLP models have been more and more easy to use and share, partly

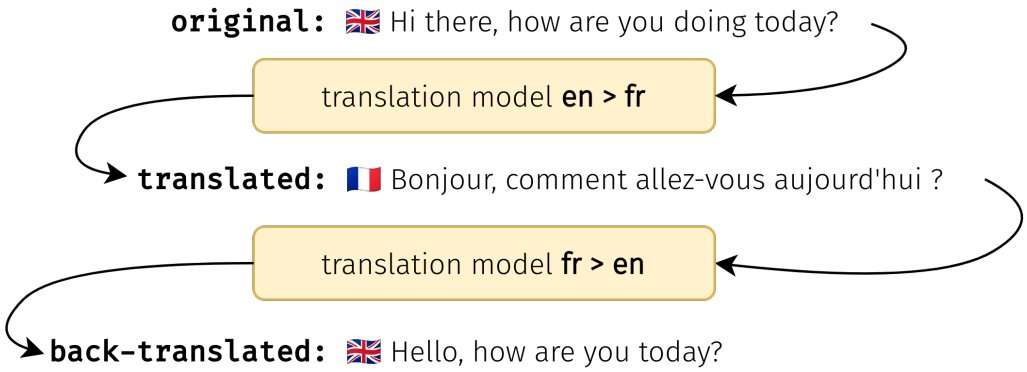


Figure 2.17: Back-translation example using French as a pivot language.

thanks to huggingface's `transformers` library. With more than 10,000 models available on their model hub[6], one could easily find pre-trained translation models in order to perform back-translation.

While back-translation operates at the sentence level by using machine translation models, some other approaches to data augmentation operate at the token-level. In the next section, we will describe EDA, one of those token-level data augmentation method.

2.4.4.2 EDA

EDA [145], standing for **E**asy **D**ata **A**ugmentation, is a token-level data augmentation method introduced to boost performances of text classification models. In EDA, data augmentation is performed by perturbing a given sentence. To do so, authors use 4 different perturbation techniques, which we describe below. For all methods which require to find the synonym of a given word, authors use WordNet [102] from the nltk library[7].

Synonym Replacement (SR) Randomly choose words from the sentence, which are not stop words. Replace each word by one of its synonym, chosen at random.

Random Insertion (RI) A random synonym of a random word in the input sentence in inserted at a random position.

[6] https://huggingface.co/models
[7] http://www.nltk.org/howto/wordnet.html

Method	Sentence
-	I have lost my password
SR	I have forgot my identification
RI	I have identification lost my password
RS	Have lost my I password
RD	I lost my password

Table 2.2: Example of data augmented sentence using EDA operations

Random Swap (RS) Two random words from the sentence exchange positions.

Random Deletion (RD) Each word is the sentence is randomly removed, according to a probability p.

We generated examples of sentences derived from the EDA methods in Table 2.2. All in all, methods performed by EDA are aimed to be complementary to each other. Random swaps completely changes the order of words in a sentence, often resulting in a sentence which is not grammatically correct. Adding synonyms using SR or RI also enrich the dataset, as it adds new words which might not be present in the training data. Last but not least, random deletion also changes the structure of a given sentence, and often yields a non-correct sentence. One of the many strengths of EDA is the ability to control the amount of perturbation applied to a given sentence. Indeed, if one want more perturbations, one can increase the amount of tokens which are replaced by their synonyms, or increase the amount of random swaps. Increasing the amount of perturbation directly increases the difficulty of the task, because more perturbed augmentation are more distance to original sentences. In practice, this control mechanism is important, as the amount of perturbation we want to introduce in order to benefit the most out of data augmentation can vary from one task to another.

In the original paper, EDA has shown great results on 6 different text classification tasks. Results are more impressive the lower the amount of training data, as those are situations where many words important to a given classification task might not have their synonyms present in the training set. As this training set increases in size, one could argue that the chances of having synonyms present in other training samples increases. Further research [144] have successfully applied EDA in a curriculum learning [12], where the difficulty of the classification task artificially

increases through time. In this research, authors voluntarily increase the difficulty of the task by increasing the amount of perturbed tokens in the data augmentation step, showing great results in the few-shot learning scenario.

In the following section, we will introduce all the datasets we will experiment with in this thesis.

2.5 Datasets

In this section, we will describe the different datasets we will use in all our experiments. Since the experiments presented in this thesis may differ depending on the relevance of each datasets or their availability at the time of the experiment, I decided to regroup their presentation in this preliminaries for practical reasons and for the sake of a single point of reference about datasets for the reader.

All datasets are in English. First, we will introduce datasets which are about the general task of text classification. Then, we will describe the intent detection datasets, which is the task we are the most interested in. We summarize the main statistics of the different datasets in Table 2.4.

2.5.1 Text Classification

Text classification is the task of classifying texts into categories. Given a piece of text, the goal is to put the correct label on it. The task of text classification is very large, and can take many shapes: detecting fake news [18], finding the sentiment in tweets [2], or detecting abusive language [104]. Each of those sub-tasks have in common the fact that they belong to the text classification category, and some model architectures might be suited for many of those sub-tasks. However, it is not always granted: if a model performs well on the sentiment analysis task, it is not certain that such model, trained on another text classification task, will perform well. We therefore usually include in our intent detection experiments some limit-testing scenarios with text classification, for which texts are usually much longer yet with fewer issues (less issues, out of vocabulary words, no emojis, etc.). We introduce in what follows the text classification datasets we used in our experiments.

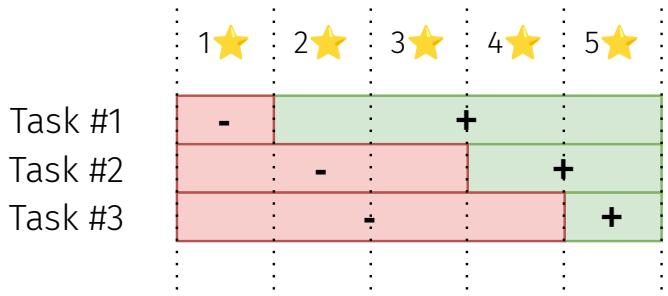


Figure 2.18: ARSC Meta-learning task construction. The (3+) vs (2-) task is missing, as in [158].

2.5.1.1 ARSC

The Amazon Review Sentiment Classification (ARSC) dataset [15], as its name suggest, is a sentiment classification dataset. It was created using reviews of product on the Amazon website. Each review consists of a rating (from 1 to 5 stars), a review text, a product type, and some additional metadata – which we will not use. This raw dataset has been organised for the meta-learning task by multiple authors [158, 49]. In their work, authors consider 23 different product types, from *Musical Instruments* to *Electronics*. For each product type, they create three binary classification tasks, each one having a different threshold on the rating. More precisely, each task has a fixed threshold t, and consists in telling whether a given review text is positive (rating $\geq t$) or negative (rating $< t$). The three thresholds considered for the three tasks are $t \in (2, 4, 5)$. With 23 product types and 3 threshold values, this yields $23 \times 3 = 69$ binary classification tasks to work with. To avoid any overlap between training and testing tasks, training and testing data are separated domain-wise. That is, the testing tasks consists of $12 = 3 \times 4$ binary classification tasks from 4 domains: *Books, DVD, Electronics, Kitchen*. By doing so, authors ensure there are no text reviews which belong to both training and testing tasks. More importantly, it comes close to the real-life scenario, where new domains are added and not much reviews are available to train a model on.

2.5.1.2 R8

R8 is a news articles dataset, and a subset of the more famous Reuters-21578 [81] dataset. It was originally collected and labeled by Carnegie Group, Inc. and Reuters,

Class	# Samples	Examples
earn	3915	mercantile stores co inc th qtr shr dlrs vs dlrs
acq	2285	ibc acquisition gets shares in tender to buy pct
crude	366	marathon to raise crude prices cts bbl tomorrow wti to dlrs
trade	312	volcker says more stimulus abroad needed for adjustments in trade balances
money-fx	278	bank of france buys dollars at paris fixing dealers
interest	247	royal bank of canada lowers prime rate to pct down
money-supply	141	u k september m rises pct m up pct bank of england
ship	141	pentagon says u s warships begin escorting gulf tanker convoy south from kuwait

Table 2.3: Classes of the R8 Dataset, with some examples.

Ltd. From Reuters-21578 to R8, only the 8 most common classes are retained. Details about the class repartition as well as some examples are showcased in Table 2.3. Following the work of previous authors [165, 156] on the subject, we will experiment using only those 8 most common classes. This process ensures that we have enough samples per class, and still yields an imbalanced dataset, as the most (resp. least) represented class contains 3,915 (resp. 141) samples. As we are following the meta-learning framework, we will split those 8 classes into train (3), validation (2), and test classes (3) in our experiments.

2.5.2 Intent Detection

Now, we will introduce the intent detection datasets we will use in the different experiments. We consider the DialoGLUE benchmark [97], a set of natural language understanding benchmark for task-oriented dialogue, which contains three datasets for intent detection: `Banking77`, `HWU64` and `Clinc150` – the three datasets were already available prior the release of DialoGLUE. We also consider a private dataset, which contains utterances from Meetic users when they talk to Lara, the dating chatbot.

2.5.2.1 Lara

Lara is the dating chatbot of Meetic. She talks to thousands of users on a daily basis: she provides dating tips, recommends profiles based on user criteria, helps users in their dating journey, and much more. Users interact with Lara by written language and not by voice. This can lead to a lot of typos as users type on their phone or computer's keyboards. Lara is available in multiple european countries, hence we have data in multiple languages. However, for the sake of comparison with state-of-the-art methods that are usually only based on English, we only use the English part of the dataset in our experiments. Over the years, we have collected a large dataset of conversations between users and Lara, which we have partially labeled. As of now, there are more than 300 classes in the Lara dataset. However, a lot of classes contain really small amounts of labeled data. In the experiments, we only retain the classes which contain more than 20 samples per class. This filtering still yields 96 classes to work with.

While some of this data come from raw user queries that we have manually annotated, this process has flaws. Indeed, if we want to add a new class corresponding to the question "What's the weather like today?", but no users have asked this question in the past, then we cannot just rely on past conversations to acquire annotated data. To overcome this issue, we also collected data from amazon mechanical turk, where we asked worker to reformulate some sentences with their own words. After acquiring such reformulations, we manually annotated them in order to verify that turkers [8] effectively reformulated the sentences they were prompted. As such, the dataset is composed of both real users queries, and reformulations from mechanical turk.

2.5.2.2 Snips

Snips [31] is a very popular intent detection dataset, and has been widely used in many research experiments [51, 25, 55]. Aside from the intent detection part, this dataset also contains slots to perform the task of slot filling. However, this related task is outside of the scope of this thesis, as we will only focus on the intent detection part. The dataset contains 7 different voice commands, like *PlayMusic*, or

[8] Turkers is the name given to Amazon Mechanical Turk workers

GetWeather. One particularity of this dataset is that it is very large with respect to the number of classes it contains: on average, there are about 2,000 samples per class. While this is not usually the case in practice, this dataset is still interesting to use, as its nature fit the real life scenario of popular voice command-based devices.

The low (7) number of classes of snips make it hard to use in a meta-learning framework. In such framework, train, validation, and test classes are disjoint. Hence, we will not be able to perform 5-way classification for example, as it would imply having at least 5 classes in each split. Still, we will experiment with this dataset in a C-way scenario, with C being small.

2.5.2.3 Clinc

The Clinc [76] dataset – or sometimes named OOS dataset – is a large intent detection dataset. It was artificially created for the out-of-scope prediction task. In a real life scenario, users do not always ask for things that task-oriented bots were designed for. For those specific cases where user queries are considered out-of-scope, it would be nice if the model would directly recognise the query as out-of-scope. To do so, authors introduce this novel out-of-scope prediction dataset. It is composed of 150 classes, each having 150 samples to work with. An additional class – the *out-of-scope* class– is also introduced. It contains many sentences (1,200) which do not fit the 150 other classes.

In this thesis, we are not particularly interested in the out-of-scope prediction, but more on the core intent detection task. Still, this dataset is interesting, as its large number of classes increase the difficulty of having a well separated embedding space. To adapt this dataset to our needs, we discard the out-of-scope class, and only keep the 150 other labeled classes to work with, as in other previous studies [97].

2.5.2.4 HWU64

HWU64 [151] classifies 25,716 user utterances into 64 intents. It features intents spanning across 21 domains (alarm, audio, audiobook, calendar, cooking, datetime, ...). When separating training, validation, and test labels, we ensure each domain is represented only in one set of labels. This prevents the model from learning domain classification instead of intent classification. Additionally, as intents in a single

domain are closer to each other, it makes the task a little bit harder.

2.5.2.5 BANKING77

BANKING77 [21] is a single-domain intent detection dataset. It is quite recent, as it was released in 2020. This dataset is composed of 77 different intents, which, compared to some other previously introduced datasets, makes it challenging. Having this much different intents better reflect the real-life scenario. Compared to HWU64 and Clinc150 which span multiple domains, this dataset is about one single domain: banking. Having 77 different intents in a single domain makes the task particularly hard, as intents are very fine-grained.

2.5.2.6 Liu

Introduced in [151], this intent detection dataset consists of 54 classes. It was collected on the Amazon Mechanical Turk platform, where workers were given an intent and had to formulate queries for this intent with their own words. Hence, this dataset was artificially created, as annotators were given instructions and formulated queries themselves. It is highly imbalanced: the most common class (`query`) holds $5,920$ samples while the least common one (`volume_other`) 24 samples.

2.6 Conclusion

In this chapter, we introduced the background knowledge required to capture the line of research explored in this thesis. First, we introduced the different methods to embed pieces of text into machine-readable vectors. From one-hot encodings to language models like transformers, we showcased how embedding techniques evolved in the past few years. Those embedding methods evolve quickly, and the best method today might be obsolete a few months from now. Still, by doing a historical review of the various methods, I hope it helps the reader capture where the works of this thesis are situated in NLP research. We also introduced the task that we attempt to solve in this thesis, intent detection. Additionally, we detailed the different methods of previous research to deal with Few-Shot Learning, a problem which is becoming more and more important as the number of use cases which can

Dataset	Task	Public?	#sentences	#classes	#sentences/class	#tokens/sentence
Snips	ID	✓	14,484	7	$2,069 \pm 21$	9.0 ± 3.2
Clinc	ID	✓	23,700	150	157 ± 85	8.5 ± 3.3
Liu	ID	✓	25,478	54	472 ± 823	7.5 ± 3.4
BANKING77	ID	✓	13,083	77	170 ± 33	11.7 ± 7.6
HWU64	ID	✓	11,036	64	172.4 ± 40.1	6.6 ± 2.9
Lara	ID	✗	8,142	96	84.8 ± 123	5.2 ± 3.6
R8	TC	✓	7,685	8	$961 \pm 1,303$	102 ± 117
ARSC	SA	✓	137,928	$27 \times 3 \times 2$	$999 \pm 1,468$	99 ± 115

Table 2.4: Classification datasets we will use in our experiments. ID stands for Intent Detection, and TC stands for Text Classification, SA stands for Sentiment Analysis. For ARSC, the number of classes corresponds to 27 (product types) $\times$ 3 (rating thresholds) $\times$ 2 (binary classification).

be answered by machine learning increase. Finally, we presented the different intent detection datasets that we will use in the various experiments of this thesis.

Chapter 3

Few-Shot Text Classification Reality Check

This chapter is based on the following publication [37]

> Thomas Dopierre, Christophe Gravier, Wilfried Logerais. "A Neural Few-Shot Text Classification Reality Check". In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL2021)*, pages 935–943, 2021.

3.1 Introduction

Text classification often requires a large number of mappings between texts and target classes. It is therefore challenging to build few-shot text classification models [49]. With the recent advances of transformer-based models [33, 146] along with their fine-tuning techniques [131], text classification has significantly improved.

As discussed in Section 3.3, few-shot methods based on these extracted text representations have been historically made of semi-supervision, especially thanks to pseudo-labeling [16, 99, 167], which aims at propagating known labels to unlabeled data points in the representational space. Such methods depend on the number of collected unlabeled data, which can also be costly to obtain [23], and also suffer from the infamous pipeline effect in NLP [135], as cascade processing tends to make errors accumulate. In order to address the hindrance of collecting unlabeled data, modern

approaches include unsupervised data augmentation techniques [150]. It consists of generating samples through well-established text augmentation techniques in Neural Machine Translation, such as back-translation [124, 40], and then use a consistency loss, training the classifier to assign the same prediction to all variations of the same sample text. While collecting new pseudo-labels can therefore be overcome by manipulating the dataset (especially using data augmentation techniques), the pipeline error accumulation effect instead calls for new neural architectures supporting scarcity of labeled data in an end-to-end fashion. Such end-to-end few-shot neural architectures for few-shot classification were discovered in image processing – it includes Matching Networks [139], Prototypical Networks [129] plus a follow-up known as Prototypical Networks++ [117], and Relation Networks [133]. Ultimately Induction Networks [49] is a meta-learning based method dedicated to few-shot text classification, supposedly the state-of-the-art. Nonetheless, it is important to stress that most of these neural architectures were originally devised to integrate image feature extractors. Despite both text and image relying on features extractors, a paragraph or sentence of few words hardly convey as much information as a full-fledged three-canals 600×400 image ($720,000$ numerical values intrinsically). It is therefore of the utmost practical interest to validate and compare if what works best for end-to-end few-shot *image* classification is the same for end-to-end few-shot *text* classification. Moreover, when applying these end-to-end few-shot models to text, two main system components are into action: the text feature extractor itself and the downstream part of the neural network that provides a learning strategy over few shots. If we want to compare these systems, we need to plug the same feature extractor (hopefully the best one, that is transformer-based currently) into each end-to-end model. For the time being, the literature on end-to-end few-shot text classification compare aforementioned techniques using a different text extractor for each system, which is the one available when the technique was discovered – these text encoding varying greatly (Section 3.2). From that point-of-view, it is hardly possible to conclude if the improvement over time in few-shot text classification is due to new end-to-end few-shot learning techniques or plainly to the significant advances made by text feature extractors that is included in this end-to-end learning scheme. The same applies to vectors metrics: one method can use the cosine and another the euclidean distance,

and that choice alone can impact conclusions made on the method being the state-of-the-art, although it could well rely only on the metric at work.

With respect to these understudied factors for few-shot learning, our first contribution described in this chapter are summarized as follows:

- We study the impact of fine-tuning a transformer language model on the intent detection datasets in an unsupervised fashion (using masked language modeling) before training the few-shot classification task,
- We revise different end-to-end neural architectures for few-shot text classification using the *same* transformer-based feature extractor and we empirically demonstrate that most of the recent advances usable for intent detection meta-learning is based on text encoders improvements and hardly from new neural meta-learning architectures themselves,
- We investigate how these re-implemented state-of-the-art solutions compete with very simple baselines found to be yet very competitive for few-shot classification in the field of image-processing,
- We introduce an evaluation framework based on a number of intent detection datasets which is significantly bigger than what is usually used as evaluation in seminal papers transposing each of these architectures from image to text classification,
- The entire framework used in this paper, including all the re-implemented methods plugged with up-to-date transformers, is provided as an open-source repository for further research.

In a nutshell, we will demonstrate that providing a transformer-based encoder to a previously obsolete few-shot technique makes it the state-of-the-art again, that standard baselines are surprisingly strong, and that Induction Networks, while performing well for binary sentiment classification, struggles to perform correctly in the most common setups of few-shot text classification.

3.2 Various sentence encoders

In previous works comparing few-shot text classification methods, sentence encoders were not always the same. This is a side effect of text encoding methods evolving rapidly in the last few years. While some research works [158] use a convolutional neural network on top of word embeddings, induction networks [49] use a Bi-LSTM instead. Those differences make the results hard to compare since they do not use the same method to convert sentences into vectors. Moreover, such encoders have become quite obsolete with the arrival of the transformers [137, 33] era, beating the state-of-the-art in most NLP tasks. In our experiments, in order to reduce the encoder method selection bias, and since it is now the state-of-the-art in many applications, we use a BERT [33] encoder, using models from the Hugging Face [146] team.

While BERT can be used as-is, further fine-tuning the language model on a task domain can greatly improve performances on the downstream task. This has been shown in Definition Classification [62], where the model is trained to tell whether a given sentence contains a definition or not. While this additional fine-tuning step had a positive impact for this task, it cannot be automatically generalised and we cannot conclude that it would be beneficial for any NLP task. To validate or invalidate this hypothesis, we tried to evaluate the quality of embeddings when using an off-the-shelf BERT model versus using a model which we have further fine-tuned on our custom domain.

3.2.1 Language Model Fine-tuning details

As a starting point for our transformer, we pick an off-the-shelf `bert-base-cased` model[1]. A large corpus of English data was used to train this model. Two tasks which do not require having the data labeled were considered: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). In the former, the model must learn to recover masked words (15% of the total sentence) in a given sentence. To recover such words, the model must rely on the context, in the form of other words which have not been masked. In the latter tasks, NSP, the model is given two sentences

[1] https://huggingface.co/bert-base-cased

which are concatenated together. The model then has to predict whether those sentences were following each other in the original text or not.

To improve the quality of our sentence embeddings, we want to fine-tune this language model on our intent datasets, which are custom domains. Each sample of an intent detection dataset is composed of one sentence (the user query), and one label (the associated intent). When fine-tuning the language model, we must only use sentences, and we must not use labels (intents), as it would be cheating the downstream few-shot classification problem. Additionally, we cannot consider the NSP task, as each sample is only composed of one sentence, and not a piece of text like a paragraph. Hence, we will fine-tune the language model on the MLM task.

We repeat this training process for each dataset, yielding one language model per dataset. For each dataset, the data is first split into training and evaluation sets. Samples are uniformly distributed among the two sets, no matter the label. The learning rate is first set to $5e^{-5}$, and linearly decreases during training. Every epoch, the model is evaluated using the evaluation set. We retain the checkpoint for which the evaluation loss is the lowest. The various intent detection datasets that we use (introduced in Section 2.5) are quite small compared to the large amounts of data that the transformer was initially trained on. As such, this custom domain fine-tuning is quite fast: we are able to run 20 epochs in under an hour for almost all the datasets. ARSC, the sentiment classification dataset, contains many samples, and took a lot more time (about 5 days). All those training have been performed on a single Nvidia GPU, either GTX 1080 Ti's or Titan RTXs[2].

3.2.2 Evaluating embedding quality

In order to decide which embedding model to use for the different few-shot classification models, we first have to evaluate them. To do so, for each embedding model we consider, we measured the silhouette score[3] of the datasets' provided their ground truth labels. The silhouette score is a clustering evaluation method representing how correct is each point's assignment. For a given data point i assigned to cluster C_i, the method first computes a cohesion score a_i (Equation 3.1), and a separation score b_i (Equation 3.2), and combines them in to a silhouette score s_i

[2]https://www.nvidia.com/fr-fr/deep-learning-ai/products/titan-rtx/
[3]https://en.wikipedia.org/wiki/Silhouette_(clustering)

(Equation 3.3)

$$a_i = \frac{1}{|C_i| - 1} \sum_{j \in C_i, i \neq j} d(i, j) \tag{3.1}$$

$$b_i = \min_{k \neq i} \frac{1}{|C_k|} \sum_{j \in C_k} d(i, j) \tag{3.2}$$

$$s_i = \begin{cases} \dfrac{b_i - a_i}{\max\{a_i, b_i\}} & \text{if } |C_i| > 1 \\ 0 & \text{otherwise} \end{cases} \tag{3.3}$$

In order to compare the transformers' ability to separate classes using no supervision, we compare their embeddings with embeddings derived from FastText[4]. Such embeddings rely on exploiting sub-word knowledge, and the embedding of a word is obtained by averaging the embeddings of its sub-words. Then, words embeddings are averaged into sentence embeddings. We use the available pre-trained model for English, which is publicly available[5]. In Figure 3.1, we display the 2-dimensional representations of both Clinc and Snips datasets, obtained by performing a T-SNE [91] transformation on embeddings derived from three models: FastText, `bert-base-cased` and our fine-tuned model, `bert-fine-tuned`. Interestingly, when comparing the two pre-trained off-the-shelf models, FastText appears to be going a better job at embedding our intent detection datasets than `bert-base-cased`. This visual hint is confirmed by the silhouette score, being better using FastText, especially on the Snips dataset. When using our `bert-fine-tuned` model, this gap is more than compensated, as the silhouette score is higher than the two other models. Visually, it is particularly impressive on the snips dataset, where labels become very separated.

To visualize the impact of fine-tuning language models on our custom domains, we asks such models to perform the masked language modeling task on a few examples, displayed in Table 3.1. In this table, some sentences are written with a masked token. Then, the different models, which have been fine-tuned on the corresponding domains, were applied to predict the missing word under the mask, so we can

[4]https://fasttext.cc/

[5]https://fasttext.cc/docs/en/crawl-vectors.html

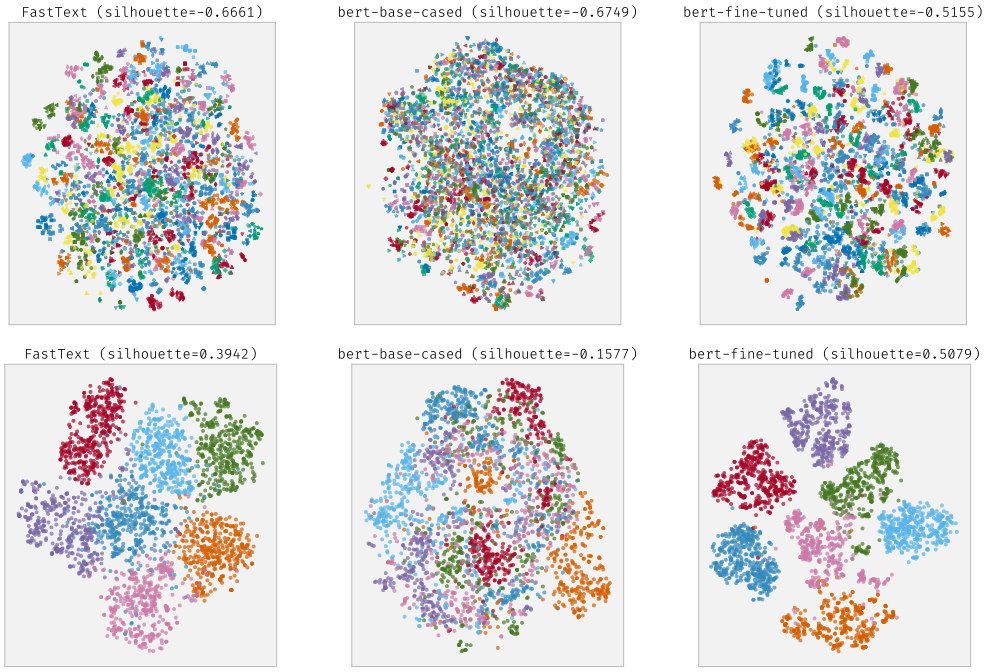


Figure 3.1: T-SNE visualisations of Clinc (top) and SNIPS (bottom) datasets. Embedding methods as well as silhouettes scores are included on top of each sub-figure. From left to right, embedding methods are FastText, bert-base-cased, and bert-fine-tuned.

bert-base-cased		+BANKING77		+Clinc		+HWU64	
(1)*"I want you to skip this **[MASK]**."*							
0.085	one	0.837	transaction	0.498	next	0.613	song
0.084	part	0.054	transfer	0.260	song	0.145	event
0.053	meeting	0.030	payment	0.059	slow	0.096	one
0.032	time	0.009	meal	0.040	play	0.019	music
0.027	thing	0.007	morning	0.027	current	0.014	alarm
(2)*"Can you increase the **[MASK]** please?"*							
0.333	volume	0.131	currency	0.998	volume	0.431	volume
0.136	speed	0.099	charge	0.001	speed	0.344	lights
0.023	power	0.091	exchange	0.000	sound	0.164	light
0.020	pressure	0.089	fee	0.000	oven	0.029	brightness
0.016	number	0.086	rate	0.000	lights	0.023	lighting
(3)*"List my **[MASK]** details"*							
0.297	contact	0.514	identity	0.821	insurance	0.226	contact
0.120	personal	0.258	personal	0.059	income	0.211	party
0.027	family	0.055	identification	0.019	location	0.147	job
0.026	career	0.030	address	0.011	health	0.112	calendar
0.017	biographical	0.026	account	0.010	bank	0.022	meeting

Table 3.1: Examples of masked words filled by our fine-tuned language models. Each column corresponds to a language model. The first one is the `bert-base-cased` pre-trained model. The 3 others corresponds to language models which have been fined tune on the 3 respective datasets. For each model, we output the 5 most likely tokens, along with their probability.

see the impact of the custom domains on the predictions. In all the three examples, from "transaction" to "currency", the BANKING77 models often fits banking-related words in the mask spot. For the other two datasets, audio and music-related tokens like "song" and "volume" often come up. This lexical field is very present in those two datasets, as they contain labels such as "volume_up", "volume_down" or "play_music". Additionally, we also see the term "insurance" chosen by the Clinc model in the third example. This is directly due to the presence of both "insurance" and "insurance_change" labels in this dataset. While fine-tuning the language model on each dataset was very fast (about 1 hour per dataset on a single GPU, on average), those examples illustrate how well language models have adapted to those custom domains.

Now that we have introduced those fine-tuned language models, we will now focus on the few-shot learning methods which we will plug on top of those models.

The central part of our experiments will be about comparing these methods when they are equipped with the same transformer-based text encoder. These few-shot learning methods have been detailed in Section 3.3. In the next section, we will introduce a baseline and then very briefly remind the end-to-end few-shot learning methods we will experiment on in this chapter.

3.3 Few-Shot Methods

In our experiments, we will compare a wide variety of few-shot learning methods, on the text and intent classification tasks. As discussed earlier, meta-learning can be seen as *learning to learn*. Given a set of training tasks, we want to learn a model which is able to look at data in a certain way so that it will be able to perform well on a set of testing tasks, which will be different. In this whole process, what we are trying to fine-tune is the text encoder. While some methods (Relation and Induction networks) introduce additional learnable parameters, some others (Matching and Prototypical networks) only rely on the text encoder's embedding capacity.

In this section, we introduce two classifier baselines to solve the few-shot intent detection task. Additionally, will briefly remind the different few-shot learning methods which we will experiment with. For more details on each method, please refer to Section 2.4.2 of Chapter 2.

3.3.1 Classifier Methods as baselines

Few-shot learning methods were introduced because most existing solutions for text classification were too data hungry, and did not work when the amount of labeled data was too limited. However, with the developments of the attention mechanism and their generalisation with Transformers [137, 33], most tasks have seen a lot of changes in their state-of-the-art methods. This huge change in the way we look at text, and the great improvements shown by having a lot of pre-training makes us ask ourselves how does a regular classifier perform when it is added on top of a state-of-the-art transformer-based encoder. We will quickly describe a regular classifier (called `Baseline`) and a variant of it (`Baseline++`) with which we will experiment

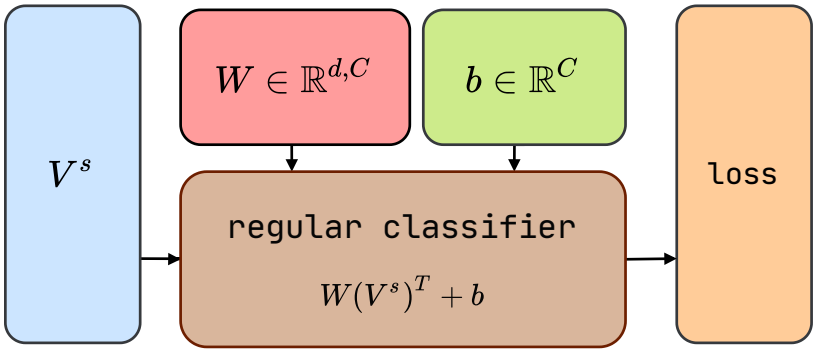


Figure 3.2: Baseline Network

in this chapter.

$$s_{i,c}^{\text{baseline}} = (Wv_i + b)_c \quad ; \quad W \in \mathbb{R}^{C,d} \tag{3.4}$$

The `Baseline` model is a simple traditional classifier used on top of a transformer encoder. The score given to a point x_i with representation v_i using this model is computed by Equation 3.4. Note that this model corresponds to a classifier depending on learnable weights W and b. Also, note that these parameters depend on C, the number of classes in a given episode. Hence, elements W and b depend on the given classes sampled in an episode.

To overcome this issue, W and b are initialised and trained *at each episode* for a few iterations. This is the case for both training and testing tasks. At each episode, W and b are initialised, then fine-tuned on the *support* set of the episode for a few iterations. Finally, the model is evaluated on the query points of the episode. There is no fine-tuning on the query samples of the episode, as it would be cheating: using the labels of query samples unfair, only the labels of the support set are used.

At test time, even though W and b are fine-tuned for each test task, the encoder's weights are frozen, and only the classifier part is fine-tuned. This ensures that the encoder model does not over-fit on the testing domain and completely forgets the knowledge from training tasks.

The variant of this `Baseline` is what we will denote the `Baseline++` [26] model. This variant is about distance-based classification [98], which has been successfully applied [50, 111] to few-shot learning case.

The training procedure of this variant is the same as the vanilla one: learnable

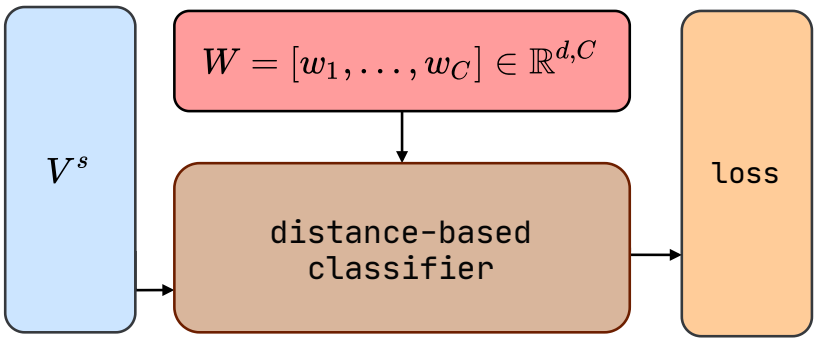


Figure 3.3: Baseline++ Network

parameters are initialised and fine-tuned at each episode. However, the architecture of the classifier is different. Note that the learnable matrix W of the `Baseline` can be seen as $W = [w_1, \ldots, w_C]$ with each $w_i \in \mathbb{R}^d$. This means that each vector w_i can be seen as a representative of class i. To make the prediction for x_i with embedding v_i, we will compute all its cosine similarities $[s_{i,1}, \ldots, s_{i,c}]$ to the different $w_i, i = 1, \ldots, C$, as describe in Equation 3.5

$$s_{i,c} = \frac{v_i \cdot w_c}{\|v_i\| \, \|w_c\|} \tag{3.5}$$

Applying a softmax function on top of cosine similarities $[s_{i,1}, \ldots, s_{i,c}]$ yields a probability vector for sample x_i: this is what we consider to be the prediction for this sample. This classifier makes a prediction based on the cosine distance the input's representation and the different class representatives. Training a model with a distance-based classifier tends to reduce the intra-class variance, as it teaches the model to bring closer representatives of a given class and their class representations, which are learnable. The different class representatives $[w_1, \ldots, w_C]$ can be seen as prototypes from [129], except this time they are learnable, and not computed. Choosing the right distance metric can have a huge impact: when switching from cosine to euclidean distance, authors of Prototypical Networks [129] found they yielded very different performances. Since the Baseline++ approach being distance-based, we will therefore experiment with the two distance metrics (euclidean and cosine).

3.3.2 Few-Shot Learning approaches

In this section, we briefly recap each of the few-shot learning method we consider. We will describe them in chronological order, which is also the complexity order (in terms of number of parameters). While this section acts as a quick reminder, please note that the details about the origin and operation of each of these approach is detailed in preliminaries Chapter, at Section 2.4.1 (p. 25).

Matching Network [139] This model computes the distances between support and query points. The prediction is class with the lowest average cosine distance.

Prototypical Networks [129] For each class, a prototype is constructed as the average embedding of all points of this class. Then, the predicted class is the one closest prototype, in terms of euclidean distance.

Relation Network [133] This method uses the same prototype mechanism introduced before. Then, query points are compared to prototypes using a learnable Relation Module. We consider two types of relation modules: a shallow neural network one, and the Neural Tensor Layer [130].

Induction Network [49] Induction networks follow the same idea of relation networks, as they use the same relation module to compare prototypes to query points. However, they change the way prototypes are computed: they are now obtained through a dynamic routing algorithm, a capsule network mechanism introduced by an older research study [121].

3.4 Experimental Setup

3.4.1 Few-Shot Evaluation Setup

Introduced by [139], few-shot classification corresponds to the case when a classifier must adapt to new classes, denoted here as $\mathcal{C}_{test}$, unseen during training, and only given a few labeled examples of these new classes. To this end, the approaches

assume that during training, a task-significant set of classes noted $\mathcal{C}_{train}$ is available, along with an accordingly task-significant number of labeled data for each class $c_{train_i} \in \mathcal{C}_{train}$. For each training episode, C classes are sampled from $\mathcal{C}_{train}, C \ll |\mathcal{C}_{train}|$. Then, K support examples and Q query examples are randomly drawn for each of these classes. The model is then iteratively trained using both query and support points.

At testing time, the same sampling strategy is made, this time drawing classes among $\mathcal{C}_{test}$, with $\mathcal{C}_{test} \cap \mathcal{C}_{train} = \emptyset$. The model is then evaluated on its ability to predict labels for the Q query samples, using the K support samples (unless otherwise stated, C, Q, and K values are the same at both testing and training time).

This training procedure is called C-way K-shot classification. Following previous works on the subject [129, 49], we use $K = Q = 5$ in our experiments. On the Lara dataset, we run additional experiments using $K = 5$, to compare models when they are given only one sample per class. Concerning the value of C, it is fixed to 2 for ARSC, as this dataset is already composed of binary classification tasks. Regarding the intent detection datasets we introduce later (Section 5.4.1), in order to see the shift between ARSC binary tasks and the more common 5-way evaluation [49, 117], we measured performances of the different models with C ranging from 2 to 5.

Datasets We briefly expose here the different datasets which will be used in the experiments of this chapter. For more information about the characteristics of each dataset, refer to Section 2.5. We first conduct experiments on the ARSC dataset (Table 3.2). For this particular dataset, there are twelve evaluation tasks, and each of these tasks comes with a number of support test samples ($K = Q = 5$ as stated previously). Nonetheless, in [158] the same 5 samples per testing class are fixed for all experiments[6], which leads to a significant selection bias towards these 5 randomly selected samples used throughout the evaluation. In order to get more consistent results, we ran additional experimental runs, each of them selecting randomly new support samples. In Table 3.2, exposing the results on ARSC, this corresponds to the last column (`BERT + Sample shots`). We also conduct experiments on three public intent detection datasets: Liu, Clinc, and TREC28. We conduct a final round of experiment on the Lara dataset.

[6]See labeled sampled in `https://github.com/Gorov/DiverseFewShot_Amazon`

3.5 Observations

We report results for the ARSC dataset in Table 3.2, and results for the Intent Detection tasks in Table 3.4. We also ran experiments on the Lara dataset, and exposed results in Table 3.5. We hereby provide our analysis of these results.

3.5.1 Baselines are surprisingly strong

Few-shot learning methods were originally used to overcome data scarcity. In those situations, training a classifier on top of a small dataset – in our case, 5 samples per class – can be hard. However, our experiments on ARSC show that the Baseline and Baseline++, plain and simple classifiers, get surprisingly close to state-of-the-art results. Table 3.3 provides four correct and four incorrect classification examples for the Baseline model. On the Lara dataset, when using 5 shots per class, the Baseline++ also yields impressive results, with a 97.7% binary classification accuracy (compared to 97.9% for Proto++).

While it fails to predict the correct text label for some shots, it is also able to correctly classify sentences such as *What do I take home ?* among the 50 test classes of the Clinc dataset. On the ARSC dataset, it is also important to note that the Baseline++ model is significantly better than the Baseline, and is even on par with all other architectures, except Prototypical Networks. These great results confirm the recent studies about GPT-3 [19], showing that very large language models succeeds in learning from few shots. We attribute these capacities in our few shot intent detection task to the higher separability that a fine-tuned language model provides (Figure 3.1): the task becomes much simpler, hence boosting such baselines. Finally, on the case of classifier baselines: note that for each testing episode, baselines are first fine-tuned on the (very few) **support** samples of each episode. While most models get a sense of the task they are trying to solve by iterating over training episodes, they do not quite learn about the domain in which they are operating. By adding this small fine-tuning step, baselines are capable of learning something on testing domains, which gives them an advantage over their competitors.

Model	Configuration		Mean binary accuracy		
	Metric	Relation module	Original encoder †	BERT as encoder (↗ or ↘ w.r.t. original encoder)	BERT + Sample shots
Matching Network [139]	euclid.	N/A	–	81.2	82.9
	cosine	N/A	65.7	81.9 (↗)	83.3
Prototypical Network [129]	euclid.	N/A	68.2	80.0 (↗)	82.6
	cosine	N/A	–	81.7	83.5
Proto++ [117]	euclid.	N/A	📷	82.4	**84.0**
	cosine	N/A	📷	**82.6**	83.6
Relation Network [133]	N/A	base	–	81.0	82.9
	N/A	ntl	83.1	81.7 (↘)	83.3
Induction Network [49]	N/A	ntl	85.6	79.3 (↘)	80.3
Baseline	N/A	N/A	📷	80.7	79.8
Baseline++	euclid.	N/A	–	81.9	82.2
	cosine	N/A	📷	79.7	81.1

Table 3.2: Mean accuracy on the 12 ARSC binary classification test tasks. In column †, results are reproduced from the Induction Networks seminal paper [49] (where applies), a dash (–) means that results for that encoder/metric pair were not reported, and 📷 denotes models only tested on computer vision tasks (first time applied to text in our contribution). The *BERT* column is our implementation using the same 5 shots as the first column but using a BERT encoder for all methods. The last column is also using BERT, but results are averaged over five runs, sampling different shots for each run. In the Configuration column, N/A means that the configuration criteria does not apply to the model.

Correct classification examples	
S:	Do I have enough in my boa account for a new pair of skis ?
P:	`balance`
T:	`balance`
S:	What's 15% of 68 ?
P:	`calculator`
T:	`calculator`
S:	I need to know the nearest bank's location.
P:	`directions`
T:	`directions`
S:	What do I take home ?
P:	`income`
T:	`income`
Incorrect classification examples	
S:	On Tuesday you are supposed to have a meeting.
P:	`meeting_schedule`
T:	`calendar`
S:	What are my insurance rewards ?
P:	`insurance`
T:	`redeem_rewards`
S:	How much farther is Orlando from my location?
P:	`current_location`
T:	`distance`
S:	Stop talking please.
P:	`change_speed`
T:	`cancel`

Table 3.3: Examples of Clinc query examples correctly and incorrectly predicted by the Baseline method using 5 shots. S (resp. P, T) is the sentence (resp. prediction and true label).

3.5.2 Sample selection bias

The mean accuracy difference between the last and the second columns of Table 3.2 accounts for the difference of randomly selecting new support samples at each iteration (last column) as opposed to picking the same fixed pool of support samples as done previously (second to last column). We can see that this difference alone

is in the range of the increments brought by each model over time (baselines aside, bringing from 1 point up to 2.6 points for Prototypical Networks). This significant gap shows the importance of using evaluation tricks like cross-validation, instead of evaluating only for one run over a fixed set of shots.

3.5.3 Impact of switching to transformers

One of the main contributions of this experiment is to compare few-shot learning methods with the lowest bias possible. In our experiments, we reduce the bias by equipping all methods with the same fine-tuned language model (see Section 3.2 for more details). On the ARSC dataset, using transformers drastically changes the performances of all methods. When feeding the same transformer-based encoder to all few-shot methods, Prototypical Networks are now on top, whereas metric learning approaches (Induction & Relation Networks) tend to struggle, almost reaching the same performances as Matching Networks.

Such metric learning approaches rely on various weight matrices and parameters, while more traditional approaches (Matching and Prototypical Networks) do not use any additional parameter apart from the encoding step. This hints that the upstream transformer does most of the learning and is able to model the embedding space well enough such that no more additional metric learning is needed. Moreover, the complexity brought by adding more parameters to metric based meta-learning approaches probably makes them harder to train especially in a few shot setting. The massive increase in embedding quality brought by the BERT encoder makes Prototypical Network approaches reclaim the state-of-the-art position.

3.5.4 The curious case of induction networks

When Induction Networks [49] were introduced, both the ARSC dataset and a private intent detection dataset were used for evaluation (publicly unavailable). Evaluating such model on this sentiment classification dataset as well as a private intent detection dataset made it hard for us to acknowledge their results as is, so we intended to evaluate this model on public datasets. As authors neither did release their official code nor did respond my contact attempts, I re-implemented their code myself. Experiments of this method using my code on the ARSC dataset confirm those

	Metric	Relation Module	Liu				Clinc				TREC28			
			2	3	4	5	2	3	4	5	2	3	4	5
Matching	euclid.	-	96.6	93.7	91.1	89.1	99.2	98.7	98.1	97.7	89.4	81.6	76.6	69.6
	cosine	-	93.3	87.9	84.8	81.0	96.8	95.8	95.1	94.7	81.6	75.4	68.5	63.5
Proto	euclid.	-	97.4	95.3	93.4	91.8	**99.5**	99.0	98.7	98.4	**92.6**	**87.6**	82.0	**79.2**
	cosine	-	94.6	90.4	88.5	85.6	97.6	97.3	96.9	96.5	85.6	79.1	74.5	71.3
Proto++	euclid.	-	**97.7**	**95.7**	**93.7**	**92.2**	99.5	**99.1**	**98.8**	**98.5**	91.7	84.9	**82.0**	76.8
	cosine	-	94.0	90.9	87.9	85.4	97.5	97.3	97.0	96.5	83.8	78.1	71.0	65.9
Relation*	-	base	88.2	76.5	71.8	65.1	91.1	86.0	79.9	77.9	80.8	66.3	61.7	51.8
	-	ntl	92.1	85.1	81.6	76.5	92.9	92.1	90.0	89.1	80.1	72.9	64.1	59.4
Induction*	-	ntl	88.4	81.3	74.3	70.1	92.3	88.7	85.7	80.4	78.3	65.7	57.3	51.3
Baseline	-	-	94.3	89.0	84.1	79.8	99.1	98.5	97.7	97.2	90.5	83.6	79.3	75.7
Baseline++	euclid.	-	93.1	87.6	81.4	78.1	95.8	93.3	92.1	90.6	87.7	78.3	72.5	69.1
	cosine	-	93.1	86.8	81.0	75.1	98.9	97.9	96.8	96.1	86.7	78.2	72.1	70.0

Table 3.4: Mean accuracy of C-way 5-shot intent detection, with C ranging between 2 and 5. For each column, the best method is highlighted in **bold**. Each reported value is the average over five runs with different random seeds. **Remark*:** some of those results are not the same as in the paper [37] associated to this chapter. This is after identifying a bug in the code. While the corrected experiments gives better results for the victims of this bug – relation and induction networks –, all previous conclusions still stands (the improvement of the corrected implementation is not enough).

	Metric	Relation Module	Lara – 1 shot				Lara – 5 shot			
			2	3	4	5	2	3	4	5
Matching	euclid.	-	93.4	88.5	85.7	83.5	96.4	95.0	92.7	91.7
	cosine	-	92.0	87.8	85.2	82.0	95.9	92.5	90.4	88.5
Proto	euclid.	-	93.1	89.8	86.3	84.0	97.6	96.5	95.2	94.1
	cosine	-	93.3	88.3	84.7	81.7	97.0	94.8	93.4	91.7
Proto++	euclid.	-	**95.9**	**93.3**	**90.4**	**87.8**	**97.9**	**96.6**	**95.5**	**94.6**
	cosine	-	95.4	91.3	88.6	86.5	97.3	95.1	93.5	92.0
Relation	-	base	79.6	67.8	61.9	58.1	85.2	77.1	68.2	63.7
	-	ntl	87.7	81.5	77.6	73.2	92.2	88.4	84.2	81.6
Induction	-	ntl	81.5	73.7	64.9	64.2	86.6	77.1	73.2	66.1
Baseline	-	-	90.3	83.9	77.9	73.6	95.7	92.4	89.4	86.6
Baseline++	euclid.	-	91.3	86.1	81.3	78.1	97.7	95.8	94.2	92.4
	cosine	-	89.4	82.4	76.7	72.2	94.9	90.8	87.0	83.7

Table 3.5: Mean accuracy of C-way K-shot intent detection on the Lara dataset, with C ranging between 2 and 5, and K between 1 or 5. For each column, the best method is highlighted in **bold**. Each reported value is the average over five runs with different random seeds.

results in an acceptable range, even when trying to get more consistent results using multiple random seeds. Nonetheless, the performances of this method are underperforming on all three intent detection datasets, even when matching the binary classification scenario using $C = 2$. Such a big performance gap between sentiment and intent classification tasks show that Induction Networks, while suited for the former, are not applicable to any other classification task out of the box.

3.5.5 On metric choice

Prototypical Networks were originally designed to do better than Matching Networks. The two differences between them are the placement of the `class average` step, and the choice of the metric (cosine for Matching, euclidean for Prototypical). Our results show that metric choice yields a big gap in performances for both methods, this gap being larger than the gap caused by the model design. This hints that when using a pre-defined metric – excluding the case of metric learning –, choosing the right metric is of paramount importance. Moreover, while Matching Networks

were designed to use the cosine distance, we found here that they perform significantly better (on all datasets for all number of given test classes) when equipped with the Euclidean distance, the metric of choice in the Prototypical Networks seminal paper. Retrospectively, this probably had been tested by the authors of Prototypical Networks in order to select the best metric for the architecture, though this was not public knowledge. The choice of the metric is therefore the second most important choice when designing a meta-learning framework, after the text encoder (Section 3.5.3).

3.5.6 On architectural choices

Overall, Prototypical Networks come on top of every intent detection dataset. More importantly, their gap between other competing approaches is wider as the number of classes increases. This result is important, as in practice, the number of classes is likely to be higher than what is used in the literature – we remind here that our private intent detection module at Meetic encompass 96 different classes at the time of writing. The extended variant, Proto++, gets excellent results on the Lara dataset (Table 3.5), especially in the 1-shot scenario, where it scores about than 3 points of accuracy higher than the second-best method (vanilla Prototypical Network). This performance is not observed on the other intent detection datasets (Table 3.4), Proto++ gets more mixed results. While this shows that using unlabeled data can have some benefits, we also observe that the Proto++ way of integrating this external knowledge is perfectible. Ultimately, note that our results do not mirror Computer Vision results. Since few-shot learning methods are used on top of embeddings, we could emit the hypothesis that they can be applied to any embeddings, regardless of the field. However, while Relation Networks, for example, were performing well in Computer Vision classification tasks – the tasks which they were originally designed for – as well as text classification back in the days when transformers did not exist –, this is not the case any longer. The drawback is that all methods are very sensitive to the feature extractor used in prior steps.

3.6 Conclusion

In this chapter, we first identified the need to have a fair and rigorous benchmark to compare all few-shot learning methods. The text encoding methods used so far were based on fixed word embeddings. Transformers have drastically changed the way we work with text, and few-shot learning methods were yet to be tested using such language models. First, we showed how we could fine-tune such language models for short user utterances in an unsupervised way, prior to solving the supervised task. This additional step is cheap, as it does not require any labelling effort. Additionally, in the conversational agent framework, collecting unlabeled data is easy, as we just need to wait for real-life users to interact with the agent. Through an analysis of the embedding space, we showed that fine-tuning the language model already helps at separating classes, which means a better starting point for the downstream task of text classification.

We then established a fair comparison of the various end-to-end neural few-shot text classification methods discovered over the last few years. When they are all equipped with a transformer-based text encoder, we showed that Prototypical Networks, which still stands as one of the most simple and straight-forward approach, become the state-of-the-art again. We also found that a traditional classifiers trained on few shots yields very competitive results. In terms of parameters, prototypical networks and baselines are not very heavy. Their good performances show that most of the learning is done by language models, which are able to represent text into vectors in a way that was not possible before. We also confirmed the impact of the chosen metric. This difference in performances, which was already observed by the authors of Prototypical Networks, is illustrated in our experiments with all methods which require a pre-defined distance metric. Overall, we have found that euclidean distance works better than the cosine one.

With this contribution, we not only revisit the state-of-the-art, but we also build a complete setup in order to implement, test and compare further contributions with respect to existing works, which will be be of the utmost practical interests in this thesis and other researchers in the community. Reimplementing all the existing methods and prepare this meta-learning evaluation framework took a significant part of

my first years of PhD. The complete source code (re-implementation and evaluation framework) is publicly available[7]. To the best of my knowledge, this framework is already used by Raphäel Chevasson, a second year PhD student at the laboratory working on non auto-regressive method for text generation, and Dina El Zein, a Master student from ENS Lyon in our team who worked on mitigating gender bias for meta-learning frameworks. I hope this public contribution will help the community to build upon consistent comparative experiments, and foster end-to-end few-shot text classification.

[7] https://github.com/tdopierre/FewShotText

Chapter 4

Few-shot Pseudo-Labeling for Intent Detection

This chapter is based on the following publication [38]

> Thomas Dopierre, Christophe Gravier, Julien Subercaze, Wilfried Logerais. "Few-shot Pseudo-Labeling for Intent Detection". In *Proceedings of the 28th International Conference on Computational Linguistics (COLING2020)*, pages 4993–5003, 2020.

4.1 Introduction

Labeling user utterances as intents is of paramount importance for chatbots. Intent classes are usually engineered by a linguistic team, and these classes are likely to evolve with time. The most frequent situations include: the addition of new unanticipated intents found by dialogue traces analysis, the need to introduce new intents that were previously handled using web forms, or the urge to split existing intents into several new ones to support more fine-grained user interactions. Regardless the rationale, each time the intent referential evolves, a significant amount of new labeled data is required to train a decent intent detection model. This calls for a time-consuming, error-prone, and overall expensive labeling process. Consequently, intent detection systems need robust models in few-shot settings in order

to avoid to repeatedly suffer from labeling user utterances into intents. In the previous Chapter 3, we evaluated the different end-to-end few-shot intent classification systems. Through extended experiments using recent text encoders, we showed that simple approaches equipped with a transformer model yield the best results – "simple is beautiful" for few-shots pseudo-labeling. In this Chapter we build upon this observation and we provide an original and state-of-the-art approach based on pseudo-labeling for few shot intent detection. Besides that first observation, a second preliminary observation is as follows: while labeled data are costly to obtain, unlabeled data are not. Indeed, when a conversational agent interacts with real world users on a daily basis, a lot of raw data can be collected. While those raw queries do not always match known labels, they contain some knowledge that can be used. Answering the few-shot classification problem can be done using those unlabeled data, by generating pseudo-labels for unlabeled utterances. A pseudo-label is a label (intent class) automatically assigned to an unlabeled utterance using the knowledge found in the set of labeled data. In the literature, it is also sometimes called weak label, as such data are less confident than human-annotated sentences. The resulting dataset composed of both labeled and pseudo-labeled utterances is then used to train an intent classification model[1]. Hopefully, using both labeled and pseudo-labeled data yields better results than just using the labeled set. Most pseudo-labeling methods combine advances in word representations [110, 17, 136] and then consider unsupervised clustering algorithms [141] in order to propagate known labels to unlabeled data. Nonetheless, there are major drawbacks to existing pseudo-labeling algorithms. First, since clustering algorithms like k-means are used to partition user utterances representations, it provides a pseudo-label to each unlabeled data point. When collecting unlabeled data, one cannot entirely be sure that all those utterances match an intent for which we have labeled data. For example, users might be asking out-of-scope questions, like "What's the weather like?" to a dating conversational agent, a question which he might not be trained to answer. In such scenario, it would be better to simply discard this data point instead of assigning it a pseudo-label of a known class, which would inherently not be correct. Having wrong pseudo-labels makes it is much harder to learn a discriminate hyperplane when training the downstream intent detection models since some data

[1] Pseudo-labeling is described in chapter 2, more precisely in section 2.4.3.1 (p. 35)

are incorrectly labeled. Worse still, this grows exponentially as the number of intents (classes) increases in practice as predicting correct pseudo-labels then becomes even more difficult. A corollary is also that confidence values in pseudo-labels is hard to produce: since cluster shapes are unknown in advance, distances to centroids cannot reliably be used as a factor a confidence in a pseudo-label. In Computer Vision, the same issues exist for the image classification task and this lead to end-to-end systems like Prototypical networks [129] that we already exploited in chapter 3 when we applied them to intent detection – such end-to-end systems remain baselines in the contribution described in this chapter.

In this chapter, we present different methods for pseudo-labeling utterances and leverage their results for the downstream task of intent detection. We first assess the correlation between good pseudo-labeling and the intent detection downstream task, as well as assessing the robustness of various methods in one/few-shot configurations.

Using the advances in word representations [17, 136, 33], we consider unsupervised algorithms to embed user utterances in order to propagate labels. Partitioning user intents using hierarchical clustering has already been shown to be effective for intent discovery [127]. Note that in intent discovery, we are looking to recover the latent intent from user utterances. In our experiments, intent classes are known in advance – domain-specific chatbots usually rely on linguists team to provides such classes.

We introduce a new folding/unfolding hierarchical clustering algorithm for pseudo-labeling, which can be considered as a dynamic hierarchical clustering algorithm (Section 4.4). This original two-step algorithm is able, by design, to predict a pseudo-label by organizing data points in a hierarchical manner. Interestingly, because some pseudo-labels may be incorrect, we introduce a normalized confidence score based on a temperature weighting mechanism, to penalize pseudo-labels which might be incorrect. Indeed, if there are too many incorrect pseudo-labels, we might be left with a model which performs worse than the model only using the labeled data, as we would add too much noise. That is the reason why we weight the pseudo-labels depending on a confidence score. When training the downstream intent detection model using both labeled and pseudo-labeled utterances, those weights are

included in the cost function we want to minimize (Section 4.4.3). In order to validate our pseudo-label approach, we evaluate its impact on the downstream task of intent detection using four datasets and fine-tuned BERT-based language models (Section 4.5.1). Overall, in few-shot configurations and even when the number of classes is large, our proposed method is able to positively impact the intent or text classification tasks accuracies (Section 4.5.2), beyond the state-of-the-art. Additionally, our method is able to discard some unlabeled data points if its confidence about such data points is too low. In a context where unlabeled data might concern intent that are not supported (users asking out-of-scope queries), having this discarding feature is great improvement to avoid adding to much noise to the downstream classifier.

4.2 Philosophy and approach with respect to existing solutions

A growing amount of work has been done on the Conversational AI domain [46] over the recent years. There are different types of conversational agents: Question-Answering systems [116], End-to-end Conversational Models [140, 116], or Task-Oriented bots [92]. In the latter case, a central part of the process is the understanding of user utterances, a special case of text classification. Convincing approaches to this problem rely on neural networks [80], thus requiring large labeled datasets, which are costly to obtain. In practice, as more users interact with the conversational agent, it is easy to collect a large amount of raw data. Having access to a small labeled dataset along with a large unlabeled one calls for semi-supervised learning solutions [22, 171] for which label propagation is of paramount importance. Many solutions exist for this and we provided an overview of the main family of approaches in chapter 2, Section 2.4.3.1. Among them, one approach to this problem is self-training [157, 99], where the model trains itself for multiple iterations, each time adding its most confident prediction to the training data. This approach has multiple weaknesses, the main one being the fact that the model cannot correct its own mistakes. One way of dealing with such a problem is to use tri-training [167], a framework where three models are trained iteratively. At each step, common predictions of two models are added to the training set of a third. This continues until

all models reach a stationary state. A substantial body of works focus on the few-shot classification problem, that is handling labeled data scarcity in the downstream training phase exclusively [45, 129].
From a broader perspective, it is worth noticing that semi-supervised learning (learning using both labeled and unlabeled data) has also received a lot of attention in computer vision [139], where annotations are time-consuming and error-prone as well. To overcome this problem, image-specific features – like coloration, pixel-level values – are used [84]. Given a distance matrix between data points, labels from known samples are propagated to previously unlabeled data. Unfortunately this is hardly applicable in NLP as complex features are hardly intrinsic to words but rather based on distributional semantics. Indeed, in NLP we usually rely on pre-trained word embeddings [101] and build sentence representations by averaging word vectors [66] or contextualized word embeddings [110, 30].
In this work, we therefore tackle the problem differently: we propagate the knowledge we have on this small amount of data to unlabeled data points, before feeding both truly labeled and pseudo-labeled data to the intent detection system. This intermediate step is of the utmost practical interest: firstly, propagating knowledge on unlabeled points helps us understand how the data is structured. Secondly, in a labeling framework, it allows us to select interesting samples for our in-house linguists to label or correct given low pseudo-label confidence[2] as a human-in-the-loop approach, instead of selecting samples randomly. The more accurate the pseudo-labels, the more efficient it is for linguists.

In this chapter, in addition to introducing a novel way to derive pseudo-labels for unlabeled data points, we also show that our method work in complement with other existing methods, the aggregation of them all providing a significantly even better pseudo-labeling method for intent detection.

4.3 Baselines

In this section, we will name the various pseudo-labeling methods which we will use as baselines in the experiments of this chapter. While some of them were not particularly designed to assess the pseudo-labeling cased, any classifier can be used to

[2]The pseudo-label confidence value is the alpha value calculated in section 4.4.3.

predict a pseudo-label for a given unlabeled data point. Pseudo-labeling techniques can be directly plugged on top of sentence representations. These sentence representations can be obtained in many different ways (see Section 2.2 in Chapter 2). For consistency purpose in the various experiments showcased in this thesis, we are using the same BERT models here than the ones used in Chapter 3. As a reminder, this language model was an off-the-shelf `bert-base-cased` model, which we fine-tuned using the masked language modeling task on each dataset custom domains. This fine-tuning was done once for each dataset, thereby resulting in one custom transformer per dataset.

From these sentence representations, labels from known utterances can be propagated in different ways. First, we consider Prototypical Networks as a baseline. We already proved their efficiency in Chapter 3, where they claimed the top spot for few-shot text classification. Having the better accuracy in this task, those models should also be very good at predicting correct pseudo-labels using the learnt class prototypes. Additionally, we also considered their variant `proto++` in our experiments, as this variant was often slightly better than the vanilla version of prototypical networks.

Second, another baseline we consider is the following: for each unlabeled data point and each class, we compute its average similarity to the known labeled samples of this class. Then, we assign to this point the label for which this average similarity is maximal, mapping all unlabeled data points to a pseudo-label. Comparing the average similarity to different classes is the essence of Matching Networks [139], which were already described previously (see Section 2.4.2). We will denote this method `matching`. However, we do not fine-tune this method in an end-to-end manner, instead we clamp this prediction function on top of fixed word embeddings. This comes from the fact that our method, which we will introduce later, does not predict labels in a differentiable manner. As such, it cannot be used to fine-tune representations using traditional gradient descent.

4.4 Two-fold Pseudo-Labeling

In this section, we describe our folding/unfolding pseudo-labeling method (Section 4.4.1), as well as the aggregated approach (Section 4.4.2). We also introduce a loss weighting scheme (Section 4.4.3), which is hopefully, able to mitigate the impact of incorrect pseudo-labels on the intent detection downstream task. We advocate that our algorithm is state-of-the-art (Section 4.5.2). Being state-of-the-art and simple, this algorithm is an appealing and elegant solution for pseudo-labeling.

4.4.1 Folding/unfolding algorithm

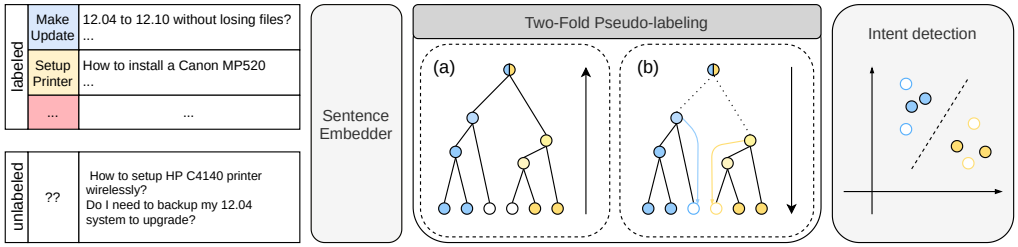


Figure 4.1: The two-fold Pseudo-Labeling process taking as an input embedded user utterances from our fine-tuned transformer. First, a hierarchical clustering method is applied from bottom to top, leading to a tree structure (*(a) – folding*). Second, from top to bottom, nodes are expanded iff they contain multiple labeled sentences with different labels ((b) – *unfolding*). Finally, retrieved pseudo-labels are used to train an intent detection model (Section 4.5.1).

Our pseudo-labeling method, illustrated in Figure 4.1, is a folding/unfolding algorithm, using hierarchical clustering. Its pseudo-code is also detailed in Algorithm 5. We decompose our approach in two steps.

Step (a) – folding We start with all data points $X = X_l \cup X_u$, mixing together both labeled data points X_l as well as unlabeled data points X_u. Those data points are sentence representation, previously obtained using a text encoder. At the beginning of the step, each data point is considered as an individual cluster, containing a single sample. Iteratively, we construct a tree in a hierarchical way: at each iteration, we select the two closest clusters, and bind them together into a new cluster. To choose which clusters to merge, we use Ward's method [143]. This method, also called the

Algorithm 5 Two-Fold Pseudo-labeling

Input:
Labeled data $L \leftarrow \{(x, y) \in L\}$
Unlabeled data $U \leftarrow \{(x, \texttt{None}) \in U\}$
Output:
Pseudo-labeled data $P \leftarrow \{(x, \tilde{y}) \in P\}$

// Step (a): Folding
$V \leftarrow \{\{(x, y)\} \mid (x, y) \in L \cup U\}$ ▷ `Init. clusters as single data points`
while $|V| > 1$ **do**
  $v_1^*, v_2^* = \underset{(v_1 \neq v_2) \in V^2}{\operatorname{argmin}} \operatorname{Ward}(v_1, v_2)$ ▷ `Find best couple of clusters`
  $v \leftarrow v_1^* \cup v_2^*$ ▷ `Merge clusters together`
  $V \leftarrow (V \setminus \{v_1^*, v_2^*\}) \cup \{v\}$

// Step (b): Unfolding
$V' \leftarrow \varnothing$
$P \leftarrow \varnothing$
while $|V| > 0$ **do**
  draw $v \in V$ randomly
  $L_v \leftarrow \{(x, y) \mid (x, y) \in v\,;\, y \neq \texttt{None}\}$
  $U_v \leftarrow \{x \mid (x, \texttt{None}) \in v\}$
  if $|L_v| = 0$ **then**
    Do nothing. ▷ `If no labeled samples, discard the cluster`
  else
    $C_v \leftarrow \operatorname{uniques}(\{y \mid (x, y) \in L_v\})$ ▷ `Unique labels of this cluster`
    if $|C_v| > 2$ **then**
      $v_1, v_2 = \operatorname{unfold}(v)$ ▷ `Unfold cluster`
      $V \leftarrow (V \setminus \{v\}) \cup \{v_1, v_2\}$
    else
      $\tilde{y} = C_v[0]$ ▷ `Derive the unique label of the cluster`
      for $x \in U_v$ **do**
        $P \leftarrow P \cup \{(x, \tilde{y})\}$ ▷ `Apply pseudo-label`

Ward's minimum variance method, minimizes the total within-cluster variance. At each iteration, it selects the two clusters which, if merged, would yield the smallest increment of total within-cluster variance. As each iteration consists in merging two clusters, if we have N data points at the beginning, the overall process will take $N-1$ steps. The output of this folding step is a tree structure, obtained in a bottom-up manner. In this tree, each non-leaf node has exactly two children, representing a merge between two nodes at some iteration. The root of the tree corresponds to one big cluster which contain all data points.

In traditional hierarchical clustering, the goal is to separate the data points into clusters. Once the tree is constructed, one can either cut the tree at a chosen threshold corresponding to the minimum distance between two clusters. The higher the threshold, the less clusters it would yield. In our method, we are not particularly interested on fixing a threshold or a given number of clusters. Indeed, having a constraint on either of those value would mean adding additional hyper-parameters. While hyper-parameter search is doable, it is hardly applicable to the few-shot scenario, where there are not much labeled data to find the parameters which would yield the perfect results. Additionally, if anyone has to use few-shot methods because of the lack of labeled data, the hyper-parameters found in the literature on a few datasets might not work for a particular dataset. As this first step, a standard hierarchical clustering, is finished, we describe in the next section the more original and second step of our algorithm: the unfolding step.

Step (b) – unfolding After doing the bottom-to-top process detailed previously, we now operate in a top-to-bottom manner. The step (a) yielded one tree structure containing all data points, with each node of this tree represents a merge of two clusters. Iteratively, we are going to unfold the whole structure, by splitting clusters given a specific condition. More precisely, each cluster will be split into its two sub-clusters (which were merged at some point in step (a)) until all generated clusters contain either no labeled data points (in this case all its data points, which are unlabeled, are discarded), or some labeled data points with a unique label. In the latter case, we assign this unique label to all the unlabeled data points also belonging to the same cluster. Here is a more detailed overview of the different scenarios when processing a given cluster:

1. **The cluster only contains unlabeled data points**. In this case, as we have no labeled information whatsoever about the cluster, we simply discard it.

2. **The cluster contains only one labeled data point, and some unlabeled data points**. In this case, we assign to the unlabeled data points of the cluster the the same pseudo-label as the labeled data point.

3. **The cluster contains several labeled data points but with a unique label, and some unlabeled data points**. This case is very close to the previous one, except there are multiple labeled data points with the same label. Still, we assign pseudo-labels the same way as in (2).

4. **The cluster contains multiple labeled data points with multiple unique labels**. In this case, it is hard to directly conclude on which label to assign to unlabeled data points, as we have mixed signals. To overcome this issue, we split the cluster into its two sub-clusters in order to have more precise information about the data points.

5. **The cluster contains no unlabeled data points**. In this case, there is not much to do, as we want to assign pseudo-labels to unlabeled data. If there are no unlabeled data in the cluster, we decide to not make anything out of it.

This step is quite intuitive: if a cluster contains labeled data points with different labels, then it is not easy to choose which of those label will be assigned to unlabeled data points in this cluster. On the opposite, splitting clusters until only one label is represented in each cluster improves the confidence when assigning the pseudo-label to previously unlabeled data points. Additionally, we could push it a little further, and only stop when only one labeled sample is left in the cluster (that means, split a cluster event if it has a unique label, but multiple labeled data points). This way, we would be more precise and increase the confidence of pseudo-labels. However, this deeper variant did not yield good results in our experiments, as it led to a discard rate which was too high.

There are several underlying motivations for this fold/unfold approach. First, unlike most techniques, this approach does not require any hyper-parameter tuning, apart from the method used to choose which clusters to merge in step (a). Having little to no hyper-parameters make the method more robust to different datasets. In

practice, when there is not much data to validate methods – this is the case in few-shot learning –, this is very important. Secondly, thanks to its iterative hierarchical process, it is able to model unusual cluster shapes, which makes it more robust to sentence representations. Thirdly, the tree structure obtained via hierarchical clustering is richer than a more traditional cluster output. By iteratively unfolding a tree structure, it helps understanding the clustering process and how data points are linked to each other. Lastly, unlike the `matching` pseudo-labeling method or prototypical networks, this method does not give pseudo-labels for all the unlabeled data points, as we discard clusters which only contain unlabeled data points. As discussed earlier, in a real-world scenario, this is very important, as we cannot ensure that all unlabeled data points – representing raw user utterances – belong to a known class. It is also better to consider not labeling a data point that is very hard to pseudo-label to avoid noise in the training data.

4.4.2 Aggregated pseudo-labeling approach

While pseudo-labeling methods have their own way of assigning pseudo-labels, they can be aggregated to improve the quality of pseudo-labels. As such, we also experimented with an aggregated method using common pseudo-labels obtained from each of our single methods. Indeed, all pseudo-labeling schemes introduced earlier have a unique way to represent data and assign pseudo-labels. Following the unanimity vote for each given data point, we add an aggregated pseudo-labeling approach in our experiments. To do so, after retrieving pseudo-labels from the various methods, we retain data points where all methods have assigned the same pseudo-label, and discard the rest. If all methods agree on a pseudo-label, then there is a higher chance of this pseudo-label being correct, but it's not guaranteed. However, the more methods we aggregate together, the less pseudo-labels we will have left to train our intent classification system. This can be a problem, and there is a trade-off between quality and quantity to assess. In order to measure the impact of each single method in the aggregation step, we will also conduct ablation experiments, where we aggregate common predictions of all methods except one. By doing so for all methods, we will be able to measure the contribution of each method to the aggregation (Section 4.5.3). If a method is orthogonal to the others, than its contribution to the aggregation should be high. Conversely, if a method yields pseudo-labels which are

quite similar to another method, then its contribution to the aggregation should be small.

4.4.3 Loss weighting mechanism

We will later empirically demonstrate that pseudo-labeling methods are able to recover most labels in one/few-shot configurations when the number of classes is relatively small (Section 4.5.2). However, some pseudo-labels can be easier to assign than others, depending on their distance to labeled data points in the embedding space. When the number of labels is very high, generating correct pseudo-labels while refraining to introduce incorrect pseudo-labels at the same time becomes harder. As the number of intent classes grows, the effect of providing correct labels issued by semi supervision is vastly reduced by incorrect pseudo-labels. This is the case even if those incorrect pseudo-labels are provided in a lower amount than correct ones, as it confuses the intent detection model and make the optimization task unnecessarily harder/incorrect. In order to overcome this issue, we introduce a confidence score in our algorithms so that the loss function used to train the intent detection model penalizes mis-predictions on pseudo-labels with respect to the confidence in the generated pseudo-label. More formally, given the similarity matrix S, an unlabeled data point $x_u \in X_u$ with pseudo-label $\hat{y}$, for each class c, we compute its logits $z_{u,c}$ as follows:

$$z_{u,c} = \frac{1}{n_{l,c}} \sum_{i=1}^{n_l} \mathbb{1}(y_i = 1) S_{i,u} \tag{4.1}$$

Then, using a softmax, we turn this z vector into a probability vector $P(z)$ (that is $P(z) = \sigma(z)$, where $\sigma(\cdot)$ is the softmax function). From here, the last steps are inspired from [84]. We apply a final transformation in order to increase the differences between logits. The reason behind this is that all logits, being an average of similarity scores, are bounded between 0 and 1. Hence, if we just apply the softmax on those raw logits, the differences between classes will not be very significant. As a consequence, to increase sparsity and enforce extreme weights, we define the pseudo-label weight α as follows (we set the variable $\tau = 10$ in the experiments as

in [84], and $\hat{y}$ still denotes the pseudo-label):

$$\alpha = \frac{\exp\left(\tau z_{u,\hat{y}}\right)}{\sum_{j \neq \hat{y}} \exp\left(\tau z_{u,j}\right)} \tag{4.2}$$

We will further discuss the impacts of this weighting technique in Section 4.5.4.

4.5 Experiments

In this section, we first introduce the experimental setup for our contribution. This includes datasets, experimental conditions, as well as a brief reminder about the meta-learning framework. Then, we will showcase the results and provide our critical analysis of these results.

4.5.1 Setup

Methods Alongside with our contribution, the systems we place under evaluation are `matching`, `Proto/Proto++`, as introduced in Section 2.4.2. We implement the `matching` method of predicting pseudo-labels, and for the Prototypical-based approaches, we use the code associated to the seminal Prototypical Network paper[3], using our fine-tuned transformer as our sentence encoder. For the `Proto++` baseline, to refine prototypes, we use 20 unlabeled samples per class both at training and testing time, as Ren [117] showed that it yielded better performances. As suggested by Chen [26], we do not fine-tune Prototypical Networks at test time, as it is reported to decrease performances. Moreover, note that Prototypical Networks are impossible to fine-tune using a single shot for each class, which will be the case in some of our experiments (one-shot learning scenarios).

Evaluation The evaluation of the different systems is two-fold. First, we will assess the relevance of pseudo-labels retrieved from the different methods in few-shot situations. While step measures the quality of pseudo-labels, it is not what we are trying to achieve in the end. Indeed, our main goal is to have pseudo-labels positively impacting the performances of the training of an intent classifier. Hence, we

[3] https://github.com/renmengye/few-shot-ssl-public

then evaluate the performances of the intent detection model (see paragraph 4.5.1) trained using those pseudo-labels alongside with the few labeled data. In order to measure the various performances, we use the weighted F1-score in all our experiments.

Few-shot & Cross-validation setup As detailed earlier (see Section 2.4.1), we experiment in the meta-learning framework, where datasets are split into training and testing sets [133], with no overlapping classes between the sets. Since few-shot regimes are highly dependent on the random initialisation of given shots, it is standard to repeat this process many times to create as many training/evaluation tasks and report average accuracy values. More formally, for each training task, we sample C classes from the training classes C_{train}. For each class $c \in C$, we sample K support samples $S_c = \{(x_i, y_i = c)\}_{i=1}^{K}$, as well as Q query samples $Q_c = \{(x_i, y_i = c)\}_{i=1}^{Q}$. Those samples will be used to train the model for this task. This setup is often named C-way K-shot settings. Test tasks are constructed in the same way, using the test set. At test time, we will evaluate the quality of our utterance encoder on those new classes C_{test}, disjoint from classes the model trained on. In our experiments, for each dataset, we use a third of classes for each training, validation, and testing stages. We also vary the number of shots K to assess the robustness of the different systems.

In order to get a sense of consistency in the results, each reported metric is averaged over 10 different cross-validation runs. For each run, the whole training set is used to train the pseudo-labeling model (e.g. Prototypical Networks). Only 90% of the test set is selected to assign pseudo-labels to query samples using support samples. The remaining 10% of the test set is used to evaluate the performance of the text classification system which is trained on top of those pseudo-labels.

Datasets To measure the effectiveness of our method, we explore 5 intent detection datasets. Additionally, because our method is not specific to intent detection, we also consider R8, a text classification dataset in order to have an insight of the accuracy when the input text differ (see datasets statistics reported in 2.5 in Chapter 2 – p. 44).

Sentence Encoder As in the previous Chapter, we use our fine-tuned the `bert-base-cased` model [146] on its former task, masked language modeling. Because this is an unsupervised task, we are able to use both labeled and unlabeled utterances during training. As shown earlier, this additional fine-tuning greatly increases the quality of embeddings. This fine-tuned model is then used as a checkpoint upon which prototypical-based approaches are built. Additionally, because prototypical networks benefit from a training on classification training tasks, it would be unfair to prevent the other approaches – `matching Fold/Unfold` – from benefiting from such a fine-tuning. To account for this difference, we further fine-tune the BERT model obtained earlier on the classification task, using only few shots – `BERT-Fit`. The encoding part of this model is then frozen, and sentence embeddings used as input for both `matching` and `Fold/Unfold` methods. This ensures that all methods benefit from a classification fine-tuning, only using the few shots which they are given.

Intent Detection model Given the recent advances in NLP using transfer learning, we tried several methods for the intent detection model. We used pre-trained ELMo [110], InferSent [30], FastText [17] and BERT [33] models. For ELMo (resp. BERT), we use the last hidden vector of the last (resp. first) token as the sentence embedding. All language models are fine-tuned with a last classification layer. We compare those methods on the intent detection task, and found that BERT was yielding the best results. As in [131], we tried several variants for the BERT model – fine-tuning the model in the Masked Language Modeling (MLM) task as well as freezing or not freezing the transformer part. We find similar results as this paper: the variant which works best is obtained when fine-tuning on the MLM task, then further fine-tuning on the classification task, without freezing any layer. Hence, we chose this version as our intent detection model, in all our experiments.

We show that given a set of sentence representations, our method is the best as propagating the knowledge from labeled to unlabeled data points. We conduct additional experiments, where sentence embeddings are derived from a prototypical network which has been trained on training tasks, and we will plug the different pseudo-labeling methods on top of those embeddings. Results for those experiments are available at Table 4.4

4.5.2 Results and Analysis

We first report samples of assigned pseudo-labels by the various methods in Table 4.1. Sentences containing single digits numbers (sentences numbered 1, 2, 3 in that table) are often mistaken for RateBook, because those kind of tokens appear in a lot of Book Rating utterances. In (3, 4), sentences contain time-related tokens, which are informative for the BookRestaurant class. In (5), most methods predict PlayMusic because of the "play" token, and only the hierarchical approach assigns the correct pseudo-label. The sixth example (6) is interesting: in this sentence, "shivers in summer" is the name of a movie. While there could be a mis-understanding caused by the word "summer" – which might refer to the GetWeather class–, the hierarchical approach is able to see beyond this token and assign the correct pseudo-label. Moreover, aggregating pseudo-labels from different methods will intuitively discard a lot of pseudo-labels. This effect becomes more and more important as we increase the amount of methods aggregated together. In our experiments, we aggregate 4 methods: `matching`, `Proto`, `Proto++` and our contribution, `Fold/Unfold`. As shown in Table 4.2, using 5 shots, aggregating pseudo-labels yields a discard rate of 10.2%, 25.1%, 47.8%, 45.5% for snips, Clinc, Liu and R8 datasets respectively. In Table 5.3, we report all results from pseudo-labeling as well as intent detection performance score. In order to measure the quality of pseudo-labels, we compute their F1-score. For the intent detection part, we report the F1-score on the held-out 10% of the testing data.To measure this trade-off between recall and precision of pseudo-labels, we will further discuss the impacts on the intent detection performances for each dataset. Additionally, we report results of the different pseudo-labeling methods using a fixed set of embeddings in Table 4.4. As showcased in this Table, our fold/unfold method performs the best given the same fixed set of sentence representations. Also note that `matching` and Proto are equivalent in the 1-shot case: while one compares the distance to the average embedding, the other compares the average distance to embeddings. When only one labeled sample is available, both these techniques are equivalent.

Snips On the `Snips` dataset, even though our Folding/Unfolding approach outperforms all competitors, those results must be taken lightly. As we can see, with only 5 shots, we are able to recover pseudo-labels and build an intent detection model

Sentence	Matching	Prototypical	Hierarchical
(1) book for 8 am in massachusetts for 1	RateBook	SearchCreativeWork	**BookRestaurant**
(2) which movies are playing at b&b theatres at 2 pm	RateBook	**SearchScreeningEvent**	**SearchScreeningEvent**
(3) in 1 hour and 1 minute find a cinema nearest for films	RateBook	BookRestaurant	BookRestaurant
(4) forecast for wisconsin at 10 pm	RateBook	BookRestaurant	BookRestaurant
(5) i want to play the game show me the wonder	PlayMusic	PlayMusic	**SearchCreativeWork**
(6) when can i catch a screening of shivers in summer	BookRestaurant	BookRestaurant	**SearchScreeningEvent**

Table 4.1: Some examples of disagreement on recovered labels for the three methods. For each sentence, the correct label is in **bold**.

Dataset	Amount of given shots		
	1	2	5
Snips	0.210	0.173	0.102
R8	0.642	0.513	0.455
Clinc	0.468	0.337	0.251
Liu	0.689	0.586	0.478

Table 4.2: Fraction (between 0 and 1) of discarded pseudo-labels by the aggregated method.

		Pseudo-label F1			Intent Detection F1		
	shots	1	2	5	1	2	5
Snips	Proto	0.914	0.915	0.948	0.913	0.921	0.947
	Proto++	0.862	0.898	0.920	0.863	0.899	0.920
	BERT-Fit+`matching`	0.943	0.958	0.985	0.945	0.957	0.985
	BERT-Fit+Fold/Unfold	**0.986**	**0.982**	**0.992**	**0.988***	**0.983**	0.992
	BERT-Fit+Fold/Unfold+weights	-	-	-	0.987	0.983	**0.992**
	aggregated	**0.994**	**0.996**	**0.997**	0.983	0.983	0.992
	$\text{agg}_{-\text{BERT-Fit+matching}}$	0.994	0.996	0.997	**0.985**	**0.986**	**0.992**
	$\text{agg}_{-\text{Proto}}$	0.993	0.988	0.996	0.984	0.980	0.991
	$\text{agg}_{-\text{Proto++}}$	0.991	0.995	0.995	0.984	0.982	0.991
	$\text{agg}_{-\text{BERT-Fit+Fold/Unfold}}$	0.988	0.993	0.995	0.973	0.978	0.988
	shots-only	-	-	-	0.948	0.965	0.987
Clinc	Proto	0.644	0.738	0.807	0.648	0.749	0.829
	Proto++	0.415	0.512	0.622	0.416	0.520	0.640
	BERT-Fit+`matching`	0.723	0.831	0.895	0.725	0.837	0.902
	BERT-Fit+Fold/Unfold	**0.883**	**0.930***	**0.939**	0.878	0.932	**0.945***
	BERT-Fit+Fold/Unfold+weights	-	-	-	**0.880***	**0.932***	0.944
	aggregated	**0.890**	**0.921**	**0.940**	0.839	0.890	0.927
	$\text{agg}_{-\text{BERT-Fit+matching}}$	0.880	0.912	0.933	**0.851**	**0.908**	**0.940**
	$\text{agg}_{-\text{Proto}}$	0.856	0.899	0.925	0.848	0.890	0.929
	$\text{agg}_{-\text{Proto++}}$	0.856	0.903	0.928	0.849	0.887	0.928
	$\text{agg}_{-\text{BERT-Fit+Fold/Unfold}}$	0.844	0.896	0.931	0.784	0.859	0.913
	shots-only	-	-	-	0.714	0.830	0.901
Liu	Proto	0.513	0.618	0.697	0.515	0.624	0.694
	Proto++	0.467	0.534	0.605	0.472	0.535	0.605
	BERT-Fit+`matching`	0.500	0.601	0.642	0.505	0.608	0.645
	BERT-Fit+Fold/Unfold	**0.587**	**0.668**	**0.763**	0.537	0.636	**0.728**
	BERT-Fit+Fold/Unfold+weights	-	-	-	**0.539**	**0.637**	0.726
	aggregated	**0.852**	**0.895**	0.902	0.585	0.686	0.747
	$\text{agg}_{-\text{BERT-Fit+matching}}$	0.829	0.887	**0.905**	**0.590**	**0.702**	**0.778**
	$\text{agg}_{-\text{Proto}}$	0.828	0.874	0.894	0.584	0.679	0.749
	$\text{agg}_{-\text{Proto++}}$	0.824	0.864	0.874	0.572	0.678	0.740
	$\text{agg}_{-\text{BERT-Fit+Fold/Unfold}}$	0.722	0.819	0.840	0.576	0.665	0.715
	shots-only	-	-	-	0.443	0.620	0.756
R8	Proto	0.575	0.629	0.709	0.598	0.650	0.725
	Proto++	0.556	0.620	0.647	0.566	0.607	0.628
	BERT-Fit+`matching`	0.737	0.789	0.890	0.758	0.799	0.885
	BERT-Fit+Fold/Unfold	**0.816**	**0.837**	**0.901**	0.817	**0.834***	0.901
	BERT-Fit+Fold/Unfold+weights	-	-	-	**0.817**	0.831	**0.902***
	aggregated	**0.890**	**0.921**	**0.940**	0.813	0.824	0.894
	$\text{agg}_{-\text{BERT-Fit+matching}}$	0.880	0.912	0.933	0.806	0.815	**0.900**
	$\text{agg}_{-\text{Proto}}$	0.856	0.899	0.925	**0.817**	**0.822**	0.900
	$\text{agg}_{-\text{Proto++}}$	0.856	0.903	0.928	0.808	0.822	0.893
	$\text{agg}_{-\text{BERT-Fit+Fold/Unfold}}$	0.844	0.896	0.931	0.767	0.790	0.895
	shots-only	-	-	-	0.738	0.780	0.892

Table 4.3: Evaluation of pseudo-labels for the **label recovery** and **intent detection** tasks. We separate results from single and aggregated methods, highlighting in **bold** the best method in each category. If a single method is better than the best aggregated one, it is highlighted with a wildcard (*). The $\text{agg}_{-\lambda}$ line represents the aggregation between all single methods **except method** λ. For the different aggregations, we underline the worst (i.e. stressing the method contributing the most to the aggregation).

Embedding	PL method	BANKING77		HWU64	
		PL Acc.	IC f1	PL Acc.	IC f1
Prototypical	-	-	0.639	-	0.732
Prototypical	Prototypical	0.595	0.590	0.691	0.680
Prototypical	Matching	0.595	0.590	0.691	0.680
Prototypical	Fold/Unfold	**0.664**	**0.663**	**0.777**	**0.772**

Table 4.4: 1-shot pseudo-label (PL) accuracy and intent classification (IC) f1-score on BANKING77 and HWU64 datasets. The same sentence representations are fed to each pseudo-labeling method.

with a 0.988 F1-score. Those huge performances clearly hint that this dataset is not challenging: the t-SNE projection for this dataset in Figure 3.1 (p. 58) illustrates this claim. This means that we can hardly make concluding remarks on how the systems relatively perform on more realistic datasets. Overall, the massive usage of SNIPS to evaluate intent detection techniques in the literature [149, 51] is disturbing.

Clinc On the `Clinc` dataset, our method largely comes out on top, both on pseudo-label and intent detection results. On the intent detection part, our method is better than the aggregated ones, showing that other methods penalize the aggregation. This finding is very important, as it proves that using an ensemble of various methods do not always lead to better performances – introducing wrongly labeled utterances strongly penalizes a learning process. Concerning the prototypical baselines, this is the dataset where the Proto++ is the further away from its former version, Proto. This comes from the fact that this dataset has the highest number of classes: when doing a soft-KMeans step, if the number of classes is high, then it is as much noise for the prototype refinement. Results on this dataset prove that our method is not handicapped when using a large number of classes, making it more robust to real-life scenarios. Additionally, we see that removing the `BERT-Fit+nKNN` from the aggregation yields better results. This poor performance was not to be seen in the pseudo-label results, as this baseline was performing second behind our approach. This raises the importance of using this double experiment, and confirms that better pseudo-labels do not directly imply a better model in the downstream task of intent detection.

Liu As explained in Section 2.5, this dataset is very imbalanced. Still, we get the same results as the two previous datasets, which were more balanced: our method is able to come out on top on both pseudo-labeling as well as intent detection results. Additionally, the ablation study where we withdraw our method from the aggregation yields the worst results in almost every situation, hinting that we contribute the most to this aggregation. This shows that our method is robust to various situations, including the case of imbalanced datasets.

R8 On the `R8` dataset, even though we now deal with text classification and much longer sentences, we have similar results as on others intent detection datasets. Our method comes out on top, even achieving better performances than aggregated methods during the classification step. This hints that our Folding/Unfolding method is robust, and not only suited for intent detection tasks, but also for the more general subject of text classification.

4.5.3 Aggregated approach

On all datasets, aggregating pseudo-labels obtained from single methods always improves the pseudo-label quality, at the cost of retrieving less pseudo-labels. However, this increment is not fully reflected in the intent detection performance results: most of times, there is a single method which, when discarded from the aggregation, yields better text classification results. This is important to consider in practice, and to have in mind that better pseudo-labels does not necessarily imply better results for the intent detection downstream task.

Overall, in the ablation study where we withdraw each single method from the aggregation, the aggregation without our method is very often the worst. This additional study shows that our method contributes the most to the aggregation, as removing it is often the worst case.

4.5.4 Weighting mechanism

When training the intent detection model with pseudo-labels recovered by our method, we can use or ignore the confidence weights. The goal of those weights is to penalize pseudo-labels which might have been incorrectly assigned (i.e. with small

confidence). We show that the weighting mechanism has a limited impact on the classification task. Remember that our method does not assign a pseudo-label to every unlabeled utterance: in step (b) 4.4.1, it can discard clusters which are far from one another. Through this, our method intrinsically already discards hard-to-predict utterances – we show that our framework is already close optimal in this regard: a weighting scheme do not contribute to improving our algorithm. We find this an strong advantage of our contribution since it therefore does not require a weighting hyper-parameter, which makes it simpler to train and do not require any grid search.

4.5.5 Proto and Proto++

We discuss here the performances of prototypical baselines. In our experiments, we showed that the Proto++ variant performs less than its former version, Proto. Using a BERT model only fine-tuned on the masked language modeling task directly for the Proto++ also introduces a lot of noise: because embeddings are not well separated, adding 20 unlabeled samples per class adds to much noise for the model to separate classes. This effect is amplified on `Clinc` dataset, as the number of classes is high.

4.6 Conclusion

In this chapter we introduced a new state-of-the-art, hierarchical clustering inspired method for pseudo-labeling intent detection datasets. This is of the utmost practical interest in order to help linguists to faster provide new intent detection model as the intent classes repository evolve with time. Our method, being hyper-parameter-free, is robust to various situations, even when the number of classes is high. By discarding clusters which are far from known classes, it is able to mitigate the noise which would have been introduced in the unlabeled dataset. Through an ablation study, we also show that our method is complementary to other baselines so that it contributes to aggregated approaches significantly. We also demonstrated that if we continue to evaluate intent detection using very easy – yet popular – datasets like Snips, we are ultimately not solving the actual downstream task in practice, so we encourage any intent detection researchers and practitioners to use it with caution. Additionally, our method is not end-to-end, and as such, can be seen as complex. Indeed, we saw that the accuracy on pseudo-label predictions is not always perfectly

correlated with the accuracy on the downstream intent detection task. In practice, as we do not have many label intent utterance in a few-shots settings, this can set false expectations on the performance of the downstream model looking only on pseudo-labels predictions. In the next chapter, we will introduce a semi-supervised end-to-end system, which is a novel extension of prototypical networks making use of unlabeled data.

Chapter 5

PROTAUGMENT: Unsupervised diverse short-texts paraphrasing for intent detection meta-learning

This chapter is based on the following publication [36]

> Thomas Dopierre, Christophe Gravier, Wilfried Logerais. "PROTAUGMENT: Intent Detection Meta-Learning through Unsupervised Diverse Paraphrasing". In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL2021)*, pages 2454–2466, 2021.

5.1 Introduction

Training a classification model with very few labeled data is usually a problem. For conversational agents, unlabeled utterances are fortunately cheap to acquire, as users are interacting with the agent on a regular basis. While unlabeled data do not always match known classes, they still contain knowledge to learn from. As users have their own way of expressing themselves, and this can lead to difficulties during training. However, if the model is able to figure out those ways of expressing

requests, then it will surely help in solving the downstream task, even though the initial data the model was trained on does not match the data used for this task. This is what happens when using large pre-trained language models. When they are trained on a large corpus of unlabeled data, the goal is to make the model understand the language structure, and how sentences are constructed. Transfer learning benefits from this pre-training, and ultimately, it leads to better performances on many downstream tasks [33, 19].

We already studied how different few-shot end-to-end system performed on the task of intent classification (Chapter 3). In those experiments, we found that prototypical networks were the best approach to such problem. We then introduced a novel pseudo-labeling method (Chapter 4) to make use of unlabeled data, which are cheap to acquire. Our pseudo-labeling approach yields good results, and pseudo-labels can then be verified by linguist teams. The better the quality of pseudo-labels, the quicker the verification of pseudo-labels will be. Labeling utterances is a painful job, hence if we can make it quicker, it will hopefully be less sore and with less errors. As discussed in the conclusion of the last Chapter, this approach is nonetheless not end-to-end, therefore prone to contribute to accumulate error in a NLP pipeline.

Consequently, our idea for this third and last contribution chapter was to focus back on end-to-end system, but that is able to learn from unlabeled samples. Starting with the back-again state-of-the-art which are prototypical networks, we wanted to explore how unlabeled data could be useful for such networks, in hope to further improve it. A major challenge comes in the form of text diversity: ultimately we want diverse data (labeled or unlabeled) to gather knowledge about given class. Yet, diverse data means representations that are far from each other, hence it will be harder to group them into clusters – most few shot neural approaches approaches actually aim at clustering or quantifying data (Chapter 3). This diversity is also hard to manage in a context of small utterances like chatbots: small modifications to small texts quickly lead to utterances with a different meaning, possibly falling into another intent class without detecting it.

Inspired by [150], we introduce an unsupervised diverse paraphrasing loss in the Prototypical Networks framework. A key idea is consistency learning: by augmenting unlabeled user utterances, our approach, which we call PROTAUGMENT, enforces a more robust text representation learning. Unfortunately, as we will see in

this chapter, back-translation is a poor data augmentation strategy for short-texts. Indeed, in our case, neural machine translation provides very similar (if not the same) sentences to the original ones, which hinders its ability to provide diverse augmentations (see Section 5.4.3). Consequently, in this work, we transfer a de-noising autoencoder pre-trained on the sequence-to-sequence task [82] to the paraphrase generation task and then use it to generate paraphrases. As fine-tuning is very efficient for such a model, it is not easy to optimize it to generate **diverse** paraphrases. [53] presents an approach for diverse paraphrasing that reorders the original sentence to guide the conditional language model to generate diverse sentences. The diversity in that work is provided by the reordering of the elements, which surprisingly affects the attention mechanism. In [86], expression diversity is part of the unsupervised paraphrasing system supported by simulated annealing. Both approaches imply domain transfer, and consequently, as many diverse paraphrasing models to maintain as the number of considered application domains, which do not scale very well. In this work, we instead introduce diversity in the downstream decoding algorithm used for paraphrase generation. Diverse decoding methods are mostly extensions to the beam search algorithm, including noise-based algorithms [27], iterative beam search [72], clustered beam search [134], and diverse beam search [138]. There is no clear optimal solution, the choice is task-specific and dependent on one's tolerance for lower quality outputs as a diversity/fluency trade-off [61]. While diverse beam search allows controlling the diversity/fluency trade-off partially, we further demonstrate that adding constraints to diverse beam search in order to generate tokens not seen in the input sentence (that is, *constrained diverse beam search*) is a simple yet powerful strategy to further improve the diversity of the paraphrases. Paired with paraphrasing user utterances and its consistency loss incorporated in Prototypical networks, our model is the best method for intent detection meta-learning on 4 public datasets, with neither extra labeling efforts nor domain-specific conditional language model fine-tuning. We also show that PROTAUGMENT, having access to only 10 samples of each class of the training data, still significantly outperforms a Prototypical Network which is given access to *all* samples of the same training data.

5.2 Notations (reminder)

We remind here the notations that we use relatively to meta-learning. Meta-learning algorithms are trained using a specific procedure made of consecutive episodes. Let $\mathcal{C}_{ep}$ be the set of C classes sampled for the current training episode, such as $\mathcal{C}_{ep} \subset \mathcal{C}_{train}$, where $\mathcal{C}_{train}$ is the set of all classes available for training. We note $\mathcal{C}_{test}$, the set of classes used for testing, with $\mathcal{C}_{train} \cap \mathcal{C}_{test} = \emptyset$. Each class $c \in \mathcal{C}_{ep}$ comes with K labeled samples, used as support. The set of $C \times K$ samples are usually referred to as $\mathcal{S}$, the support set, so that $\mathcal{S} = \{(x_1, y_1), \dots, (x_{C\times K}, y_{C\times K})\}$. We denote S_c the set of support examples labeled with class c. Each episode comes with a query set $\mathcal{Q}$, which serves as the episode-scale optimization – the model parameters are updated based on the prediction loss on $\mathcal{Q}$, given $\mathcal{S}$ as an input. $\mathcal{Q}_c$ is the set of query examples labeled with class c.

5.3 PROTAUGMENT

In this section, we present our semi-supervised approach PROTAUGMENT. Along with the labeled data randomly chosen at each episode in prototypical networks, this approach uses U, the set of unlabeled data randomly drawn from the whole dataset – that is, data from training, validation, and test labels. We first do a data augmentation step from this unlabeled data, where we obtain M paraphrases for each unlabeled sentence. The m^{th} paraphrase of x will be denoted $\tilde{x}^m$. Then, given unlabeled data and their paraphrases, we compute a fully unsupervised consistency loss (we do not enforce a given class to both samples, but instead we enforce that they should belong to the same one). That loss is noted as $\tilde{L}$. Finally, we combine both the supervised loss $\bar{L}$ (the Prototypical Network loss using labeled data) and unsupervised loss ($\tilde{L}$) and run back-propagation to update the model's parameters.

5.3.1 Generating augmentations through paraphrasing

The BART [82] model is a Transformer-based neural machine translation architecture that is trained to remove artificially corrupted text from the input thanks to an autoencoder architecture. While it is trained to reconstruct the original noised input,

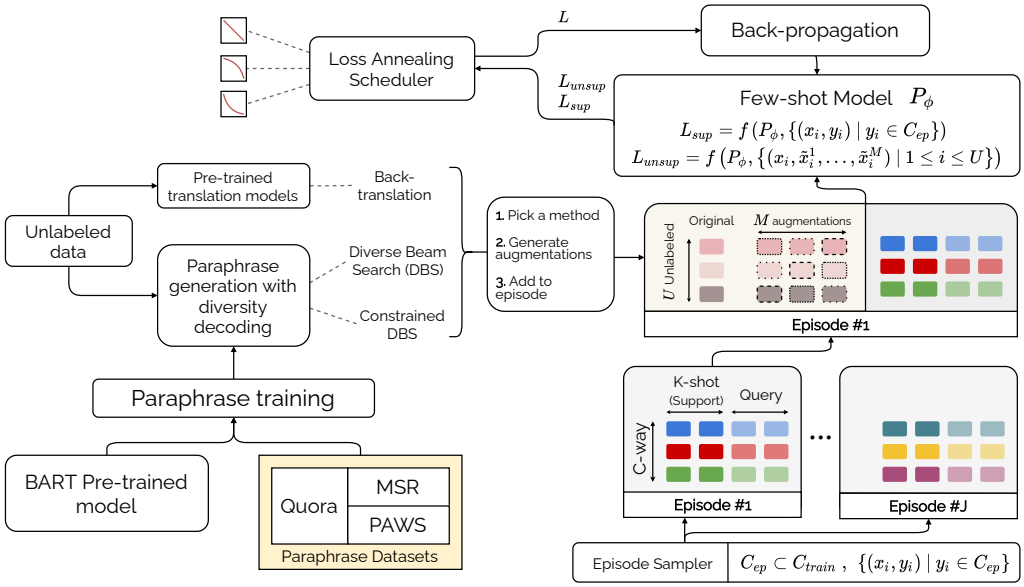


Figure 5.1: PROTAUGMENT illustrated on a 3-way 2-shot short text classification meta-learning task ($C = 3, K = 2$). BART is pre-trained for the paraphrasing task on three datasets: Quora [126], MSR [164] and Google PAWS-Wiki [155, 163]. The paraphrase model is used to paraphrase unlabeled samples but equipped with diversity strategies (back translation being proposed as a baseline). The final loss is computed using a loss annealing scheduler, which is expected to smooth the supervised (given shots) and unsupervised (augmented unlabeled sentences) prediction errors to yield parameter gradients. A new episode means sampling other classes along with their support and query points.

it can be fine-tuned for task-specific conditional generation by minimizing the cross-entropy loss on new training input-output pairs [13]. In PROTAUGMENT, we fine-tune a pre-trained BART model on the paraphrasing task. The paraphrase sentence pairs we use for this task are taken from 3 different paraphrase detection datasets[1]: Quora [126], MSR [164], and Google PAWS-Wiki [155, 163]. Those datasets have different sizes, and the largest one – Quora – consist of 149,263 pairs of duplicate questions. To balance turns of sentences (questions/non questions paraphrases), 50% of our fine-tuning paraphrase datasets is made of Quora, 5.6% of MSR and 44.4% PAWS-Wiki. This yields 94,702 sentence pairs to train the model on the paraphrasing task. We include both code and data on our github repository [2].

[1] we take only pairs that are paraphrases of each other since these are *paraphrase detection* datasets
[2] https://github.com/tdopierre/ProtAugment

Using this fine-trained paraphrasing model, we can generate paraphrases of unlabeled sentences, hopefully having paraphrases representing the same intents as the original sentences. To add some diversity in the generated paraphrases, we use Diverse Beam Search (DBS) instead of the regular Beam Search. As [138] has shown in the original paper, adding a dissimilarity term during the decoding step helps the model produce sequences that are quite far from each other while still retaining the same meaning. The next section describes how we constrained this decoding to enforce even more diversity among generated paraphrases in PROTAUGMENT.

5.3.2 Constrained user utterances generation

While DBS enforces diversity between the generated sentences, it does not ensure diversity between the generated paraphrases and the original sentences. It was formerly designed for tasks that do not need this diversity with the original sentence (translation, image captioning, question generation). To enforce that, our generated paraphrases are diverse enough, we further constraint DBS by forbidding using parts of the original sentences. In the following paragraphs, we introduce two forbidding strategies.

Unigram Masking. In this strategy, we randomly select tokens from the input sentence which will be forbidden at the generation step. The goal here is to force the model to use different words in the generated sentences than it saw in the original sentences. Each word of the input sentence is randomly masked using a probability p_{mask}. The underlying assumption is that forbidding tokens at the beginning of a sentence with a higher probability than the end of the sentence may have a greater impact on the beam search algorithm. Indeed, as the decoding is a conditional task based on prior generated tokens, masking the first tokens may significantly impact diversity. We therefore introduce two additional variants: one where we put more probability on the first tokens and the reverse where there is more weight in the last tokens. To ensure that all three variants mask the same amount of tokens on average, we ensure the area under the curve of the three probability functions are equal to a fixed value noted p_{mask}.

Bi-gram Masking Another strategy we consider is to prevent the paraphrasing model from generating the same bi-grams as in the original sentence. This time, we are not masking any single word but forcing the model to change the sentence's structure, which will, hopefully, increase the diversity of the generated paraphrases.

We could go take the strategies further, and introduce n-gram masking, where we take $n > 2$. However, notice that n-gram masking indirectly implies m-gram masking for all $m > n$. For example, if bi-grams are forbidden, then indirectly, tri-grams are also forbidden. Indeed, tri-grams are composed of two bi-grams. As such, we only experimented with uni-gram and bi-gram masking, which are more restrictive than tri-gram masking and any n-gram masking for $n > 2$.

5.3.3 Unsupervised diverse paraphrasing loss

After generating paraphrases for each unlabeled sentence, we create unlabeled prototypes. For each unlabeled sentence $x_u \in U$, we derive the unlabeled prototype p_{x_u} as the average embedding of the paraphrases of x_u (Equation 5.1). Our goal is to mimic the way prototypical networks work, but with unlabeled data.

$$p_{x_u} = \frac{1}{M} \sum_{m=1}^{M} f_\phi(\tilde{x}_u^m) \tag{5.1}$$

After obtaining the unlabeled prototypes, we compute the distances between all unlabeled samples and all unlabeled prototypes. Given such distances, we model the probability of each unlabeled sample being assigned to each unlabeled prototype (Equation 5.2), as in the supervised part of the Prototypical Networks – except this time, it is fully unsupervised. This probability should be close to 1 between an unlabeled sample and its associated unlabeled prototype and close to 0 otherwise.

$$\mathbb{P}_\phi(u = v|x_u) = \text{softmax}\left(-d(f_\phi(x_u), p_{x_v})\right) \tag{5.2}$$

We remind here the notation we used to describe Prototypical Networks (Chapter 2, Section 2.4.2.2): f_ϕ is an embedding function with learnable set of parameters ϕ – this is the function we want to optimize. Moreover, d is a distance function, usually euclidean or cosine, and it is set to euclidean in our case as this yields the best results.

Given assign probabilities between unlabeled samples and unlabeled prototypes, we can compute a fully unsupervised cross-entropy loss $\tilde{L}$, training the model to bring each sentence closer to its augmentations' prototype and further from the prototypes of other unlabeled sentences.

After obtaining both supervised loss $\bar{L}$ and unsupervised loss $\tilde{L}$, we combine them into the final loss L using a loss annealing scheduler (see Equation 5.3), which will gradually incorporate the unsupervised loss as training progresses.

$$L = t^{\alpha} \times \tilde{L} + (1 - t^{\alpha}) \times \bar{L} \quad ; \quad t \in (0, 1) \tag{5.3}$$

The goal here is to mainly use the supervised loss first so that the model gets a sense of the classification task. Then, incorporating more and more knowledge from unlabeled samples will make the model more robust to noise, which is essential as it is constantly tested on classes it has never seen before. We explore three different strategies for gradually increasing the unsupervised contribution: a linear approach ($\alpha = 1$), an aggressive one ($\alpha = 0.25$), and a conservative one ($\alpha = 4$). In a sense, our unsupervised loss can be viewed as a supervised loss using an unsupervised episode. Indeed, it is as if we had an episode where each "class" is an unlabeled data point. For each of those classes, the support points are 5 augmentations of this unlabeled sentence. In this unsupervised episode, there is only one query point per class, and this query point corresponds to the unlabeled sentence. By utilizing the same mechanism in both labeled and unlabeled losses, we are helping the model structure its euclidean space in a good way, using both labeled and unlabeled data.

5.4 Experiments

5.4.1 Datasets

We consider the DialoGLUE benchmark [97], a set of natural language understanding benchmark for task-oriented dialogue, which contains three datasets for intent detection: `Banking77`, `HWU64` and `Clinc150` – the three datasets were already available prior the release of DialoGLUE. Additionally, we also consider the `Liu57` intent detection dataset, as it contains the same order of magnitude of intent classes and is user-generated as well. All datasets are public and in English. We already presented

these datasets previously (Chapter 2, Section 2.5), and we provide a quick reminder on their main characteristics here.

Banking77 The `Banking77` dataset [21] classifies 13,083 user utterances related to into 77 different intents. This dataset i) is specific to a single domain (banking) and ii) requires a fine-grained understanding to classify due to intents being very similar. Following [97] and contrary to [21], we designate a validation set along a training and a testing set for that dataset (Table 2.4).

HWU64 `HWU64` [151] classifies 25,716 user utterances with 64 user intents. It features intents spanning across 21 domains (alarm, audio, audiobook, calendar, cooking, datetime, ...). When separating training, validation, and test labels, we ensure each domain is represented only in one set of labels. This ensures the model learns to discriminate between both intents and domains.

Clinc This dataset [76] classifies 150 user intents in perfectly equally-distributed classes. This chatbot-like style dataset was initially designed to detect out-of-scope queries, though, in our experiments, we discard the out-of-scope class and only keep the 150 labeled classes to work with, as in [97].

Liu57 Introduced by [87], this intent detection dataset is composed of 54 classes. It was collected on Amazon Mechanical Turk, where workers were asked to formulate queries for a given intent with their own words. It is highly imbalanced: the most (resp. least) common class holds 5,920 (resp. 24) samples

5.4.2 Experimental settings

Conditional language model and language model. For the BART fine-tuning process, we used the defaults hyper-parameters reported in [82], and we fine-tuned the BART model for a single epoch (two hours on a Titan RTX GPU). Increasing the number of epochs for fine-tuning BART degrades performances on the intent detection task: the downstream diverse beam search struggles to find diverse enough beam groups since the model perplexity has been lower with further fine-tuning (this is also hinted in [13]). Our text encoder f_ϕ is a `bert-base` model, and the embedding

of a given sentence is the last layer hidden state of the first token of this sentence. For each dataset, this model is fine-tuned on the masked language modeling task for 20 epochs. Then, the encoder of our meta learner is initialized using the weights of this fine-tuned model.

Datasets From a dataset point-of-view, we create two data profiles: **full** (all the training dataset is available, the usual meta-learning scenario) and **low** (only 10 samples are available for each training class, an even more challenging meta-learning scenario in which a model meta-learns on very few samples per training class). All experimental setups are run 5 times. For each run, we randomly select training, validation, and testing classes, as well as the samples for the **low** setting. We train the few-shot models for a maximum of $10,000$ C-way K-shots episodes, evaluating and testing every 100 episodes, stopping early if the evaluation accuracy has not progressed for at least 20 evaluations. We evaluate and test using 600 episodes, as in other few-shot works [129, 26]. We compare the systems in the following standard few-shot evaluation scenarios: 5-way 1-shot, and 5-way 5-shots.

Paraphrasing. At each episode, we draw $U = 5$ unlabeled samples to generate paraphrases from. For the back-translation baseline, we use the publicly available[3] translation models from the `Helsinki-NLP` team. We use the following pivot languages: `fr`, `es`, `it`, `de`, `nl`, which yields 5 augmentations for each unlabeled sentence. We also considered EDA [145] as a data augmentation technique. For this approach, we use the base parameters pointed by authors, which are 10% synonym replacements, 10% random insertions, 10% random swaps, and 10% random deletions. To generate augmentations with EDA, we use the publicly available code by the authors [4]. For our experiments with Diverse Beam Search, we generate sentences using 15 beams, group them into 5 groups of 3 beams. In each group, we select the generated sentence which is the most different from the input sentence using BLEU as a metric for diversity. This yields $M = 5$ paraphrases for each unlabeled sentence, as in the back-translation baseline. DBS uses a diversity penalty parameter to penalize words that have already been generated by other beams to enforce diversity. As advised in the original DBS paper [138], we set the diversity penalty to 0.5

[3] https://huggingface.co/models?search=helsinki-nlp

[4] `https://github.com/jasonwei20/eda_nlp/`

in our experiments, which provides diversity while limiting model hallucinations. Our Unigram Masking strategy's masking probability is set to $p_{\text{mask}} = 0.7$ found by linear search from 0 to 1 with steps of 0.1.

orig: How long will my transfer be pending for? *back*: How long will my transfer be on hold? *dbs_0*: How long will my transfer be pending? I am in first year. *dsb_1*: When are all transfers coming up and how many days are they expected? *dbs_2*: If I have a transfer for a while, how long should I wait for it?
orig: I am not sure where my phone is. *back*: I don't know where my phone is. *dbs_0*: I am not really sure where my phone is located *dsb_1*: How can I find the location of any Android mobile *dbs_2*: I don't know where is my cell phone
orig: can you play m3 file *back*: can you read m3 file *dbs_0*: M3 files: can I play the entire M3 file? *dsb_1*: Is there any way to play 3M files on Earth without downloading it *dbs_2*: Is there any way to play M3 files on Windows?

Table 5.1: Examples of sentences (*orig*) paraphrased using back translation (*back*), vanilla diverse beam search – DBS (*dbs_0*), DBS with unigram masking (*dbs_1*) and DBS with bigram masking (*dbs_2*)..

5.4.3 Evaluation of paraphrase diversity

We evaluate the diversity of paraphrases for each method, and report results for two representative datasets in Table 5.2. For each paraphrasing method and each dataset, metrics are computed over unlabeled sentences and their paraphrases. To assess the diversity of paraphrases generated by the different methods, the popular BLEU metric in Neural Machine Translation is a poor choice [9]. We still report it for reference. To better assess the paraphrase diversity, we use the bi-gram diversity (**dist-2**) metric as proposed by [61], which computes the number of distinct 2-grams divided by the total amount of tokens. We also report the average sentence similarity (denoted **use**) within each sentence set, using the Universal Sentence Encoder as an independent sentence encoder. The higher the sentence similarity, the less diverse the paraphrases. Results show that paraphrases obtained with back-translation are too close to each other, resulting in a high sentence similarity and low bi-gram diversity. On

the other hand, paraphrases issued by DBS are more diverse, and less similar to original sentences. Our masking strategies strengthen this effect and yield even more diversity. While our methods allows to generate paraphrases which are more diverse, we still have to find out if they are not too diverse. If they are, then they might add too much noise to the classifier, resulting in a performance decrease. We will discuss this effect in the results section, where we will find out measured diversity strongly correlates with the average accuracy of the intent detection task (Table 5.3).

Dataset	**Method**	**Metrics**		
		BLEU↓	dist-2↑	use↓
BANKING77	back-translation	56.0	0.183	0.896
	DBS	34.2	0.200	0.807
	DBS+bigram	**0.1**	0.228	0.702
	DBS+unigram	0.2	**0.343**	**0.613**
HWU64	back-translation	40.2	0.307	0.888
	DBS	19.5	0.340	0.769
	DBS+bigram	**0.1**	0.350	0.692
	DBS+unigram	0.5	**0.407**	**0.628**
Liu	back-translation	47.7	0.268	0.892
	DBS	19.7	0.293	0.750
	DBS+bigram	**0.4**	0.293	0.664
	DBS+unigram	0.5	**0.351**	**0.596**
Clinc	back-translation	43.9	0.205	0.903
	DBS	22.3	0.236	0.805
	DBS+bigram	**0.2**	0.257	0.717
	DBS+unigram	0.3	**0.323**	**0.644**

Table 5.2: Paraphrase diversity evaluation on all 4 datasets. The unigram variant exposed here is using the *flat* masking strategy with $p_{\text{mask}} = 0.7$. Up arrow ↑(resp. down arrow ↓) indicate that higher (resp. lower) value results in higher diversity.

5.4.4 Intent detection results

In this section, we discuss the accuracy results for the different meta-learners, for the standard 5-way and {1, 5}-shots meta-learning scenarios, as provided in Table 5.3. The reported metric is the accuracy on the test set at the iteration where the validation set's accuracy is maximal. Our DBS+unigram strategy row corresponds to the

Profile	Method	Datasets								Accuracy stats ($AVG \pm STD$)	
		Banking		HWU		Liu		Clinc			
		$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$
low profile	Prototypical Network	82.20	91.57	74.37	86.48	80.06	89.62	94.29	98.10	82.73 ± 2.32	91.44 ± 1.92
	ours w/ BT	83.83	92.16	78.70	89.36	80.84	90.87	94.06	97.62	84.36 ± 1.15	92.50 ± 0.94
	ours w/ EDA	83.93	92.60	78.35	89.83	81.46	91.49	93.72	97.48	84.37 ± 1.29	92.85 ± 0.98
	ours w/ DBS	83.10	92.56	80.06	90.21	82.31	91.64	93.70	97.83	84.80 ± 1.26	93.06 ± 0.99
	ours w/ DBS+bigram	86.04	93.55	82.09	**91.57**	83.60	92.71	95.11	98.23	86.71 ± 1.14	94.01 ± 1.05
	ours w/ DBS+unigram	**87.23**	**94.29**	**83.70**	91.29	**85.16**	**93.00**	**95.92**	**98.56**	$\mathbf{88.00 \pm 1.22}$	$\mathbf{94.29 \pm 0.76}$
full profile	Prototypical Network	86.28	93.94	77.09	89.02	82.76	91.37	96.05	98.61	85.55 ± 2.20	93.24 ± 1.22
	ours w/ BT	87.46	94.47	81.31	91.44	84.14	92.67	95.19	98.36	87.02 ± 1.36	94.23 ± 0.82
	ours w/ EDA	87.55	94.74	81.13	91.20	84.62	92.86	94.95	98.27	87.06 ± 1.24	94.27 ± 0.74
	ours w/ DBS	86.94	94.50	82.35	91.68	84.42	92.62	94.85	98.41	87.14 ± 1.36	94.30 ± 0.60
	ours w/ DBS+bigram	88.14	94.70	84.05	92.14	85.29	93.23	95.77	98.50	88.31 ± 1.43	94.64 ± 0.59
	ours w/ DBS+unigram	**89.56**	**94.71**	**84.34**	**92.55**	**86.11**	**93.70**	**96.49**	**98.74**	$\mathbf{89.13 \pm 1.13}$	$\mathbf{94.92 \pm 0.57}$

Table 5.3: 5-way 1-shot and 5-way 5-shots accuracy on the test sets for each dataset. The *ours* method is PROTAUGMENT (unsupervised consistency loss using diverse paraphrases) equipped with different paraphrasing strategies. For each dataset × C-way K-shot setting, we compute the average and the standard deviation over the 5 runs (see Section 5.4.2), so that the last two columns contains average accuracy and ± the average standard deviations. For each data profile, we highlight the best method in **bold**. We underline the methods on the **low** profile which perform better than the Prototypical Networks on the **full** profile. We trained 400 different meta-learners – 5 methods, 2 data profiles, 4 datasets, 2 meta-learning setup ($K = 1, 5$) and 5 runs for each configuration.

`flat` masking strategy, with $p_{\text{mask}} = 0.7$. First, all methods augmented with unsupervised diverse paraphrasing outperform prototypical networks. However, back translation demonstrates only a limited improvement over the vanilla prototypical network due to their narrow diversity for short texts. The same conclusions can be derived for the EDA approach, yielding about the same results than back-translation. While this may come from the fact that we used small perturbations amount (i.e. 10% perturbation for all techniques of EDA), we respected the values recommended by authors of the EDA paper. Using paraphrases from DBS yields better results – about 0.5 points over BT, on average –, hinting that using diverse paraphrases in the unsupervised consistency loss allows the few-shot model to build more robust sentence representations and therefore provides improved generalization capacities. Those results are consistent across the different datasets, except for Clinc for which accuracies are all very high, making all methods hardly separable. The dataset is not challenging enough, or in other words, meta-learning is robust to unbalanced short text classification problems given the nature of that dataset.

These results illustrate the need for unsupervised paraphrasing and show that using diverse paraphrases provide a significant performance leap. In the 1-shot (resp. 5-shot) scenario, our best meta-learner improves prototypical networks by 5.27 (resp. 2.85) points on average. Remember that these improvements are made in an unsupervised manner hence at no additional cost. Slightly different from to [150], we do not find statistical differences depending on the rate at which $\tilde{L}$ is annealed in PROTAUGMENT loss ($\alpha \in \{0.25, 1, 4\}$), which makes it easier to tune – our unsupervised loss serves as a consistency regularization. Detailed results for this finding are available at Table 5.4.

	Datasets								**Accuracy stats**	
	Banking		HWU		Liu		Clinc		($AVG \pm STD$)	
α	$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$
1	87.23	94.29	83.70	91.29	85.16	93.00	95.92	98.56	88.00 ± 1.22	94.29 ± 0.76
0.25	86.71	94.17	82.71	91.19	85.52	93.11	95.99	98.44	87.73 ± 1.09	94.23 ± 0.85
4	86.90	94.14	83.26	92.35	84.48	93.17	95.69	98.49	87.58 ± 1.64	94.54 ± 0.81

Table 5.4: Performances of DBS+unigram strategies with different values of the loss annealing parameter α. All strategies use $p_{\text{mask}} = 0.7$. Overall, there is no significant difference when changing the value of α.

Adding our masking strategies on top of DBS has a significant impact on all datasets, with the unigram variant being up about 2 points over the vanilla DBS on average. On all datasets except Clinc, given only 10 labeled samples per class (**low** profile), it even outperforms the supervised baseline which is given the full training data (**full** profile). This means that PROTAUGMENT does better than prototypical networks with much less – 15 times, and up to 47 times, depending on the dataset – labeled sentences per class. Those results indicate that our method more than compensates for the lack of labeled data and that no matter the amount of data available for the training class, there is a performance ceiling you cannot overcome without adding unsupervised knowledge from the validation and test classes. In the **full** profile, when given all the training data, our method greatly surpasses the Prototypical Network – 3.58 points given 1 shot, on average. Moreover, PROTAUGMENT is not only suited for the case where very little training data is available (**low** profile): when sampling shots from the entire training dataset (**full** profile), it outperforms a fully supervised baseline. Furthermore, note that our method is consistently more stable than the supervised baselines, as its average standard deviation over the different runs is much lower than the vanilla Prototypical Network.

5.4.5 Masking strategies

Method	**Datasets**								**Accuracy stats**	
	Banking		HWU		Liu		Clinc		($AVG \pm STD$)	
	$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$	$K=1$	$K=5$
DBSu-*flat*	87.23	**94.29**	**83.70**	91.29	**85.16**	93.00	95.92	98.56	**88.00 ± 1.22**	94.29 ± 0.76
DBSu-*down*	**87.43**	94.14	83.06	**92.14**	84.87	**93.33**	**95.93**	**98.61**	87.82 ± 0.84	**94.55 ± 0.71**
DBSu-*up*	86.18	94.12	83.30	91.21	85.14	93.15	95.84	98.30	87.62 ± 1.23	94.20 ± 0.70

Table 5.5: Performances of DBS+unigram (noted *DBSu* in this table for the sake of clarity) strategies putting either more chance to mask first tokens (*down*), last tokens (*up*), or the same chance to all tokens (*flat*). All strategies use $p_{\text{mask}} = 0.7$. Overall, there is no significant difference between the three strategies.

We experimented with three variants of the unigram strategy (Section 5.3.2), each assigning a different drop chance to each token depending on its position in the input sentence. We reported the results for those experiments in Table 5.5. Overall, as numbers reported in this table indicate, we did not observe any significant difference in performance when putting more weight on the first tokens (*down*), or last

tokens (*up*), or the same weight on all tokens (*flat*). We also conducted experiments where we tune the value p_{mask}, from 0 to 1, with increments of 0.1. When set to 0, this corresponds to regular Diverse Beam Search, as no tokens are forbidden. When set to 1, all tokens of the input sentence are forbidden, and as such, none can be re-used in the generated paraphrase. We expose the results of those experiments in Figure 5.2, selecting 0.7 as the best trade-off. This figure also clearly shows that the *Clinc* dataset is one order of magnitude easier to solve than the other datasets. We stress here that we consider the very small sensitivity of our model to its hyper-parameters a very good and sought-after quality.

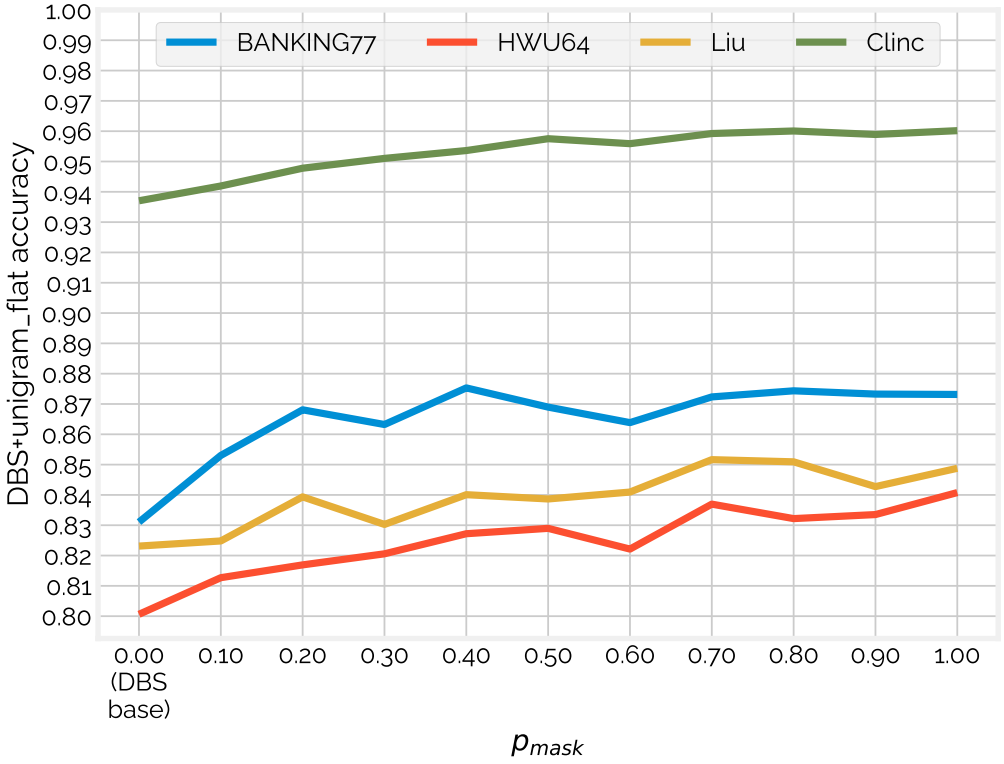


Figure 5.2: 5-way 1-shot accuracy of DBS-unigram-flat method using different values of p_{mask}. Setting this value to 0 corresponds to the vanilla DBS without masking strategies.

5.5 Conclusion & Future Work

Meta-learning is very significant for machine learning since it is inspired by how humans learn from sets of mini-batch tasks and how we then generalize to similar yet different configurations. In this chapter, we proposed PROTAUGMENT, an architecture for meta-learning for the problem of finding the intent of user-generated short-texts. We first introduced an unsupervised paraphrasing consistency loss in the prototypical network's framework to improve its representational power. This novel extension mimics the method of prototypical networks by training the model to retrieve the original sentence based on prototypes computed using paraphrases. On the paraphrase generation part, we demonstrated the lack of diversity for paraphrases issued by back-translation. While this technique has shown great results in other NLP related tasks, we found that it is not sufficient when used on short texts. Indeed, as showcased by the similarity scores, back-translation of short texts usually yields the same sentence as the original one, or very close to. When we used sentences augmented by the EDA method, it did not yield a very significant improvement over the prototypical network baseline. To generate paraphrases which are more diverse, we studied the diverse beam search algorithm, which was created to generate diverse caption for images. As we used it for paraphrase generation, we realised it was designed to enforce diversity between the generated texts, and it does not ensure diversity between the generated texts and the original sentences. To make up for the latter, we introduce constraints in the diverse beam search generation, further increasing the diversity. Our thorough evaluation demonstrates that PROTAUGMENT offers a significant leap in accuracy for the most recent and challenging datasets. PROTAUGMENT vastly outperforms prototypical networks, which was found to be the most competitive few-shot method for short-texts [37] with unsupervised-extended Prototypical Networks [117], against Matching Networks [139], Relation Networks [133], and Induction Networks [49], thereby making PROTAUGMENT the new state-of-the-art for this task. We provide the source code of PROTAUGMENT as well as code for evaluations reported in this chapter on a public repository [5].

[5] https://github.com/tdopierre/ProtAugment

Chapter 6

Conclusions

In this thesis, we studied various ways to answer the few-shot intent classification problem. In the first chapter, we introduced the thesis' context, as well as why labeling datasets is both time and energy costly. A gap emerges between large pre-trained models which are hungry in data, and the fact that such huge labeled datasets are not easy to gather. This calls for algorithms which are able to learn from small datasets, sometimes as small as having classes with only one sample. We then extensively described the technical background knowledge required to situate this thesis. Models applied to text have seen a very big shift in the last few years, from bag of words, to word embeddings, to transformers. With the increasing use of neural networks, classifiers have also changed a lot, usually growing in complexity and reaching better performances. We also completely defined the meta-learning framework, where learning is done in an episode-based fashion. In this framework, models are trained to quickly adapt to new tasks at test time, which fits real scenariis, where use-cases are evolving on a regular basis.

While few-shot learning models have made some progress in the recent years, the comparison between them is unfair, as they were not using the same text encoder at the time they were introduced. As a first contribution, to overcome this unfair comparison issue, we studied the performances of such few-shot learning models when they are equipped with a unique and recent transformer-based text encoder. Given this encoder, we showed that prototypical networks claim the state-of-the-art spot back. This finding is true both on the ARSC datasets, as well as 4 public intent detection datasets. Though few-shot learning methods were introduced because of

the inability of standard classifiers to perform well in low data regimes, we found that such traditional classifiers were not doing so poorly when used on top of transformers. Finally, we highlighted the importance of the distance metric, as our experiments with either euclidean or cosine yielded significantly better performances for the former. This finding must not be taken for granted: in the future, if a completely novel architecture is used to embed texts, it might completely change the structure of the embedding space. Given this new structure, another distance metric might be more suited. While, for us, the euclidean distance worked best, it is not perfect, and not all datasets are structured the same way. As a future line of research, one could find a way to extend such the euclidean distance to the metric learning domain. While in our experiments we did not find metric learning approaches based on relation modules particularly effective, one could experiment with a distance like the Mahalanobis [1], which could be initialised to be equivalent to the euclidean distance to favor the metric learning optimization.

As a second contribution, we focused on pseudo-labeling techniques. In a context where a large unlabeled dataset is available, one can utilize this dataset to derive pseudo-labels, and use both labeled and pseudo-labeled data to train a classification model. This forms a two-step process: pseudo-label generation and then, model training. We introduced a novel way to generate pseudo-labels, which is based on the hierarchical clustering mechanism. Our method is able to recover high quality pseudo-labels, voluntarily omitting data points which are hard to pseudo-label, hence which might correspond to noise. Additionally, it does not rely on any hyperparameter, making it robust to different configurations. Finally, our method highly complements others when aggregating pseudo-labels from various techniques. Our experiments considered only one round of pseudo-labeling, then trained an intent classification model on top. Still, we could have experimented with our algorithm by running more pseudo-labeling iterations like in self-training, co-training, or tri-training (see Section 2.4.3.1). Note nonetheless that our method has to be adapted to the self-training case. Indeed, after running one iteration of pseudo-labeling, running the folding step would give the same tree structure, as this step does not take the labels into account, and sentence representations would be the same. Additionally, our unfolding step would also yield the same decomposition, and ultimately, our set

[1] https://en.wikipedia.org/wiki/Mahalanobis_distance

of pseudo-labels would not change. For co-training or tri-training, this would not be an issue, as pseudo-labels would also be derived from other methods.

Finally, in Chapter 5, we studied end-to-end few-shot learning. As unlabeled data are cheap to collect, the main idea was to design a end-to-end few-shot classifier which was able to learn from this unlabeled data. We created PROTAUGMENT, a novel extension of prototypical networks. This method mimics the mechanism of prototypical networks and applies it on unlabeled episodes, created from unlabeled data and their paraphrases. We tested our model using common data augmentation techniques like back-translation or EDA. While it yielded some improvements, they were marginal at best. We quickly realised the reason was that such augmentation techniques generate sentences which are very close to the original ones. To make things better, we trained our own paraphrase generation model, and added constraints to increase the diversity of generated texts. Those paraphrases, used in our novel framework, yielded significant improvements over the supervised baseline, especially when the amount of training data is very limited. In some cases, it even outclassed the supervised baseline which was given more labeled samples. This unlabeled episode mechanism is not exclusively usable by prototypical networks, as we are just applying their mechanism on this episode. As a future line of research, one could experiment with those unlabeled episodes on other episode-base meta-learning methods. In our experiments using EDA as a paraphrasing method, we did respect the values recommended by the EDA paper for the amount of perturbed tokens. As we concluded that diversity tends to correlate with performances of the downstream classifier, we could spend some more experiments using EDA with higher amounts of perturbed tokens. Additionally, while the paraphrases we generated under constraints are highly diverse and positively contribute to improving the classification model's performances, there is one major drawback to such method: the language. Indeed, our paraphrase generation model was first a BART model, trained on english corpora. Then, we adapted 3 english paraphrase *detection* datasets to the task of paraphrase *generation* to make BART a paraphrase generation model. This makes our model english-only, and experiments on other languages are not possible as is. Finally, a line of research is within the text generation model for diverse paraphrasing. In PROTAUGMENT, we are using a auto-regressive model and

the left-to-right generation makes it hard to set generation constraints while obtaining fluent result with the beam search algorithm. It is possible that this task may be more suitable to non auto-regressive text generation [162], which could better enforce generation constraints, especially by setting must-have tokens. While this falls in the scope of the PhD thesis of an ongoing student at the laboratory, I contributed to such experiments mostly about the meta-learning framework I developed and that can be easily reused in this case.

Overall, this thesis extensively studied few-shot intent classification. Labeled data are costly to obtain, especially in the case where domain experts are required to label such data. As there are not a lot of those experts, labeling data can also take a lot of time. When no unlabeled data are available, we exposed (Chapter 3) a state-of-the-art of the various methods which can be used to address this problem. If unlabeled data are available, then others techniques can be applied. In a human-in-the-loop approach, if methods automatically assign a pseudo-label (Chapter 4) to unlabeled data, it is much more efficient for a human to confirm or inform the pseudo-label than selecting the correct label themselves. As this process can still take a lot of time, end-to-end systems which are able to learn from both labeled and unlabeled data are an option. We showed (Chapter 5) that adding lots of noise to unlabeled samples greatly benefits the downstream classifier, especially in the extreme case where very few samples are available for each training class.

All code used to run experiments showcased in this thesis is available in my own github repository [2], so that future research can build upon those methods.

[2] https://github.com/tdopierre/

Chapter 7

Version Française

7.1 Introduction

7.1.1 Agents conversationnels

Le langage naturel est utilisé chaque jour par des milliards d'êtres humains pour communiquer entre eux. Ces langues évoluent généralement avec le temps, avec l'introduction de nouveaux concepts et de nouvelles façons de formuler ces concepts, ou avec des mots qui disparaissent de la langue commune car ils sont utilisés de moins en moins souvent. Si la communication peut se faire à l'aide de mots, il existe également d'autres moyens de partager de l'information sans recourir aux dictionnaires classiques. Par exemple, les emojis sont des moyens d'exprimer une idée dont l'usage ne cesse de croître [1]. Chaque année, de nouveaux emojis sont introduits, créant de nouvelles façons d'exprimer une idée.

Si les langues de différents pays partagent certains aspects communs, les siècles passent et les langues évoluent toutes différemment. Même à l'intérieur d'un même pays, les dialectes sont différents selon les régions. Des personnes d'âges différents utilisent également des mots différents, car les dialectes ont évolué en se transmettant de génération en génération. Ces fluctuations dans la façon dont les gens s'expriment rendent les langues difficiles à cerner par un ensemble de règles.

Néanmoins, dans notre espace numérique, il existe un besoin de traiter des contenus exprimés dans ces langues multiples et en constante évolution. La façon la

[1] https://blog.emojipedia.org/emoji-trends-that-defined-2020/

plus évidente de répondre à ce besoin est d'utiliser des algorithmes opérés par ordinateurs afin de pouvoir classer, trier, regrouper ou même générer des contenus. Ce domaine de recherche en informatique est connu sous le nom de traitement automatique du langage naturel (TALN, ou NLP en anglais).

Compte tenu du nombre croissant de services en ligne disponibles, les technologies utilisant le TALN sont de plus en plus répandues. Cela va de la recherche de spams dans les courriers électroniques à des tâches plus complexes telles que la génération de romans par ordinateur ou le résumé de flux d'informations. Les technologies de TALN sont non seulement accessibles sur un ordinateur personnel, mais elles sont également à portée de main des individus. Les utilisateurs de téléphones mobiles modernes sont exposés à ces technologies quotidiennement, consciemment ou non selon la technologie. Par exemple, l'autocorrection et la prédiction du mot suivant sur le clavier d'un téléphone portable sont considérées comme une technologie basée sur le TALN, car elles analysent les phrases correctement écrites afin de faciliter la vie de l'utilisateur.

L'un des plus grands défis que le TALN poursuit est de fournir des agents conversationnels complets qui seraient capables de discuter avec les humains afin de les assister dans leurs activités. Cette thèse s'inscrit dans cette ère de recherche comprenant regain d'intérêt pour construire de tels agents grâce aux architectures neuronales [52]. Dans cette thèse, nous nous concentrerons sur l'application du TALN aux agents conversationnels. La course à l'élaboration d'un agent conversationnel "intelligent" a commencé très tôt : en 1964, ELIZA, le premier programme informatique conversationnel a été créé par Joseph Weizenbaum. Ce programme informatique était basé sur un ensemble fixe de règles, qui donnait à l'utilisateur l'illusion d'être compris, l'illusion d'avoir une vraie conversation. Dans le contexte de l'économie numérique actuelle, les entreprises traitent plus de 265 milliards [2] de demandes clients par an, ce qui représente des dépenses pour les entreprises de l'ordre de 1,3 trillion de dollars. L'utilisation de chatbots (une autre façon de désigner les agents conversationnels dans ce manuscrit) pour traiter ces demandes pourrait permettre d'économiser jusqu'à 30% de ce coût[2].

[2]https://www.ibm.com/blogs/watson/2017/10/how-chatbots-reduce-customer-service-costs-by-30-percent/

Du point de vue des applications, les chatbots sont conçus à la fois pour comprendre les requêtes utilisateur et fournir des réponses appropriées. L'une des façons d'aborder le premier point est d'effectuer une classification des intentions, c'est-à-dire d'essayer de comprendre quelle est l'intention exprimée dans la requête de l'utilisateur. Par exemple, le modèle de classification devrait séparer "Je veux commander une pizza" de "Je dois réserver un vol". Pour la partie réponse, l'agent conversationnel peut soit générer une réponse (modèle basé sur la génération), soit extraire une réponse d'un ensemble de réponses possibles. Dans cette thèse, nous nous concentrons sur la tâche de détection d'intention, qui tente de comprendre et de catégoriser les requêtes des utilisateurs en classes d'intentions. Il s'agit d'un cas particulier de la tâche de classification de texte standard, dans lequel les contenus textuels sont générés par les utilisateurs (ce qui implique des problèmes tels que l'hétérogénéité de la longueur, les erreurs grammaticales et de construction, l'argot, ...) et les classes sont généralement comptées par dizaines. A l'inverse, un algorithme de détection de spam fait de la classification binaire. Pour construire un modèle de détection d'intention, il faut donc s'attendre à avoir besoin d'une grande quantité de données d'entraînement, car les approches modernes du langage naturel reposent sur des réseaux neuronaux artificiels profonds [52, 137, 33], qui sont gourmands en données. Compte tenu des fluctuations du langage humain évoquées ci-dessus, de la variété des différentes langues ainsi que de leurs dialectes propres, et du fait que les chatbots évoluent généralement avec le temps au fur et à mesure de l'introduction de nouvelles fonctionnalités, la construction d'un jeu de données d'entraînement robuste n'est pas une tâche simple. Au contraire, étant donné le coût de l'annotation et le fait que de telles annotations peuvent être non pertinentes dans un futur proche en raison de l'évolution du chatbot, ces derniers sont souvent confrontés au cas où peu de données étiquetées sont disponibles pour l'entraînement. Cette thèse étudie donc comment entraîner efficacement un modèle de détection d'intention pour les chatbots avec une quantité limitée de données, ce qui est généralement un régime appelé "apprentissage few-shot" (Few-Shot Learning, en anglais) tel que décrit dans la Section 2.4. Comme l'apprentissage few-shot signifie toujours avoir quelques données d'entraînement (le scénario sans données d'entraînemenr étant appelé apprentissage zero-shot [149] dans la littérature), nous aborderons brièvement dans la section suivante les enjeux et les problèmes de l'annotation des données.

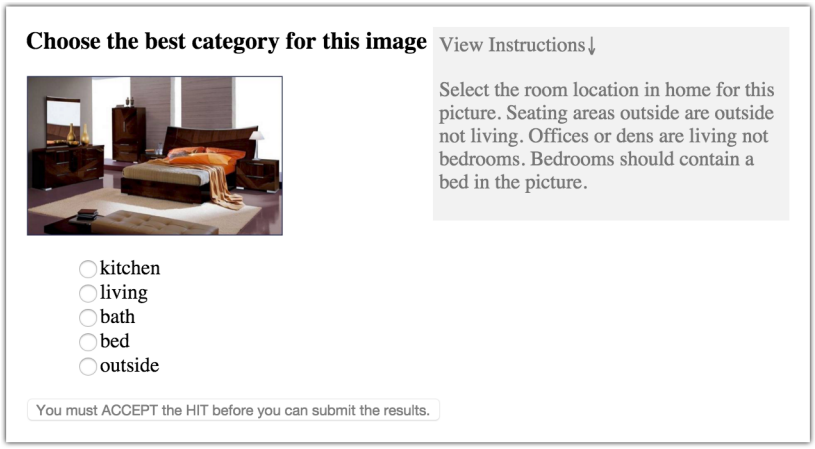


Figure 7.1: Exemple d'une tâche d'annotation d'image sur un site Web de crowd-sourcing.

7.1.2 Annotation des données

L'annotation (parfois appelée étiquetage) de données consiste à apporter une connaissance humaine à une donnée brute. Dans le cadre des chatbots, des échantillons de utilisateur brutes sont annotées avec une ou plusieurs intentions, afin d'apprendre au modèle à classer les demandes des utilisateurs qu'ils recevra à l'avenir. L'augmentation exponentielle du volume de données dans le monde est une aubaine, car elle signifie que de nombreuses données sont disponibles. Dans le contexte des agents conversationnels, l'acquisition de données non étiquetées est assez bon marché, puisqu'il suffit de collecter les historiques des conversations passées entre les utilisateurs et l'agent conversationnel. Malheureusement, ces données ne sont pas annotées et nécessitent un jugement humain pour pouvoir être exploitées comme des données annotées. Pour annoter de telles données, les sites Web de crowd-sourcing sont principalement utilisés [108, 153], où des personnes du monde entier sont payées pour effectuer des tâches d'annotation.

Cependant, une étude récente [3] sur la plateforme Amazon Mechanical Turk [4] montre que le nombre d'annotateurs dans le monde est au mieux stable, au pire en baisse entre 2017 et 2018. Avec le volume croissant de données non étiquetées, ainsi

[3] https://www.cloudresearch.com/resources/blog/how-many-amazon-mturk-workers-are-there/
[4] https://www.mturk.com/

que le nombre croissant d'applications basées sur l'apprentissage automatique nécessitant des données étiquetées, la diminution du nombre d'annotateurs conduit à une impasse. Pire encore, les tâches de traitement automatique des langues devenant de plus en plus complexes, la durée du processus d'annotation augmente, ce qui entraîne une augmentation rapide des coûts. La compréhension automatique en est un exemple [114] : il faut du temps aux annotateurs pour lire et traiter un paragraphe, puis pour sélectionner la réponse à une question donnée sur ce paragraphe. Pour les tâches de classification de texte plus standard, comme nous nous tournons vers des jeux de données de plus en plus difficiles afin d'entraîner des modèles capables de prendre en charge des scénarios plus sophistiqués au fur et à mesure que nous progressons dans la recherche, nous attendons plus de subtilité, donc d'efforts, dans le processus d'annotation. Cette complexité accrue pour les annotateurs humains est significative, et elle est amplifiée par la nécessité d'avoir plusieurs humains qui annotent les mêmes échantillons afin d'obtenir des étiquettes plus fiables mais aussi d'identifier les travailleurs malveillants.

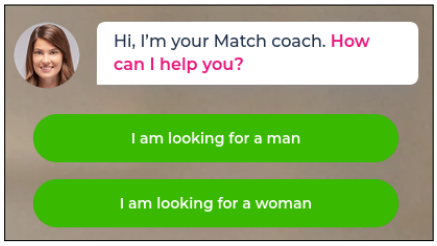
Pour surmonter ce problème, nous recherchons des algorithmes qui ne sont pas très gourmands en données, ou pour le moins, qui sont *suffisamment performants* même s'ils ne disposent pas d'une grande quantité de données étiquetées. Les écarts de performances entre de tels models dépendent de la tâche et du contexte. Dans cette thèse, nous explorerons des pistes afin de lutter contre les délais temps et coûts associés à l'annotation des données en essayant d'explorer des moyens pertinents et efficaces pour limiter le besoin d'annotateurs humains. Il s'agit d'une part de réduire le coût de construction d'un module de détection d'intention pour les chatbots. D'autre part, plus on allége le besoin d'avoir recours à l'humain dans le processus, plus la construction et l'amélioration d'un algorithme d'apprentissage dédié à la détection d'intention est rapide.

Cette thèse s'inscrivant à la fois dans un contexte académique (Laboratoire Hubert Curien UMR CNRS 5516) et dans un contexte d'entreprise (Meetic de Match Group), la section suivante présente quelques idées sur le chatbot industriel sur lequel nos contributions peuvent être instanciées. Bien qu'aucune de nos contributions ne soit spécifique à ce chatbot, cette prochaine section donne un aperçu des questions pratiques qui motivent notre recherche.

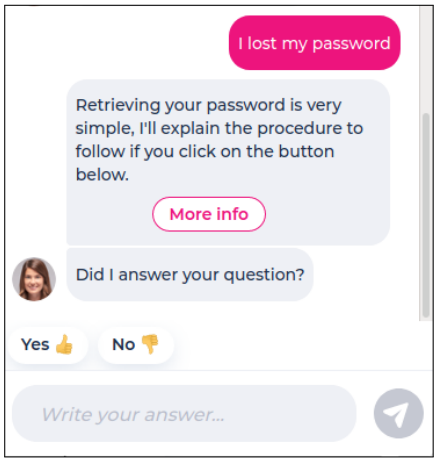
7.1.3 Lara, le chatbot de Meetic

Meetic est un site de rencontres français fondé en 2001. Avec des centaines de milliers d'utilisateurs qui l'utilisent quotidiennement, il est l'un des sites de rencontre les plus populaires en France. Créée en 2016, Lara est l'agent conversationnel de Meetic. Lorsqu'elle interagit avec les utilisateurs sur l'application mobile, elle couvre plusieurs objectifs (Figure 7.2). Dès le début de l'expérience utilisateur, elle aide à créer son profil. Lorsque les utilisateurs rencontrent un problème, par exemple lorsqu'ils perdent leur mot de passe, Lara est là pour les aider à résoudre le problème. Cependant, l'essentiel de son travail consiste à agir comme un coach en matière de rencontres. À ce titre, elle aide les utilisateurs à améliorer la qualité de leur profil (par exemple, en remplissant les champs vides ou en choisissant la meilleure photo), recommande des profils (plus de 70000 profils recommandés par jour) et fournit des conseils personnalisés. Dans l'ensemble, elle reconnaît plus de 300 d'intentions d'utilisateurs couvrant ces différents cas d'utilisation (ceci indique le nombre de classes que nous considérons dans nos algorithmes de détection d'intentions). Ces intentions évoluent en nombre et en portée avec le temps. Au début de ma thèse, il y en avait moins de 30. Au fur et à mesure que les fonctionnalités du site web ont évolué, et le comportement des utilisateurs aussi, nous avons progressivement ajouté de plus en plus d'intentions pour mieux capturer les propos des utilisateurs. Cela a donné lieu à des scénarios dans lesquels nous avions besoin de méthodes de classification des intentions qui s'adaptent rapidement : ces adaptation en continu sont difficilement compatibles avec la reprise d'un processus complet d'annotation des données à chaque fois que des classes sont ajoutées, supprimées ou partiellement fusionnées.

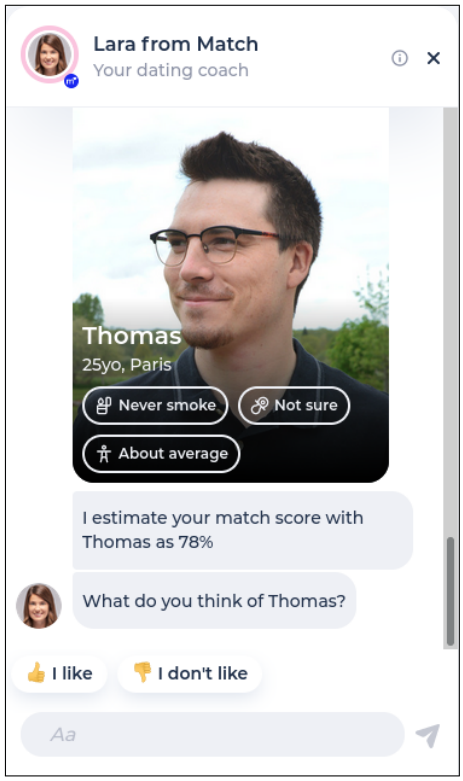
Lara parle également 6 langues, comme le montre la Figure 7.3. Bien que la collecte et l'annotation de données en anglais et en français étaient possibles – mais pénibles – pour nous, faire de même dans toutes les langues est d'un ordre de grandeur. En outre, les données annotées en anglais sont plus fréquentes sur le Web, mais il est beaucoup plus difficile de trouver des jeux de données qui pourraient être utiles dans les autres langues. Bien que l'on puisse soutenir que les systèmes de traduction automatique neuronale [7] peuvent apporter une solution à ce problème, nous démontrerons plus tard (Section 5.4.3) qu'ils ne sont guère une solution pour la détection de l'intention de l'utilisateur étant donné les caractéristiques spécifiques du



(a) Création du profil



(b) Customer care



(c) Recommandation de profil

Figure 7.2: Différents cas d'utilisation de Lara

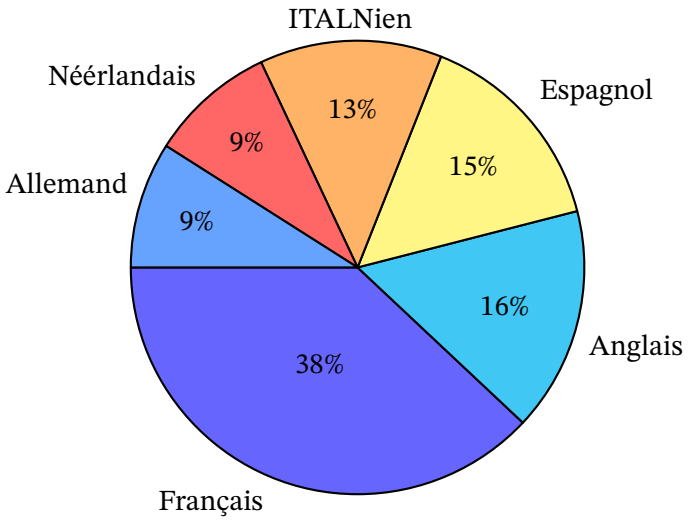


Figure 7.3: Répartition du langage parlé par les utilisateur interagissant avec Lara

texte généré par l'utilisateur. Dans l'ensemble, le multilinguisme accroît le besoin d'une méthode capable de s'adapter rapidement à un nouveau contexte de détection d'intention à partir de quelques données étiquetées.

7.2 Contributions

7.2.1 Sur l'importance des représentations de phrases pour les modèles de type Few-Shot

Cette section récapitule la contribution détaillée dans le chapitre 3, et se base sur la publication suivante

> Thomas Dopierre, Christophe Gravier, Wilfried Logerais. "A Neural Few-Shot Text Classification Reality Check". In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics (EACL2021)*, pages 935–943, 2021.

Dans cette contribution, nous avons tout d'abord identifié le besoin de disposer d'un point de référence équitable et rigoureux pour comparer toutes les méthodes d'apprentissage few-shot. Les méthodes d'encodage de texte utilisées jusqu'à présent étaient basées sur des représentations vectorielles de mots non-contextuelles. Les

transformers [137, 33] ont radicalement changé la façon dont nous travaillons avec le texte, et les méthodes d'apprentissage few-shot n'avaient pas encore été testées en utilisant de tels modèles de langue. Tout d'abord, nous avons montré comment nous pouvions adapter ces modèles de langage pour des textes courts générés par des utilisateurs de manière non supervisée, avant de résoudre la tâche supervisée. Cette étape supplémentaire est peu coûteuse, car elle ne nécessite aucun effort d'étiquetage. De plus, dans le cadre d'un agent conversationnel, la collecte de données non étiquetées est facile, car il suffit d'attendre que des utilisateurs réels interagissent avec l'agent. Grâce à une analyse des représentations vectorielles, nous avons montré que l'adaptation du modèle de langue permet déjà de séparer les classes, ce qui signifie un meilleur point de départ pour la tâche en aval de la classification de texte.

Nous avons ensuite établi une comparaison équitable des différentes méthodes neuronales de bout en bout de classification de textes en "few-shot" découvertes au cours des dernières années. Lorsqu'elles sont toutes équipées d'un encodeur de texte basé sur un transformer, nous avons montré que les réseaux prototypiques [129], qui restent l'une des approches les plus simples et les plus directes, redeviennent l'état de l'art. Nous avons également constaté qu'un modèle de classification traditionnel entraîné sur quelques exemples donne des résultats très compétitifs. En termes de paramètres, les réseaux prototypiques ne sont pas très lourds. Leurs bonnes performances montrent que la majeure partie de l'apprentissage est assurée par les modèles de langue, qui sont capables de représenter le texte en vecteurs d'une manière qui n'était pas possible auparavant. Nous avons également confirmé l'impact de la fonction de distance. Cette différence de performances, déjà observée par les auteurs des réseaux prototypiques, est illustrée dans nos expériences avec toutes les méthodes qui nécessitent une métrique de distance prédéfinie. Globalement, nous avons constaté que la distance euclidienne fonctionne mieux que la distance en cosinus.

Avec cette contribution, nous revisitons non seulement l'état de l'art, mais nous construisons également une configuration complète afin d'implémenter, de tester et de comparer d'autres contributions par rapport aux travaux existants, ce qui sera du plus grand intérêt pratique pour la reproductibilité cette thèse et d'autres chercheurs

de la communauté. La réimplémentation de toutes les méthodes existantes et la préparation de ce cadre d'évaluation du méta-apprentissage ont occupé une partie importante de mes premières années de thèse. Le code source complet (réimplémentation et cadre d'évaluation) est disponible publiquement [5]. A ma connaissance, ce code est déjà utilisé par Raphäel Chevasson, un doctorant de deuxième année du laboratoire travaillant sur une méthode non auto-régressive pour la génération de texte, et Dina El Zein, une étudiante de Master de l'ENS Lyon dans notre équipe qui a travaillé sur l'atténuation du biais de genre pour les cadres de méta-apprentissage. J'espère que cette contribution publique aidera la communauté à s'appuyer sur des expériences comparatives cohérentes, et à favoriser la classification de textes de bout en bout dans un cadre few-shot.

7.2.2 Une méthode de pliage/dépliage pour le pseudo-étiquetage

Cette section récapitule la contribution détaillée dans le chapitre 4, et se base sur la publication suivante

> Thomas Dopierre, Christophe Gravier, Julien Subercaze, Wilfried Logerais. "Few-shot Pseudo-Labeling for Intent Detection". In *Proceedings of the 28th International Conference on Computational Linguistics (COLING2020)*, pages 4993–5003, 2020.

Dans un cadre d'étiquetage, il est difficile pour l'annotateur de sélectionner l'étiquette correcte si le nombre de classes est élevé, et ceci est amplifié si le volume de données à annoter est élevé. Une approche possible consiste donc à faire l'hypothèse d'une étiquette à une donnée non étiquetée étant donné sa proximité dans l'espace de représentation. Ce processus est appelé pseudo-étiquetage [5]. Les pseudo-étiquettes résultantes peuvent alors être utilisées pour l'apprentissage, laissant l'algorithme d'apprentissage gérer l'incertitude associée, ou laissant l'annotateur décider si la pseudo-étiquette donnée est appropriée – il améliore le processus d'annotation dans ce dernier cas. Dans ce contexte, la deuxième contribution que nous présentons est un nouvel algorithme de pseudo-étiquetage en deux étapes pour la détection des intentions. Cette méthode, inspiré du clustering hiérarchique, est d'un intérêt pratique

[5] https://github.com/tdopierre/FewShotText

extrême pour aider les linguistes à fournir plus rapidement de nouveaux modèles de détection d'intention. Notre méthode, étant sans hyper-paramètres, est robuste à diverses situations, même lorsque le nombre de classes est élevé. En écartant les clusters qui sont éloignés des classes connues, elle est capable d'atténuer le bruit qui aurait été introduit dans l'ensemble de données non étiquetées. Par le biais d'une étude d'ablation, nous montrons également que notre méthode est complémentaire aux autres méthodes déjà existantes, de sorte qu'elle contribue de manière significative aux approches agrégées. Nous avons également démontré que si nous continuons à évaluer la détection d'intention en utilisant des ensembles de données très faciles – mais populaires – comme Snips, nous ne résolvons finalement pas la tâche réelle en aval dans la pratique, et nous encourageons donc tous les chercheurs et praticiens de la détection d'intention à l'utiliser avec prudence.

7.2.3 Extension des réseaux prototypiques à l'aide de données non étiquetées et de paraphrases diverses

Cette section récapitule la contribution détaillée dans le chapitre 5, et se base sur la publication suivante

> Thomas Dopierre, Christophe Gravier, Wilfried Logerais. "PROTAUGMENT: Intent Detection Meta-Learning through Unsupervised Diverse Paraphrasing". In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics (ACL2021)*, pages 2454–2466, 2021.

Dans le domaine des agents conversationnels, il est facile et peu coûteux de recueillir un grand nombre de requêtes d'utilisateurs non étiquetés, car les utilisateurs interagissent avec l'agent quotidiennement. Étant donné quelques données étiquetées et des données non étiquetées, nous voulons trouver un moyen de combiner les deux afin d'obtenir un modèle robuste. Le méta-apprentissage est très important pour l'apprentissage automatique car il s'inspire de la façon dont les humains apprennent à partir d'ensembles de mini-tâches et de la façon dont nous généralisons ensuite à des configurations similaires mais différentes. En guise de dernière contribution, nous avons proposé PROTAUGMENT, une architecture de méta-apprentissage

pour le problème de détection d'intention dans les textes courts générés par les utilisateurs. Nous avons d'abord introduit une fonction de coût de cohérence de paraphrase, non supervisée, dans le cadre du réseaux prototypique afin d'améliorer son pouvoir de représentation. Cette nouvelle extension imite la méthode de ce réseau en entraînant le modèle à retrouver la phrase originale sur la base des prototypes calculés à partir des paraphrases. En ce qui concerne la génération de paraphrases, nous avons démontré le manque de diversité des paraphrases issues de la rétro-traduction [40]. Bien que cette technique ait montré d'excellents résultats dans d'autres tâches liées au TAL, nous avons constaté qu'elle n'est pas suffisante lorsqu'elle est utilisée sur des textes courts. En effet, comme le montrent les scores de similarité, la rétro-traduction de textes courts produit généralement la même phrase que l'original, ou une phrase très proche. Lorsque nous avons utilisé des phrases augmentées par la méthode EDA [145], cela n'a pas donné une amélioration très significative par rapport au réseaux prototypique de base. Pour générer des paraphrases plus diversifiées, nous avons étudié l'algorithme de recherche de faisceau diversifié (Diverse Beam Search [138]), qui a été créé pour générer des légendes diversifiées pour les images. En l'utilisant pour la génération de paraphrases, nous avons réalisé qu'il avait été conçu pour renforcer la diversité entre les textes générés, et qu'il ne garantissait pas la diversité entre les textes générés et les phrases originales. Pour compenser ce dernier point, nous introduisons des contraintes dans la génération, ce qui augmente encore la diversité. Notre évaluation approfondie démontre que PROTAUGMENT offre un bond significatif en termes de performances sur les jeux de données les plus récents et les plus difficiles. PROTAUGMENT surpasse largement les réseaux prototypiques, qui s'est avéré être la méthode few-shot la plus compétitive pour les textes courts [37] avec les réseaux prototypiques étendus non supervisés [117], contre les réseaux d'appariement [139], les réseaux de relations [133], et les réseaux d'induction [49], faisant ainsi de PROTAUGMENT le nouvel état de l'art pour cette tâche. Nous fournissons le code source de PROTAUGMENT ainsi que le code pour les évaluations rapportées dans ce chapitre sur un dépôt public [6].

[6] https://github.com/tdopierre/ProtAugment

7.3 Conclusion

Dans cette thèse, nous avons étudié différentes manières de répondre au problème de la classification d'intentions de type "few-shot". D'abord, nous avons présenté le contexte de la thèse, ainsi que les raisons pour lesquelles l'étiquetage des jeux de données est à la fois coûteux en temps et en énergie. Un fossé émerge entre les grands modèles pré-entraînés qui sont avides de données, et le fait que de tels ensembles de données étiquetées ne sont pas faciles à rassembler. Il faut donc des algorithmes capables d'apprendre à partir de petits ensembles de données, parfois aussi petits que des classes avec un seul échantillon. Nous avons ensuite décrit de manière extensive les connaissances techniques de base nécessaires pour situer cette thèse. Les modèles appliqués au texte ont connu un très grand changement au cours des dernières années, du sac de mots, aux représentations continues de mots, aux transformers. Avec l'utilisation croissante des réseaux de neurones, les modèles de classification ont également beaucoup changé, augmentant généralement en complexité et atteignant de meilleures performances. Nous avons aussi complètement défini le cadre du méta-apprentissage, dans lequel l'apprentissage se fait par épisodes. Dans ce cadre, les modèles sont formés pour s'adapter rapidement à de nouvelles tâches au moment du test, ce qui correspond aux scénarios réels, où les cas d'utilisation évoluent régulièrement.

Bien que les modèles d'apprentissage de type few-shot aient fait des progrès ces dernières années, la comparaison entre eux est injuste, car ils n'utilisaient pas le même codeur de texte au moment de leur introduction. Dans un premier temps, afin de surmonter ce problème de comparaison injuste, nous avons étudié les performances de ces modèles d'apprentissage few-shot lorsqu'ils sont équipés d'un encodeur de texte unique et récent basé sur un transformer. Compte tenu de cet encodeur, nous avons montré que les réseaux prototypiques revendiquent la place de l'état de l'art. Cette constatation est vraie à la fois sur les jeux de données ARSC et sur quatre jeux de données publics de détection d'intention. Bien que les méthodes d'apprentissage few-shot aient été introduites en raison de l'incapacité des modèles de classification standard à être performants dans les régimes de données faibles, nous avons constaté que ces modèles traditionnels ne se comportaient pas si mal lorsqu'ils étaient utilisés avec des transformers. Enfin, nous avons souligné l'importance

de la métrique de distance, car nos expériences avec la distance euclidienne celle du cosinus ont donné des performances significativement meilleures pour la première. Ce résultat ne doit pas être considéré comme acquis : à l'avenir, si une architecture complètement nouvelle est utilisée pour représenter des textes, elle pourrait changer complètement la structure de l'espace des représentations. Compte tenu de cette nouvelle structure, une autre métrique de distance pourrait être plus adaptée. Si, pour nous, la distance euclidienne a donné les meilleurs résultats, elle n'est pas parfaite et tous les ensembles de données ne sont pas structurés de la même manière. Dans une future ligne de recherche, on pourrait trouver un moyen d'étendre cette distance euclidienne au domaine de l'apprentissage de métrique. Alors que dans nos expériences, nous n'avons pas trouvé d'approches d'apprentissage de métrique basées sur des modules de relations particulièrement efficaces, on pourrait expérimenter une distance comme la distance de Mahalanobis [7], qui peut être initialisée pour être équivalente à la distance euclidienne afin de favoriser l'optimisation lors de l'apprentissage.

Comme deuxième contribution, nous nous sommes concentrés sur les techniques de pseudo-étiquetage. Lorsqu'un grand ensemble de données non étiquetées est disponible, on peut utiliser cet ensemble de données pour obtenir des pseudo-étiquettes, et utiliser les données étiquetées et pseudo-étiquetées pour entraîner un modèle de classification. Il s'agit d'un processus en deux étapes : la génération de pseudo-étiquettes, puis l'entraînement du modèle. Nous avons introduit une nouvelle méthode pour générer des pseudo-étiquettes, qui est basée sur le mécanisme de regroupement hiérarchique. Notre méthode est capable de récupérer des pseudo-étiquettes de haute qualité, en omettant volontairement les points de données qui sont difficiles à pseudo-étiqueter, et qui pourraient donc correspondre à du bruit. De plus, elle ne dépend d'aucun hyperparamètre, ce qui la rend robuste à différentes configurations. Enfin, notre méthode est très complémentaire des autres lorsqu'il s'agit d'agréger des pseudo-étiquettes provenant de différentes techniques. Nos expériences n'ont considéré qu'un seul tour de pseudo-étiquetage, puis ont entraîné un modèle de classification d'intention par-dessus. Néanmoins, nous aurions pu expérimenter notre algorithme en exécutant plus d'itérations de pseudo-étiquetage comme dans le self-training [99], le co-training [16] ou le tri-training [167] (voir Section 2.4.3.1).

[7] https://en.wikipedia.org/wiki/Mahalanobis_distance

Notez néanmoins que notre méthode doit être adaptée pour le self-training. En effet, après avoir exécuté une itération de pseudo-étiquetage, l'exécution de l'étape dite du "pliage" donnerait la même structure arborescente, puisque cette étape ne prend pas en compte les étiquettes, et les représentations des phrases seraient les mêmes. De plus, notre étape de "dépliage" donnerait également la même décomposition et, au final, notre ensemble de pseudo-étiquettes ne changerait pas. Pour le co-training ou le tri-training, ce n'est pas un problème, car les pseudo-étiquettes sont également obtenues via d'autres méthodes.

Enfin, dans le chapitre 5, nous avons étudié l'apprentissage few-shot de bout en bout. Les données non étiquetées étant peu coûteuses à collecter, l'idée principale était de concevoir un modèle de classification few-shot de bout en bout capable d'apprendre à partir de ces données non étiquetées. Nous avons créé PROTAUGMENT, une nouvelle extension des réseaux prototypiques. Cette méthode imite le mécanisme des réseaux prototypiques et l'applique sur des épisodes non étiquetés, créés à partir de données non étiquetées et de leurs paraphrases. Nous avons testé notre modèle en utilisant des techniques courantes d'augmentation des données comme la rétro-traduction ou EDA [145]. Bien que nous ayons obtenu quelques améliorations, celles-ci étaient au mieux marginales. Nous avons rapidement compris que la raison en était que ces techniques d'augmentation génèrent des phrases très proches des phrases originales. Pour améliorer les performances, nous avons formé notre propre modèle de génération de paraphrases, et ajouté des contraintes pour augmenter la diversité des textes générés. Ces paraphrases, utilisées dans notre nouveau modèle, ont donné des améliorations significatives par rapport à la ligne de base supervisée, en particulier lorsque la quantité de données d'entraînement est très limitée. Dans certains cas, elles ont même surclassé la ligne de base supervisée qui a reçu plus d'échantillons étiquetés. Ce mécanisme d'épisode non étiqueté n'est pas exclusivement utilisable par les réseaux prototypiques, car nous appliquons simplement leur mécanisme à cet épisode. Dans une future ligne de recherche, on pourrait expérimenter avec ces épisodes non étiquetés sur d'autres méthodes de méta-apprentissage basées sur des épisodes. Dans nos expériences utilisant EDA comme méthode de paraphrase, nous avons respecté les valeurs recommandées par l'article d'EDA pour la quantité de mots soumis à des pertubations. Comme nous avons conclu que la diversité a tendance à être corrélée avec les performances du classificateur

en aval, nous pourrions faire d'autres expériences en utilisant EDA avec des quantités plus élevées de mots perturbés. De plus, si les paraphrases que nous avons générées sous contraintes sont très diversifiées et contribuent positivement à l'amélioration des performances du modèle de classification, il existe un inconvénient majeur à cette méthode : la langue. En effet, notre modèle de génération de paraphrases a d'abord été un modèle BART [82], entraîné sur des corpus anglais. Ensuite, nous avons adapté trois jeux de données de paraphrases anglaises à la tâche de génération de paraphrases pour faire de BART un modèle de génération de paraphrases. Cela rend notre modèle uniquement anglais, et les expériences sur d'autres langues ne sont pas possibles en l'état. Enfin, une ligne de recherche se situe dans le modèle de génération de texte pour la paraphrase diverse. Dans PROTAUGMENT, nous utilisons un modèle auto-régressif et la génération de gauche à droite rend difficile l'établissement de contraintes de génération tout en obtenant un résultat fluide avec l'algorithme de recherche de faisceau. Il est possible que cette tâche soit plus adaptée à la génération de texte non auto-régressive [162], qui pourrait mieux appliquer les contraintes de génération, notamment en définissant des mots obligatoires. Bien que cela relève du champ d'application de la thèse d'un étudiant en cours au laboratoire, j'ai contribué à ces expériences principalement grâce au cadre de méta-apprentissage que j'ai développé et qui peut être facilement réutilisé dans ce cas.

Dans l'ensemble, cette thèse a étudié de manière approfondie la classification d'intentions de type "few-shot". Les données étiquetées sont coûteuses à obtenir, en particulier dans le cas où des experts du domaine sont nécessaires pour étiqueter ces données. Comme il n'y a pas beaucoup de ces experts, l'étiquetage des données peut également prendre beaucoup de temps. Lorsque aucune donnée non étiquetée n'est disponible, nous avons exposé (Chapitre 3) un état de l'art des différentes méthodes qui peuvent être utilisées pour résoudre ce problème. Si des données non étiquetées sont disponibles, d'autres techniques peuvent alors être appliquées. Dans une approche "human-in-the-loop", si les méthodes attribuent automatiquement un pseudo-étiquette (Chapitre 4) aux données non étiquetées, il est beaucoup plus efficace pour un humain de confirmer ou d'infirmer la pseudo-étiquette que de sélectionner lui-même l'étiquette correcte. Comme ce processus peut encore prendre beaucoup de temps, les systèmes de bout en bout qui sont capables d'apprendre à partir de données étiquetées et non étiquetées sont une option. Nous avons montré

(Chapitre 5) que l'ajout de beaucoup de bruit aux échantillons non étiquetés profite grandement au classificateur en aval, surtout dans le cas extrême où très peu d'échantillons sont disponibles pour chaque classe d'apprentissage.

Tout le code utilisé pour exécuter les expériences présentées dans cette thèse est disponible sur mon compte github [8], afin que les recherches futures puissent s'appuyer sur ces méthodes.

[8] https://github.com/tdopierre/

Acknowledgements

This thesis was made possible by a collaboration between Laboratoire Hubert Curien and Meetic, in the context of a CIFRE convention. As such, I would like to thank the ANRT for making such collaboration possible. Working both on academic research and industry projects was the best of both worlds for me.

I cannot thank enough Christophe Gravier for his mentoring over these $3\frac{1}{2}$ years. Christophe knows how to make himself available, and he is always ready to listen. On both professional and non-professional matters, he is always keen to answer my questions or discuss any subject. Not only does he points out when something is not right, but he also very well knows how to cheer when things are going well.

On the company side, I would like to express my deepest gratitude to Wilfried Logerais, who supervised – and still does! – me during these years. Although not being related to academic research directly, Wilfried knows what needs to be done to keep a publication agenda on track. While it might be hard to find the right balance between time spent doing research and focusing on company projects, it went out smoothly as Wilfried always understood my unstable needs to spend time on either part.

At Meetic, I would like to thank each member of Analytics. While we did not work on the same subjects, they were always curious about what I was doing, and always asked relevant questions. On the personal side, they always know how to party, and their good daily mood was a big source of motivation for me during the hardest periods of the thesis.

Many thanks to all members – past and present – of PFT, my second family at Meetic. They were the first to trust me, and I am grateful for that. During all these years, they were also the ones pushing me forward, always coming up with novel ideas, and never backing down before adversity.

I would like to thank Pierre Maret from the Laboratory for his relevant advice when I was struggling to get my first paper published.

On a personal note, I would like to thank my family for being here during all these years, especially Valentine for her daily support under all circumstances.

Bibliography

[1] Jacob Levy Abitbol, M. Karsai, Jean-Philippe Magu é, Jean-Pierre Chevrot, and E. Fleury. Socioeconomic dependencies of linguistic patterns in twitter: a multivariate analysis. *Proceedings of the 2018 World Wide Web Conference*, 2018.

[2] Apoorv Agarwal, Boyi Xie, Ilia Vovsha, Owen Rambow, and Rebecca J. Passonneau. Sentiment analysis of twitter data. 2011.

[3] S. Agatonovic-Kustrin and R. Beresford. Basic concepts of artificial neural network (ann) modeling and its application in pharmaceutical research. *Journal of pharmaceutical and biomedical analysis*, 22 5:717–27, 2000.

[4] H. Al-Mubaid and Syed A. Umair. A new text categorization technique using distributional clustering and learning logic. *IEEE Transactions on Knowledge and Data Engineering*, 18:1156–1165, 2006.

[5] Eric Arazo, Diego Ortego, P. Albert, N. O'Connor, and Kevin McGuinness. Pseudo-labeling and confirmation bias in deep semi-supervised learning. *2020 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8, 2020.

[6] V. V. Asch and W. Daelemans. Predicting the effectiveness of self-training: Application to sentiment classification. *ArXiv*, abs/1601.03288, 2016.

[7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2015.

[8] Timothy Baldwin, Trevor Cohn, and Yitong Li. Robust training under linguistic adversity. In *EACL*, 2017.

[9] Rachel Bawden, Biao Zhang, Lisa Yankovskaya, Andre T ä ttar, and Matt Post. A study in improving BLEU reference coverage with diverse automatic paraphrasing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pages 918–932, Online, November 2020. Association for Computational Linguistics.

[10] Markus Bayer, Marc-Andr é Kaufhold, and Christian Reuter. A survey on data augmentation for text classification. 2021.

[11] Yonatan Belinkov and Yonatan Bisk. Synthetic and natural noise both break neural machine translation. *ArXiv*, abs/1711.02173, 2018.

[12] Yoshua Bengio, J. Louradour, Ronan Collobert, and J. Weston. Curriculum learning. In *ICML '09*, 2009.

[13] Michele Bevilacqua, Marco Maru, and Roberto Navigli. Generationary or "how we went beyond word sense inventories and learned to gloss". In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7207–7221, Online, November 2020. Association for Computational Linguistics.

[14] Johan Bissmark and Oscar Wärnling. The sparse data problem within classification algorithms : The effect of sparse data on the naïve bayes algorithm. 2017.

[15] John Blitzer, Mark Dredze, and Fernando Pereira. Biographies, bollywood, boom-boxes and blenders: Domain adaptation for sentiment classification. In *Proceedings of the 45th annual meeting of the association of computational linguistics*, pages 440–447, 2007.

[16] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proc. of the Eleventh Annual Conference on Computational Learning Theory*, COLT' 98, pages 92–100, New York, NY, USA, 1998. ACM.

[17] Piotr Bojanowski, Edouard Grave, Armand Joulin, and Tomas Mikolov. Enriching word vectors with subword information. *TACL*, 5:135–146, 2017.

[18] Lia Bozarth and Ceren Budak. Toward a better performance evaluation framework for fake news classification. In *ICWSM*, 2020.

[19] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*, 2020.

[20] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *ArXiv*, abs/2005.12872, 2020.

[21] Iñigo Casanueva, Tadas Temčinas, Daniela Gerz, Matthew Henderson, and Ivan Vulić. Efficient intent detection with dual sentence encoders. In *Proceedings of the 2nd Workshop on Natural Language Processing for Conversational AI*, pages 38–45, Online, July 2020. Association for Computational Linguistics.

[22] Olivier Chapelle, Bernhard Scholkopf, and Alexander Zien. Semi-supervised learning. *IEEE trans. on Neural Networks*, 20(3):542–542, 2009.

[23] Nontawat Charoenphakdee, Jongyeong Lee, Yiping Jin, Dittaya Wanvarie, and Masashi Sugiyama. Learning only from relevant keywords and unlabeled documents. 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)*. Association for Computational Linguistics, November 2019.

[24] H. Chen, Zijia Lin, Guiguang Ding, Jian-Guang Lou, Yusen Zhang, and B ö rje F. Karlsson. Grn: Gated relation network to enhance convolutional neural network for named entity recognition. *ArXiv*, abs/1907.05611, 2019.

[25] Qian Chen, Zhu Zhuo, and Wen Wang. Bert for joint intent classification and slot filling. *ArXiv*, abs/1902.10909, 2019.

[26] Wei-Yu Chen, Yen-Cheng Liu, Zsolt Kira, Yu-Chiang Wang, and Jia-Bin Huang. A closer look at few-shot classification. In *International Conference on Learning Representations*, 2019.

[27] Kyunghyun Cho. Noisy parallel approximate decoding for conditional recurrent language model. *arXiv preprint arXiv:1605.03835*, 2016.

[28] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. Electra: Pre-training text encoders as discriminators rather than generators. *ArXiv*, abs/2003.10555, 2020.

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

[30] Alexis Conneau, Douwe Kiela, Holger Schwenk, Loï c Barrault, and Antoine Bordes. Supervised learning of universal sentence representations from natural language inference data. In *Proc. of 25th EMNLP*, pages 670–680, Copenhagen, Denmark, 2017.

[31] Alice Coucke, Alaa Saade, Adrien Ball, and Bluche et al. Snips voice platform: an embedded spoken language understanding system for private-by-design voice interfaces. *arXiv preprint arXiv:1805.10190*, 2018.

[32] Roy De Maesschalck, Delphine Jouan-Rimbaud, and D é sir é L Massart. The mahalanobis distance. *Chemometrics and intelligent laboratory systems*, 50(1):1–18, 2000.

[33] J. Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT*, 2019.

[34] Rahul Dey and Fathi M Salem. Gate-variants of gated recurrent unit (gru) neural networks. In *2017 IEEE 60th international midwest symposium on circuits and systems (MWSCAS)*, pages 1597–1600. IEEE, 2017.

[35] Linhao Dong, S. Xu, and Bo Xu. Speech-transformer: A no-recurrence sequence-to-sequence model for speech recognition. *2018 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5884–5888, 2018.

[36] Thomas Dopierre, C. Gravier, and Wilfried Logerais. Protaugment: Unsupervised diverse short-texts paraphrasing for intent detection meta-learning. *ArXiv*, abs/2105.12995, 2021.

[37] Thomas Dopierre, Christophe Gravier, and Thomas Logerais. A neural few-shot text classification reality check. In *Proc. of EACL 2021*, April 2021.

[38] Thomas Dopierre, Christophe Gravier, Julien Subercaze, and Wilfried Logerais. Few-shot pseudo-labeling for intent detection. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 4993–5003, Barcelona, Spain (Online), December 2020. International Committee on Computational Linguistics.

[39] A. Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, M. Dehghani, Matthias Minderer, G. Heigold, S. Gelly, Jakob Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *ArXiv*, abs/2010.11929, 2020.

[40] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 489–500, Brussels, Belgium, October-November 2018. Association for Computational Linguistics.

[41] Hady ElSahar and Matthias Gall é. To annotate or not? predicting performance drop under domain shift. In *EMNLP/IJCNLP*, 2019.

[42] A. R. Fabbri, Simeng Han, Haoyuan Li, Haoran Li, Marjan Ghazvininejad, Shafiq R. Joty, Dragomir Radev, and Yashar Mehdad. Improving zero and few-shot abstractive summarization with intermediate fine-tuning and data augmentation. In *NAACL*, 2021.

[43] Steven Y. Feng, Varun Gangal, Dongyeop Kang, T. Mitamura, and E. Hovy. Genaug: Data augmentation for finetuning text generators. *ArXiv*, abs/2010.01794, 2020.

[44] Steven Y. Feng, Varun Gangal, Jason Wei, Sarath Chandar, Soroush Vosoughi, T. Mitamura, and E. Hovy. A survey of data augmentation approaches for nlp. *ArXiv*, abs/2105.03075, 2021.

[45] Chelsea Finn, Pieter Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. *CoRR*, abs/1703.03400, 2017.

[46] Jianfeng Gao, Michel Galley, and Lihong Li. Neural approaches to conversational AI. *CoRR*, abs/1809.08267, 2018.

[47] Kuiliang Gao, W. Guo, Xuchu Yu, Bing Liu, Anzhu Yu, and Xiangpo Wei. Deep induction network for small samples classification of hyperspectral images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 13:3462–3477, 2020.

[48] Tianyu Gao, Xu Han, Zhiyuan Liu, and Maosong Sun. Hybrid attention-based prototypical networks for noisy few-shot relation classification. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6407–6414, 2019.

[49] Ruiying Geng, Binhua Li, Yongbin Li, Xiaodan Zhu, Ping Jian, and Jian Sun. Induction networks for few-shot text classification. *arXiv preprint arXiv:1902.10482*, 2019.

[50] Spyros Gidaris and N. Komodakis. Dynamic few-shot visual learning without forgetting. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 4367–4375, 2018.

[51] Chih-Wen Goo, Guang Gao, and et al. Hsu. Slot-gated modeling for joint slot filling and intent prediction. In *Proc. of 14th NAACL, Volume 2*, pages 753–757, 2018.

[52] I. Goodfellow, Yoshua Bengio, and Aaron C. Courville. Deep learning. *Nature*, 521:436–444, 2015.

[53] Tanya Goyal and Greg Durrett. Neural syntactic preordering for controlled paraphrase generation. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 238–252, Online, July 2020. Association for Computational Linguistics.

[54] N. Gruber and Alfred Jockisch. Are gru cells more specific and lstm cells more

sensitive in motive classification of text? *Frontiers in Artificial Intelligence*, 3, 2020.

[55] E. Haihong, Peiqing Niu, Zhongfu Chen, and Meina Song. A novel bi-directional interrelated model for joint intent detection and slot filling. *ArXiv*, abs/1907.00390, 2019.

[56] U. Hasson, Samuel A. Nastase, and Ariel Goldstein. Direct-fit to nature: an evolutionary perspective on biological (and artificial) neural networks. *bioRxiv*, 2019.

[57] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Computation*, 9:1735–1780, 1997.

[58] Jeremy Howard and Sebastian Ruder. Universal language model fine-tuning for text classification. In *ACL*, 2018.

[59] Sheng Hu, Yuqing Ma, Xianglong Liu, Yanlu Wei, and Shihao Bai. Hierarchical rule induction network for abstract visual reasoning. *ArXiv*, abs/2002.06838, 2020.

[60] Binxuan Huang and Kathleen M. Carley. A hierarchical location prediction neural network for twitter user geolocation. *ArXiv*, abs/1910.12941, 2019.

[61] Daphne Ippolito, Reno Kriz, João Sedoc, Maria Kustikova, and Chris Callison-Burch. Comparison of diverse decoding methods from conditional language models. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 3752–3762, Florence, Italy, July 2019. Association for Computational Linguistics.

[62] Shelan S. Jeawak, Luis Espinosa Anke, and S. Schockaert. Cardiff university at semeval-2020 task 6: Fine-tuning bert for domain-specific definition classification. In *SEMEVAL*, 2020.

[63] Xiaoqi Jiao, Y. Yin, Lifeng Shang, Xin Jiang, X. Chen, Linlin Li, F. Wang, and Qun Liu. Tinybert: Distilling bert for natural language understanding. *ArXiv*, abs/1909.10351, 2020.

[64] T. Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. In *ICML*, 1997.

[65] Karen Sparcks Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 1972.

[66] Armand Joulin, Edouard Grave, Piotr Bojanowski, and Tomas Mikolov. Bag of tricks for efficient text classification. In *Proc. of 15th EACL, Volume 2*, pages 427–431. Association for Computational Linguistics, April 2017.

[67] Yoon Kim. Convolutional neural networks for sentence classification. In *EMNLP*, 2014.

[68] Ryan Kiros, Y. Zhu, R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler. Skip-thought vectors. In *NIPS*, 2015.

[69] O. Kolomiyets, Steven Bethard, and Marie-Francine Moens. Model-portability experiments for textual temporal analysis. In *ACL*, 2011.

[70] M. Kuhn and Kjell Johnson. Applied predictive modeling. 2013.

[71] Maciej Kula. Metadata embeddings for user and item cold-start recommendations. *ArXiv*, abs/1507.08439, 2015.

[72] Ilia Kulikov, Alexander Miller, Kyunghyun Cho, and Jason Weston. Importance of search and evaluation strategies in neural dialogue modeling. In *Proceedings of the 12th International Conference on Natural Language Generation*, pages 76–87, Tokyo, Japan, October–November 2019. Association for Computational Linguistics.

[73] Varun Kumar, Hadrien Glaude, Cyprien de Lichy, and William Campbell. A closer look at feature space data augmentation for few-shot intent classification. In *DeepLo@EMNLP-IJCNLP*, 2019.

[74] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *ArXiv*, abs/1909.11942, 2020.

[75] K. Lang. Newsweeder: Learning to filter netnews. In *ICML*, 1995.

[76] Stefan Larson, Anish Mahendran, Joseph J Peper, Christopher Clarke, Andrew Lee, Parker Hill, Jonathan K Kummerfeld, Kevin Leach, Michael A Laurenzano, Lingjia Tang, et al. An evaluation dataset for intent classification and out-of-scope prediction. *arXiv preprint arXiv:1909.02027*, 2019.

[77] Hang Le, Lo ï c Vial, Jibril Frej, Vincent Segonne, Maximin Coavoux, B. Lecouteux, A. Allauzen, Benoit Crabb'e, L. Besacier, and D. Schwab. Flaubert: Unsupervised language model pre-training for french. *ArXiv*, abs/1912.05372, 2020.

[78] Quoc V. Le and Tomas Mikolov. Distributed representations of sentences and documents. *ArXiv*, abs/1405.4053, 2014.

[79] Y. LeCun and Yoshua Bengio. Convolutional networks for images, speech, and time series. 1998.

[80] Ji Young Lee and Franck Dernoncourt. Sequential short-text classification with recurrent and convolutional neural networks. *CoRR*, abs/1603.03827, 2016.

[81] D. Lewis, Yiming Yang, T. Rose, and Fan Li. Rcv1: A new benchmark collection for text categorization research. *J. Mach. Learn. Res.*, 5:361–397, 2004.

[82] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 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, Online, July 2020. Association for Computational Linguistics.

[83] Yu Li, Xiao Li, Yating Yang, and Rui Dong. A diverse data augmentation strategy for low-resource neural machine translation. *Inf.*, 11:255, 2020.

[84] Bin Liu, Zhirong Wu, Han Hu, and Stephen Lin. Deep metric transfer for label propagation with limited annotated data. *arXiv preprint arXiv:1812.08781*, 2018.

[85] Bing Liu and I. Lane. Attention-based recurrent neural network models for joint intent detection and slot filling. In *INTERSPEECH*, 2016.

[86] Xianggen Liu, Lili Mou, Fandong Meng, Hao Zhou, Jie Zhou, and Sen Song. Unsupervised paraphrasing by simulated annealing. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 302–312, Online, July 2020. Association for Computational Linguistics.

[87] Xingkun Liu, Arash Eshghi, Pawel Swietojanski, and Verena Rieser. Benchmarking natural language understanding services for building conversational agents. *arXiv preprint arXiv:1903.05566*, 2019.

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

[89] S. Longpre, Y. Lu, Zhucheng Tu, and Christopher DuBois. An exploration of data augmentation and sampling techniques for domain-agnostic question answering. In *MRQA@EMNLP*, 2019.

[90] Liang Lu, Changliang Liu, Jinyu Li, and Yifan Gong. Exploring transformers for large-scale speech recognition. In *INTERSPEECH*, 2020.

[91] L. V. D. Maaten and Geoffrey E. Hinton. Visualizing data using t-sne. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

[92] James H Martin and Daniel Jurafsky. *Speech and language processing: An introduction to natural language processing, computational linguistics, and speech recognition, Chapters 25-26*. Pearson/Prentice Hall Upper Saddle River, 2009.

[93] Louis Martin, Benjamin Muller, Pedro Javier Ortiz Su á rez, Yoann Dupont, L. Romary, Eric Villemonte de la Clergerie, Djam é Seddah, and Beno î t Sagot. Camembert: a tasty french language model. *ArXiv*, abs/1911.03894, 2020.

[94] Justin Martineau and Timothy W. Finin. Delta tfidf: An improved feature space for sentiment analysis. In *ICWSM*, 2009.

[95] Bryan McCann, James Bradbury, Caiming Xiong, and R. Socher. Learned in translation: Contextualized word vectors. In *NIPS*, 2017.

[96] David McClosky, Eugene Charniak, and Mark Johnson. Effective self-training for parsing. In *HLT-NAACL*, 2006.

[97] Shikib Mehri, Mihail Eric, and Dilek Hakkani-Tur. Dialoglue: A natural language understanding benchmark for task-oriented dialogue, 2020.

[98] Thomas Mensink, J. Verbeek, F. Perronnin, and G. Csurka. Metric learning for large scale image classification: Generalizing to new classes at near-zero cost. In *ECCV*, 2012.

[99] Rada Mihalcea. Co-training and self-training for word sense disambiguation. In *Proc. of 8th CoNLL-2004 (HLT-NAACL 2004)*, 2004.

[100] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[101] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems 26*, pages 3111–3119. Curran Associates, Inc., 2013.

[102] G. Miller, R. Beckwith, C. Fellbaum, Derek Gross, and K. Miller. Introduction to wordnet: An on-line lexical database. *International Journal of Lexicography*, 3:235–244, 1990.

[103] F. Murtagh. Multilayer perceptrons for classification and regression. *Neurocomputing*, 2:183–197, 1990.

[104] Chikashi Nobata, Joel R. Tetreault, A. Thomas, Yashar Mehdad, and Yi Chang. Abusive language detection in online user content. *Proceedings of the 25th International Conference on World Wide Web*, 2016.

[105] Anders S ø gaard. Simple semi-supervised training of part-of-speech taggers. In *ACL*, 2010.

[106] Yingwei Pan, Ting Yao, Yehao Li, Yu Wang, C. Ngo, and Tao Mei. Transferrable prototypical networks for unsupervised domain adaptation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 2234–2242, 2019.

[107] Bo Pang and Lillian Lee. Opinion mining and sentiment analysis. *Found. Trends Inf. Retr.*, 2:1–135, 2007.

[108] Gabriele Paolacci, Jesse J Chandler, and Panagiotis G. Ipeirotis. Running experiments on amazon mechanical turk. *Behavioral & Experimental Economics eJournal*, 2010.

[109] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Lukasz Kaiser, Noam M. Shazeer, Alexander Ku, and Dustin Tran. Image transformer. *ArXiv*, abs/1802.05751, 2018.

[110] Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. In *NAACL-HLT*, 2018.

[111] Hang Qi, M. Brown, and D. Lowe. Low-shot learning with imprinted weights. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 5822–5830, 2018.

[112] Libo Qin, Wanxiang Che, Yangming Li, Minheng Ni, and T. Liu. Dcr-net: A deep co-interactive relation network for joint dialog act recognition and sentiment classification. *ArXiv*, abs/2008.06914, 2020.

[113] Alec Radford, Luke Metz, and Soumith Chintala. Unsupervised representation learning with deep convolutional generative adversarial networks. *CoRR*, abs/1511.06434, 2016.

[114] Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. Squad: 100,000+ questions for machine comprehension of text. In *EMNLP*, 2016.

[115] Alan Ramponi and Barbara Plank. Neural unsupervised domain adaptation in nlp—a survey. *ArXiv*, abs/2006.00632, 2020.

[116] Gary Ren, Xiaochuan Ni, Manish Malik, and Qifa Ke. Conversational query understanding using sequence to sequence modeling. In *Proc. of 27th TheWebConf*, pages 1715–1724, 2018.

[117] Mengye Ren, Eleni Triantafillou, Sachin Ravi, Jake Snell, Kevin Swersky, Joshua B Tenenbaum, Hugo Larochelle, and Richard S Zemel. Meta-learning for semi-supervised few-shot classification. *arXiv preprint arXiv:1803.00676*, 2018.

[118] Michele Resta, Daniele Arioli, Alessandro Fagnani, and Giuseppe Attardi. Transformer models for question answering at bioasq 2019. In *PKDD/ECML Workshops*, 2019.

[119] Anna Rogers, Olga Kovaleva, and Anna Rumshisky. A primer in bertology: What we know about how bert works. *Transactions of the Association for Computational Linguistics*, 8:842–866, 2020.

[120] Sebastian Ruder and Barbara Plank. Strong baselines for neural semi-supervised learning under domain shift. *ArXiv*, abs/1804.09530, 2018.

[121] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. Dynamic routing between capsules. In *Advances in neural information processing systems*, pages 3856–3866, 2017.

[122] K. Saito, Y. Ushiku, and T. Harada. Asymmetric tri-training for unsupervised domain adaptation. In *ICML*, 2017.

[123] M. Schuster and Kaisuke Nakajima. Japanese and korean voice search. *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 5149–5152, 2012.

[124] Rico Sennrich, Barry Haddow, and Alexandra Birch. Improving neural machine translation models with monolingual data. *arXiv preprint arXiv:1511.06709*, 2015.

[125] Hamed Shahbazi, Xiaoli Z. Fern, Reza Ghaeini, Rasha Obeidat, and Prasad Tadepalli. Entity-aware elmo: Learning contextual entity representation for entity disambiguation. *ArXiv*, abs/1908.05762, 2019.

[126] Lakshay Sharma, Laura Graesser, Nikita Nangia, and Utku Evci. Natural language understanding with the quora question pairs dataset, 2019.

[127] Chen Shi, Qi Chen, Lei Sha, Sujian Li, Xu Sun, Houfeng Wang, and Lintao Zhang. Auto-dialabel: Labeling dialogue data with unsupervised learning. In *Proc. of 26th EMNLP*, pages 684–689, 2018.

[128] Connor Shorten and Taghi M. Khoshgoftaar. A survey on image data augmentation for deep learning. *Journal of Big Data*, 6:1–48, 2019.

[129] Jake Snell, Kevin Swersky, and Richard Zemel. Prototypical networks for few-shot learning. In *Advances in neural information processing systems*, pages 4077–4087, 2017.

[130] Richard Socher, Danqi Chen, Christopher D Manning, and Andrew Ng. Reasoning with neural tensor networks for knowledge base completion. In *Advances in neural information processing systems*, pages 926–934, 2013.

[131] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. How to fine-tune bert for text classification? In *China National Conference on Chinese Computational Linguistics*, pages 194–206. Springer, 2019.

[132] Sheng-Li Sun, Q. Sun, Kevin Zhou, and Tengchao Lv. Hierarchical attention prototypical networks for few-shot text classification. In *EMNLP/IJCNLP*, 2019.

[133] Flood Sung, Yongxin Yang, Li Zhang, Tao Xiang, Philip H. S. Torr, and Timothy M. Hospedales. Learning to compare: Relation network for few-shot learning. *CoRR*, abs/1711.06025, 2017.

[134] Yik-Cheung Tam. Cluster-based beam search for pointer-generator chatbot grounded by knowledge. *Computer Speech & Language*, 64:101094, 2020.

[135] Ian Tenney, Dipanjan Das, and Ellie Pavlick. BERT rediscovers the classical NLP pipeline. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4593–4601. Association for Computational Linguistics, July 2019.

[136] Julien Tissier, C. Gravier, and Amaury Habrard. Dict2vec : Learning word embeddings using lexical dictionaries. In *EMNLP*, 2017.

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

[138] Ashwin K. Vijayakumar, Michael Cogswell, Ramprasaath R. Selvaraju, Qing Sun, Stefan Lee, David J. Crandall, and Dhruv Batra. Diverse beam search for improved description of complex scenes. In Sheila A. McIlraith and Kilian Q. Weinberger, editors, *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018*, pages 7371–7379. AAAI Press, 2018.

[139] Oriol Vinyals, Charles Blundell, Timothy Lillicrap, Daan Wierstra, et al. Matching networks for one shot learning. In *Advances in neural information processing systems*, pages 3630–3638, 2016.

[140] Oriol Vinyals and Quoc V. Le. A neural conversational model. *CoRR*, abs/1506.05869, 2015.

[141] Ulrike von Luxburg. A tutorial on spectral clustering. *CoRR*, abs/0711.0189, 2007.

[142] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *ArXiv*, abs/1804.07461, 2018.

[143] Joe H Ward Jr. Hierarchical grouping to optimize an objective function. *Journal of the American statistical association*, 58(301):236–244, 1963.

[144] Jason Wei, Chengyu Huang, Soroush Vosoughi, Yu Cheng, and Shiqi Xu. Few-shot text classification with triplet networks, data augmentation, and curriculum learning. In *NAACL*, 2021.

[145] Jason Wei and K. Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *ArXiv*, abs/1901.11196, 2019.

[146] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, R'emi Louf, Morgan Funtowicz, and Jamie Brew. Huggingface's transformers: State-of-the-art natural language processing. *ArXiv*, abs/1910.03771, 2019.

[147] Ledell Yu Wu, Adam Fisch, S. Chopra, Keith Adams, Antoine Bordes, and J. Weston. Starspace: Embed all the things! In *AAAI*, 2018.

[148] Yuting Wu, Xiao Liu, Yansong Feng, Zheng Wang, and Dongyan Zhao. Neighborhood matching network for entity alignment. *ArXiv*, abs/2005.05607, 2020.

[149] Congying Xia, Chenwei Zhang, Xiaohui Yan, Yi Chang, and Philip S Yu. Zero-shot user intent detection via capsule neural networks. *arXiv preprint arXiv:1809.00385*, 2018.

[150] Qizhe Xie, Zihang Dai, Eduard H. Hovy, Thang Luong, and Quoc Le. Unsupervised data augmentation for consistency training. In Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria - Florina Balcan, and Hsuan - Tien Lin, editors, *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, 2020.

[151] Pawel Swietojanski Xingkun Liu, Arash Eshghi and Verena Rieser. Benchmarking natural language understanding services for building conversational agents. In *Proceedings of the Tenth International Workshop on Spoken Dialogue Systems Technology (IWSDS)*, pages xxx–xxx, Ortigia, Siracusa (SR), Italy, April 2019. Springer.

[152] Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. Byt5: Towards a token-free future with pre-trained byte-to-byte models. *ArXiv*, abs/2105.13626, 2021.

[153] Jingru Yang, Ju Fan, Zhewei Wei, Guoliang Li, Tongyu Liu, and Xiaoyong Du. Cost-effective data annotation using game-based crowdsourcing. *Proc. VLDB Endow.*, 12:57–70, 2018.

[154] Min Yang, Junwu Xu, Kaifeng Luo, and Yiping Zhang. Sentiment analysis of chinese text based on elmo-rnn model. *Journal of Physics: Conference Series*, 1748, 2021.

[155] Yinfei Yang, Yuan Zhang, Chris Tar, and Jason Baldridge. PAWS-X: A cross-lingual adversarial dataset for paraphrase identification. 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 3687–3692, Hong Kong, China, November 2019. Association for Computational Linguistics.

[156] Liang Yao, Chengsheng Mao, and Yuan Luo. Graph convolutional networks for text classification. In *AAAI*, 2019.

[157] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proc. of 33rd ACL*, 1995.

[158] Mo Yu, Xiaoxiao Guo, Jinfeng Yi, Shiyu Chang, Saloni Potdar, Yu Cheng, Gerald Tesauro, Haoyu Wang, and Bowen Zhou. Diverse few-shot text classification with multiple metrics. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1*, New Orleans, Louisiana, June 2018. Association for Computational Linguistics.

[159] Chenwei Zhang, Y. Li, Nan Du, W. Fan, and Philip S. Yu. Joint slot filling and intent detection via capsule neural networks. *ArXiv*, abs/1812.09471, 2019.

[160] L. Zhang and B. Liu. Sentiment analysis and opinion mining. In *Encyclopedia of Machine Learning and Data Mining*, 2017.

[161] Shuailiang Zhang, Zhao Hai, Yuwei Wu, Zhuosheng Zhang, X. Zhou, and Xiaoping Zhou. Dual co-matching network for multi-choice reading comprehension. *ArXiv*, abs/1908.11511, 2020.

[162] Yizhe Zhang, Guoyin Wang, Chunyuan Li, Zhe Gan, Chris Brockett, and Bill Dolan. Pointer: Constrained progressive text generation via insertion-based generative pre-training. In *EMNLP*, 2020.

[163] Yuan Zhang, Jason Baldridge, and Luheng He. PAWS: Paraphrase adversaries from word scrambling. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 1298–1308, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics.

[164] Shiqi Zhao and Haifeng Wang. Paraphrases and applications. In *Coling 2010: Paraphrases and Applications–Tutorial notes*, pages 1–87, Beijing, China, August 2010. Coling 2010 Organizing Committee.

[165] Taotao Zhao, Xiangfeng Luo, Wei Qin, Subin Huang, and Shaorong Xie. Topic detection model in a single-domain corpus inspired by the human memory cognitive process. *Concurrency and Computation: Practice and Experience*, 30:e4642, 08 2018.

[166] Y. Zhao, J. Li, Xiaorui Wang, and Y. Li. The speechtransformer for large-scale mandarin chinese speech recognition. *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 7095–7099, 2019.

[167] Zhi-Hua Zhou and Ming Li. Tri-training: exploiting unlabeled data using three classifiers. *IEEE TKDE*, 17(11):1529–1541, Nov 2005.

[168] Xiangyang Zhou, L. Li, Daxiang Dong, Y. Liu, Ying Chen, Wayne Xin Zhao, D. Yu, and Hua Wu. Multi-turn response selection for chatbots with deep attention matching network. In *ACL*, 2018.

[169] Y. Zhou and S. Goldman. Democratic co-learning. *16th IEEE International Conference on Tools with Artificial Intelligence*, pages 594–602, 2004.

[170] Xiaojin Zhu. Semi-supervised learning literature survey. 2005.

[171] Xiaojin Zhu, John Lafferty, and Ronald Rosenfeld. *Semi-supervised learning with graphs*. PhD thesis, Carnegie Mellon University, language technologies institute, 2005.