UNIVERSITY OF TRENTO - Italy

Department of Information Engineering and Computer Science

Master's degree in
Computer Science

FINAL DISSERTATION

LANGUAGE MODELING: CONTINUOUS SPACE REPRESENTATION FOR FINITE STATE MODELS

Supervisor

Prof. Giuseppe Riccardi

Student

Andrei Catalin Coman

Academic year 2018/2019

Acknowledgments

Vreme trece, vreme vine,
Toate-s vechi şi nouă toate;
Ce e rău şi ce e bine,
Tu te-ntreabă şi socoate;
Nu spera şi nu ai teamă,
Ce e val ca valul trece;
De te-ndeamnă, de te cheamă,
Tu rămâi la toate rece.

Tempo passa, tempo viene,
Tutto è antico e tutto è nuovo;
Ciò ch'è bene e ciò ch'è male,
Sol domandati e rifletti;
Non sperare e non temere,
Ciò ch'è onda, com'onda passa;
Se ti esorta, se ti chiama,
Resta freddo a ogni cosa.

Time goes by, time comes along,
All is old and all is new;
What is right and what is wrong,
You must think and ask of you;
Have no hope and have no fear,
Waves that rise can never hold;
If they urge or if they cheer,
You remain aloof and cold.

Glossă, Mihai Eminescu

I would like to thank my parents Daniel and Emilia for their support during this academic journey. If I am in the position to write these lines today, it is only because of the many sacrifices they have had to make to allow me to continue my studies. I have enormous respect for them and I consider myself fortunate to have been able to count on their guidance and affection. I would also like to give a special thanks to my little sister Laura, the real engine of our family, who, despite being a little girl, has been able to teach us a lot of things.

I now address my thanks to my second family, my friends. I want to thank Mahed, Giacomo, Stefano, Lorenzo, Elia, Cosimo, Andrea, and Paolo for their support during all these years. We shared a lot of experiences, both positive and negative. We have been able to support each other and move forward together towards new goals. I am confident that the bonds we have established over time are solid, and will remain so for years to come, no matter where each of us will end up settling down.

Finally, I want to address my thanks to all the members of SiSLab, especially Alessandra, Yasin, and Aniruddha, who have contributed to making the working environment positive and stimulating. This laboratory exists thanks to the immense commitment of Professor Riccardi. I want to thank him for involving me in his research, for the numerous opportunities he has offered me and for the continuous support in carrying out the work within this document.

Contents

List of Figures

List of Tables

Abstract

Statistical Language Models (SLMs) are a fundamental component of Automatic Speech Recognition (ASR) systems. In recent years, Neural Language Models (NLMs) have proven to be significantly superior to more traditional Back-Off N-gram Language Models (BLMs). However, while these models can model longer history contexts, i.e. longer word sequences, they showed to be much slower during the decoding phase when compared to N-gram-based models. This is due to the fact that an NLM-based decoder has a much larger search space compared to a BLM. That is, it has to consider a larger number of hypotheses and therefore the processing time increases. This may pose a problem if the overall latency of the ASR system represents a crucial factor. To this end, this work is dedicated to the investigation of different methodologies for the creation of a finite representation of an NLM. This representation takes the form of a Stochastic Finite State Model (SFSM), i.e. a model with a finite number of states, where transitions between states are governed by probability values. Since an NLM can encode an unbounded number of history contexts, the reduction of this model to a limited number of states represents a challenge. This work exploits the sentences contained in the Airline Travel Information System (ATIS) text corpus for the creation of the SLMs. Starting from the NLM, an initial top-down strategy is proposed, which consists of considering all the different history contexts related to a specific word, i.e. all the different prefixes of the sentences containing the word. The contexts are then aggregated to form the representation of the word as a state of the SFSM and for the estimation of transition probabilities. The second proposed strategy involves a combination of the BLM and the NLM. The first model defines the structure of the final SFSM, i.e. states and transitions, while the second model is used for the estimation of probabilities. The third strategy instead tries to look at the problem from a different perspective and proposes a bottom-up strategy where all the history contexts are on the same level. These history contexts are then aggregated through a clustering process, whose output defines the states of the final SFSM. Transitions between states are then constructed using a states exploration algorithm. Finally, the work also aims to demonstrate how history contexts can be used in a dynamic decoding context, where sentences may not necessarily be independent and identically distributed. The results obtained from the various strategies are promising and establish a solid starting point for future research in this area.

Keywords: Back-Off Language Model, Neural Language Model, Stochastic Finite State Model

Chapter 1

Introduction

The ease of use and omnipresence of technology combined with the amount of information exchanged have led to the creation of new dynamics of interaction and the establishment of new communication links at both individual and societal levels. This has been made possible by the expansion of communication networks, which have simplified the interactions between technology and society. We, can therefore, say that digitization seems to have eliminated most of the barriers previously represented by time and space. The removal of these barriers has contributed to an increased degree of mobility, which in turn has led to the creation and consumption of massive amounts of information. This has also contributed strongly to the recent trend in basing technological evolution on data, information and knowledge [1].

Information is often transmitted in the form of text expressed in natural language, i.e. human language. This represents a challenge when computers are being used to disentangle and structure various pieces of knowledge in such a way as to facilitate access and use by the general public. From a scientific point of view, natural language has always represented an interesting object of study, succeeding in involving different disciplines. One of the major exponents in this subject is represented by Linguistics, i.e. the science that deals with the study of language and its dynamics. However, interest in this topic has spread to other domains, including Philosophy, Psycholinguistics, Mathematical Linguistics, Computational Linguistics, and Natural Language Processing. This document will address topics that fall under the last three domains.

Mathematical Linguistics as a discipline aims to delineate a mathematical formalism through which to describe natural language. It, therefore, covers the study of mathematical models and procedures to delineate the phenomena that govern language and its dynamics. These models do not use computerized techniques. The mathematical models of language are analytical and generative. They constitute mathematical constructs that capture certain relational aspects of linguistic phenomena. Their role is to arrange already known notions and relationships, as well as to discover new relationships and ways of organization.

Computational Linguistics, on the other hand, exploits computerized techniques to perform language research. This involves the definition of data structures, algorithms, formal representational and reasoning models, as well as Machine Learning techniques, aimed at solving problems such as identifying sentence structure, computer-aided translation, and statistical language processing.

Natural Language Processing is a set of processes, methods, and operations that create and implement ways of performing different tasks related to natural language. These tasks may concern morphology, which deals with the composition of words and their relationship with other words, or syntax, which

defines the structure according to which words are related within a sentence, or even semantics, which deals with the meaning of words and groups of words.
This work aims to address several challenges arising from the integration of Language Models (LMs) into Automatic Speech Recognition (ASR) systems, where latency is a crucial factor. The creation of LMs will be analyzed from two different perspectives, initially looking at Back-Off N-gram Language Models (BLMs) and eventually arriving at the most recent ones based on Neural Networks (NNs). The advantages and shortcomings of each model will be highlighted. The ultimate goal is to build a Stochastic Finite State Model (SFSM) that combines these two approaches in order to exploit the strengths of each model.

1.1 Automatic Speech Recognition

Speech Recognition is a complex process, and the development of technology in this direction has a long history that dates back to the first attempts made in 1784 by Wolfgang von Kempelen, who developed the first mechanical acoustic speech device that was able to utter sounds and words [2]. Automatic Speech Recognition (ASR) represents an evolution in the Human-Computer Interaction area. A user provides a vocal input to the system, which can identify words and expressions in the spoken language and convert them into text transcripts. The latter can be exploited within a Language Understanding System, which is concerned with understanding words and the way they are used in the communication process. This system can, in turn, be part of a larger system that is responsible for carrying out actions based on requests received from the user. It is, therefore, essential to have a robust ASR system, as several downstream processes may depend on its output.

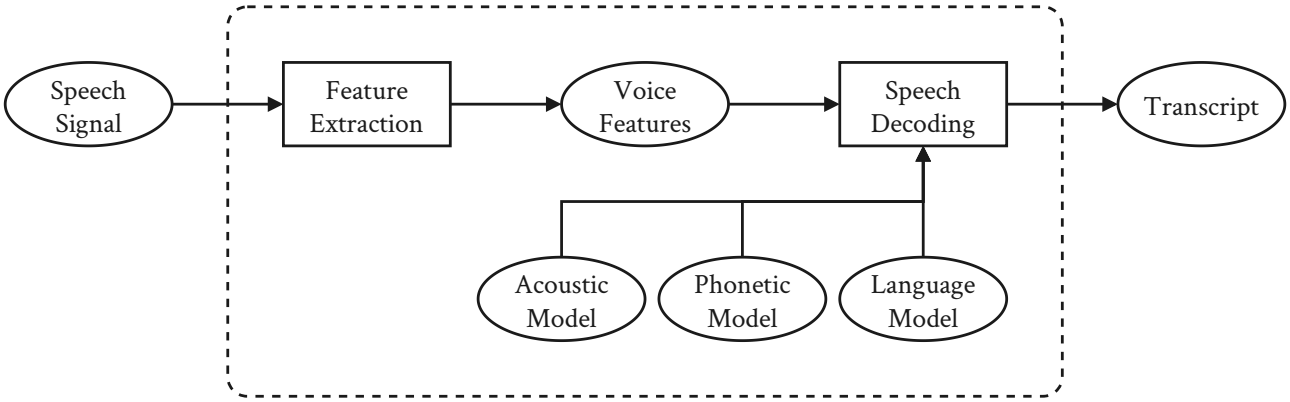


Figure 1.1: Decoder component of an ASR system.
Decoding of a speech signal into a text transcript.

The process of decoding a speech signal into a text transcript is shown in Figure (1.1) and consists of the following components:

- Speech Signal: input voice signal from the human counterpart, which has to be converted into a text transcript by the system
- Feature Extraction and Voice Features: this phase deals with extracting voice parameters from the speech signal
- Speech Decoding: this module aggregates the output from the Feature Extraction module with the output from the Acoustic, Phonetic and Language Models to decode the speech signal

- Acoustic Model: this model contains acoustic properties for each sound. It is used to represent the relationship between an audio signal and the phonemes or other linguistic units that make up speech
- Phonetic Model: this model acts as a bridge between the Acoustic and the Language Model. It is often represented in the form of a dictionary of pronunciations that associates words with their phonemes
- Language Model: this model is used to restrict the search space of words. It determines which word is most probable to follow after the previously recognized ones and helps to significantly restrict the speech-word matching process by eliminating improbable words
- Transcript: output in the form of text transcript as a result of processing the input speech signal

Within this work, efforts will be mainly focused on the Language Model (LM) module, more specifically on the development of Statistical Language Models (SLMs). The goal is to incorporate the benefits of Neural Language Models (NLM) within an ASR architecture. The challenges arising from the use of (NLMs) will be presented, and the differences with more traditional BLMs will be highlighted. Several attempts to combine these models will be reported, from the creation of hybrid models to the conversion of the NLM into a Stochastic Finite State Model (SFSM).

1.2 Language Modeling

Despite countless advances in this area, modeling human language is still a challenge today. In this section, two approaches to this problem will be presented, namely Structural and Statistical Language Modeling. The first approach is dedicated to the identification of a relatively rigid structure to describe the relationships between words, while the second approach is based on word statistics, i.e. probabilistic estimates inferred from a text corpus.

The approach based on Structural Language Modeling aims to formalize natural language through the use of the so-called Context-Free Grammars (CFGs), a set of rules that define the structure of the grammar that governs a specific language. This approach is based on the distinction within the sentences of the nominal parts, whether they relate to the subject, the object or the predicate, thus outlining a hierarchical structure. Moreover, depending on the type of sentence, whether it is of a coordinate or subordinate type, further hierarchical layers can be defined [3]. Grammar rules can be used both for analysis, i.e. the acceptance or rejection of a sequence of words, as well as for generating new sequences that belong to the language. However, constructing a complete CFG for a language is prohibitive, as it would be necessary to define a large number of rules, which must be agreed upon with language experts. To cope with this rigidity, several studies such as [4] have led to the development of Probabilistic Context-Free-Grammars and Lexicalized Probabilistic Context-Free-Grammars, which are more flexible since the validity of a sentence is expressed in probabilistic terms. However, these models are quite inefficient and their use is more appropriate for tasks where the vocabulary of the language is limited.

Statistical Language Models (SLMs) shift the dependency from language experts to text corpora. These models are of frequentist type, i.e. they represent a specific language as the frequency of occurrences for any possible word sequence. These models try to capture the regularity of a language through statistical inference on a large text corpus [5]. Unlike Structured Language Models, where a

specific sequence is either accepted or not, based on its syntactical structure, SLMs assign a non-zero probability even to syntactically incorrect sentences. While this may be considered a drawback, it also shows that these models are inherently more flexible, as they lack an explicit grammar. SLMs are easier to integrate into recognition systems, such as the ASR system under analysis in this work. This is due to the margin of flexibility in the decoding phase, as these models are inherently probabilistic. In fact, an SLM estimates the probability of different sequences during the conversion of a speech signal into a sequence of words and provides the most probable one as output.
In the next section, the functioning of an SLM will be presented in more details, highlighting two different typologies, which will constitute the basis of the study of this work.

1.3 Statistical Language Modeling

The term Statistical Language Modeling (SLM) refers to the creation of a probabilistic model capable of assigning a probability value to word sequences. This corresponds to a measure of positional adequacy of a word sequence. The measure of adequacy can be seen in terms of assigning a probability value to each word within a sequence, based on the words previously seen within the sequence itself. This corresponds to a sequential data prediction problem. In more formal terms, the SLM estimates the probability of a sequence of N words $W = w_1, w_2, \ldots, w_N$ as follows:

$$P(W) = \prod_{i=1}^{N} P(w_i|h_i)$$

where $P(w_i|h_i)$ represents the probability that w_i occurs after the history of words $h_i = w_1, \ldots, w_{i-1}$ has occurred. For instance, given the sentence *"i need to fly between philadelphia and atlanta"*, the probability to be computed is the following:

$$\begin{aligned} P(\textit{i need to fly between philadelphia and atlanta}) = & P(\textit{i}) \cdot \\ & P(\textit{need}|\textit{i}) \cdot \\ & P(\textit{to}|\textit{need i}) \cdot \\ & P(\textit{fly}|\textit{to need i}) \cdot \\ & P(\textit{between}|\textit{fly to need i}) \cdot \\ & P(\textit{philadelphia}|\textit{between fly to need i}) \cdot \\ & P(\textit{and}|\textit{philadelphia between fly to need i}) \cdot \\ & P(\textit{atlanta}|\textit{and philadelphia between fly to need i}) \end{aligned}$$

where $P(w_i|h_i)$ can be estimated by means of the Maximum Likelihood Estimation (MLE) [6] as follows:

$$P(w_i|h_i) = \frac{C(h_i, w_i)}{C(h_i)}$$

where $C(x)$ represents a counting function that returns the number of occurrences of the argument x in the training corpus. This corresponds to determining the number of times event (h_i, w_i) has been

observed when event (h_i) has been observed. As simple as this definition of the problem may result, it does not lack pitfalls. One of the biggest problems is the sparsity of data and the resulting lack of reliability of the counts returned by function C. To have good estimations, it is necessary to have a very large data set, covering a large number of sentences. However, this does not cover the possibility of encountering extensions of the sentences on which the model has been trained on, which will not be able to provide probability estimations as the C function will return a 0 value for new sentences. The problem of estimating $P(W)$ can also be seen from a different perspective. For the estimation of this probability one could consider all sentences with length equal to $|W|$ and check how many times the exact sequence W is present between them. This shows how easily the problem can become intractable [7]. It is, therefore, necessary to introduce new methods for estimating probabilities. The next subsections will introduce two ways of addressing this problem.

1.3.1 N-gram Language Model

N-gram Language Models address the problem of probability estimation by assuming that history h_i can be approximated by fewer words. This means that the conditional probability $P(w_i|h_i)$ can be expressed as $P(w_i|\tilde{h}_i)$ where $|\tilde{h}_i|$, i.e., the size of history $\tilde{h}_i$, is smaller than $|h_i|$, i.e. the size of history h_i. This assumption is called the Markov assumption, and is a characteristic of stochastic processes where the Markov property holds [8]. This property is defined as follows:

Definition 1 ***Markov property*** *[9]*
"In probability theory, Markov property refers to memoryless property of a stochastic process. The latter has the Markov property if the probability distribution of future states of the process conditioned on both the past and present states depends only on the present state. In other words, predicting the next word in a sentence depends only on the current word, and not on the words that came before the current word"

An N-gram LM corresponds to a Markov model of order $N-1$, and the probability estimation can be seen as:

$$P(w_i|\tilde{h}_i) = P(w_i|w_{i-N+1}^{i-1})$$

where the new history $\tilde{h}_i = w_{i-N+1}^{i-1} = w_{i-N+1}, \ldots, w_{i-1}$ which differs from the original history definition $h_i = w_1^{i-1} = w_1, \ldots w_{i-1}$. The probability estimation $P(w_i|\tilde{h}_i)$ uses the MLE as follows:

$$P(w_i|\tilde{h}_i) = \frac{C(\tilde{h}_i, w_i)}{\sum_{w_j \in V} C(\tilde{h}_i, w_j)}$$

where V represents the vocabulary of the training text corpus. Generally the order N of the N-gram model varies from 2 to 5, while higher orders do not bring significant improvements to the LM [10]. Despite improvements over the model introduced at the beginning of Section (1.3), this model is not free of problems. It must be noted that the number of parameters of the model increases exponentially as the order N increases. For instance, with a vocabulary size $|V| = 1000$ and order $N = 3$ the number of probabilities the model has to estimate corresponds to $|V|^3 = 1000^3 = 1000000000$ (1 billion). This, therefore, imposes a limitation on the size of the history $\tilde{h}_i$ that can be used. Furthermore, the problem regarding the number of occurrences persists. To deal with this problem, Section (3.3) will present one of the models that part of the experiments conducted in this work are based on.

The solution to this problem consists of assigning by means of back-off and smoothing techniques, a probability value different from 0 even to N-grams that were not directly observed in the training text corpus.

1.3.2 Neural Language Model

SLMs based on NNs were introduced as a method of solving the problem of data sparsity. The problem of smoothing is intrinsically solved by these models as they operate on a continuous space representation of words, allowing them to effectively model word sequences and to generalize to new sequences. These models have proven to be significantly superior to N-gram LMs [11]. Figure (1.2) shows how an NN model can be used for the SLM problem. This architecture contains a main component called Recurrent Neural Network (RNN), which is based on the Elman Network [12].

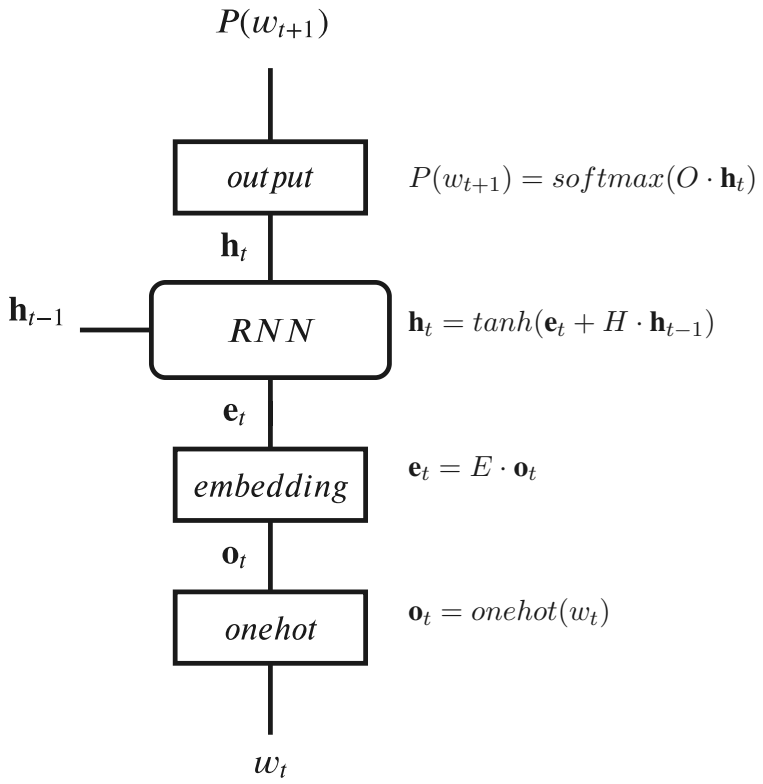


Figure 1.2: Recurrent Neural Network.
Probability estimation for the next word w_{t+1} given the current word w_t

The network receives as input a word w_t at time step t which is converted into an one-hot vector $\mathbf{o}_t$ as follows:

$$\mathbf{o}_t = onehot(w_t)$$

The $onehot$ function maps the word w_t to a binary vector of length equal to $|V|$, where V represents the vocabulary of the training text corpus. This vector contains value 1 only in the position corresponding to word w_t.

The $\mathbf{o}_t$ vector is then linearly projected to vector $\mathbf{e}_t$ by means of the projection matrix E. This phase is often called $embedding$, where the information is condensed into a continuous value vector as follows:

$$\mathbf{e}_t = E \cdot \mathbf{o}_t$$

The central part of this architecture is represented by the RNN cell, which computes:

$$\mathbf{h}_t = tanh(\mathbf{e}_t + H \cdot \mathbf{h}_{t-1})$$

where $\mathbf{h}_t$ represents the hidden state of the cell at time step t. This vector can be seen also as the encoded history of the sequence of words up to time step t, included. This function is special since the previous cell state $\mathbf{h}_{t-1}$ at time step $t-1$ is used for the computation of the new cell state $\mathbf{h}_t$ at time step t. The cell state $\mathbf{h}_{t-1}$ at time step $t-1$ is linearly projected into a new vector by the projection matrix H, followed by an element-wise sum with the embedded representation $\mathbf{e}_t$ of word w_t at time step t. The sum vector is then passed through the hyperbolic tangent function $tanh$, which acts as a non-linear activation function as follows:

$$tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \in [-1, 1]$$

The $output$ layer maps the newly computed hidden state $\mathbf{h}_t$ to a probability distribution over the words in the vocabulary V. More precisely, it takes care of estimating the probability of word w_{t+1} at time step $t+1$, i.e the probability of the next word. This is done as follows:

$$P(w_{t+1}) = softmax(O \cdot \mathbf{h}_t)$$

The probability of each word must be greater than 0 and the sum of probabilities must equal to 1. To ensure that, the $softmax$ function is applied to the output of the linear projection of the hidden state $\mathbf{h}_t$ by means of the projection matrix O. This function computes:

$$softmax(\tilde{w}_i) = \frac{e^{\tilde{w}_i}}{\sum_j e^{\tilde{w}_j}} \in [0, 1]$$

where $\tilde{w}_i \in \tilde{\mathbf{w}} = O \cdot \mathbf{h}_t$. The network weight matrices E, H and O are learned by optimizing the value of an objective function such as the Cross Entropy (CE) [13] using gradient descent optimization techniques such as Adam [14] by means of the Back Propagation Through Time algorithm [15]. Section (3.4) will present the variant of RNN used to conduct part of the experiments carried out within this work.

1.4 Contribution and Thesis Organization

The way word sequences are modeled, and primarily the history context h_i represents the crucial point that distinguishes the two SLMs that have just been presented. NLMs have proven to be significantly superior to traditional LMs based on N-grams, due to their ability to model unlimited history contexts [11]. However, while this can be an advantage, it can become a disadvantage when an NLM has to be incorporated directly into a real-time large vocabulary ASR system. In other words, in a system where the number of plausible acoustical hypotheses can become very large, the use of an NLM in

the decoding phase can be computationally demanding, leading to delays in overall system responsiveness. The decoder's search space can grow exponentially due to the way history is modeled by the NLM. This problem does not occur when N-gram LMs are used, as their search space is significantly smaller.

The most common approach for decoding within an ASR system is to divide the process into two steps. The first step consists of performing a so-called first-pass decoding by means of the N-gram LM, thus pruning the search space and extracting an m-best list of word sequences. This word lattice is then passed through the NLM for the so-called second-pass decoding, which will then re-score the word sequences, identifying the most plausible one, which will then be chosen as the final decoding. One of the problems with this strategy is that the NLM is only applied at the end of the processing phase by the N-gram LM. As a result, this second step will inevitably result in longer system response times. Another problem that must be taken into account is that a much more powerful model, i. e. the NLM, is not used in the first decoding step, thus leading to the loss of plausible hypotheses that the N-gram LM is not able to capture.

It is useful to note how probability $P(w_i|h_i)$ can be seen as the transition from a state s_i represented by history h_i to a subsequent state s_{i+1} via the arc identified by the word w_i. The weight of the transition is represented by the probability value. The set of states S and transitions T, therefore, encode a Stochastic Finite State Model (SFSM). A high-level representation of this model is shown in Figure (1.3), where two special states, <s> and </s> have been added to S, indicating the start of the sentence and the end of the sentence, respectively. An N-gram LM conforms precisely to this structure. In fact, the finite set of histories h_i represent the states of the model, while transitions are encoded by words w_i in the vocabulary V of the train corpus, and the weights of these transitions are the probabilities of the N-gram.

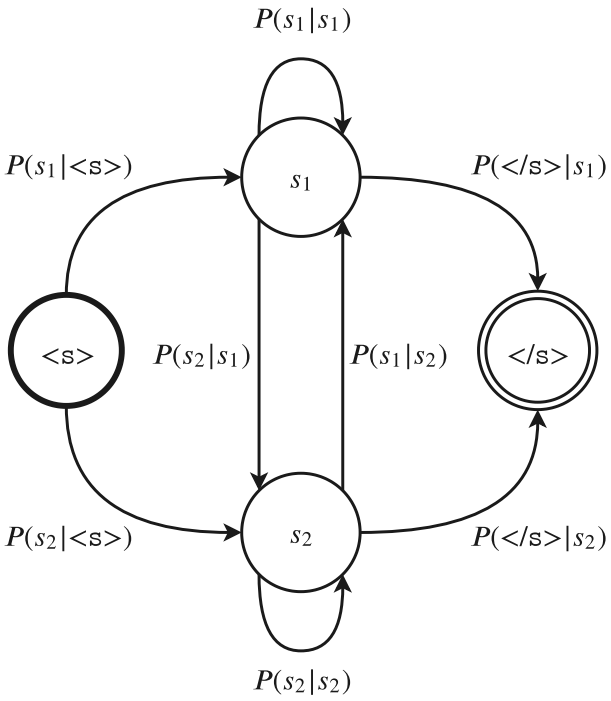


Figure 1.3: Stochastic Finite State Model.
A high-level illustration of transitions between states

The same thing does not apply to an NLM. This model can encode an infinite number of history contexts, as it operates with a continuous value vector representation of history h_i. A reduction to an SFSM is, therefore, only an approximation of the NLM. However, this reduction would have the advantage of allowing the model to be used in the same way as the N-gram LM, with all the benefits that this brings when used within the decoding module of the ASR system. The objective of this work is, therefore, to reduce the NLM to a SFSM, i.e. a static model that has the same structure as an N-gram LM. In doing so, it is important to delineate strategies that attempt to preserve as much as possible of the history context modeling power of the NLM.

This work as the first step will create a baseline model based on a BLM by exploiting the Airline Travel Information System (ATIS) corpus. The same data set will be used for training the NLM, and emphasis will be placed on how different train and evaluation techniques for this model can lead to different results when a mismatch between these two phases occurs. For the creation of an SFSM from the NLM model, firstly, a top-down strategy will be proposed and which will involve taking into account all the history contexts related to a specific word, i.e. all the different prefixes of the sentences in the train set that contain the word. Those contexts will be then aggregated to form the representation of the word as a state of the SFSM and for the estimation of transition probabilities.

The second strategy will involve a combination of BLM and NLM, where the first model will be used as the structure of the final SFSM, while the second model will be used to estimate the probabilities of transitions. The BLM model will thus provide the set of states and transitions, while the NLM model will be used for the estimation of transitions probabilities.

The third strategy will consist of a bottom-up approach, where all the history contexts representing the words in the text corpus will be considered on the same level, without defining an a priori strategy of how to aggregate them. Their combination will be left to a clustering procedure, which will combine them based on their semantic similarity. The number of clusters at the end of this process will constitute the set of candidate states of the final SFSM. The connection between the states will be determined by a states exploration algorithm based on the observed and unobserved transitions within the text corpus.

Finally, this work will also show shown how in a dynamic text decoding context, it is possible to exploit the history contexts of the train set. This will prove that this type of decoding can be useful even in conditions where sentences are not necessarily independent and identically distributed.

The results obtained from the various strategies will prove promising and will establish a solid starting point for future research in this area.

This document is structured as follows:

- In Chapter (2) several works in the literature related to the creation of LMs, whether based on N-grams or NNs, will be presented. Various approaches for the conversion of NLMs into an SFSM will also be presented.
- Chapter (3) will formalize the task, define the approach, present the data and its analysis, as well as the creation of the BLM and NLM and their evaluation.
- Chapter (4) will present the experiments carried out in this work, together with the discussion on the obtained results.
- Finally, Chapter (5) will concentrate on the conclusions and possible directions for further research.

Chapter 2

Literature Review

Before the use of Neural Networks (NNs) for Statistical Language Modeling (SLM), Back-Off N-gram Language Models (BLM) represented the main approach. These models have been successfully used in several areas such as Machine Translation [16], Part-Of-Speech tagging [17], and Speech Recognition [18]. The construction of LMs based on N-grams requires the adoption of smoothing techniques for the redistribution of probability mass from observed events to unobserved ones. Over the years several smoothing techniques have been introduced including Katz [19], Witten-Bell [20], Jelinek-Mercer [21], Church-Gale [22] and Kneser-Ney [23]. The authors of [24] and [25] focused on some of those techniques and investigated the way in which the the size of the train corpus and the order of the N-gram LM affect the performance.
One of the most relevant works related to the the representation of an SLM by means of a Stochastic Finite State Model (SFSM) is the one done by [26]. The authors introduced the so-called Variable N-gram Stochastic Automaton (VNSA) model, i.e. an SFSM that approximates a BLM. A representation of a third-order VNSA is shown in Figure (2.1). This model corresponds to a BLM that encodes 3-gram probabilities. Each state of the model can contain a limited history of symbols and since the model is a third-order one, the maximum number of symbols each state can contain is equal to two.

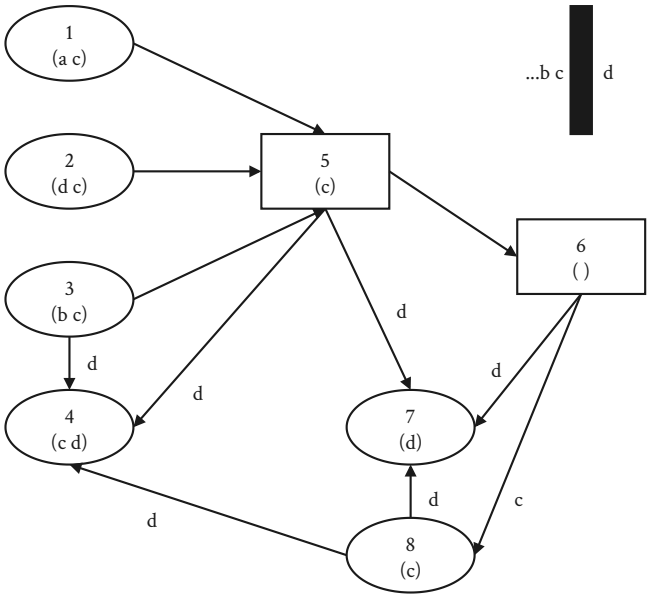


Figure 2.1: Portion of a third-order VNSA network [26]

In the figure, the automaton has processed the symbol b and c and is about to process the symbol d. For the resolution of this 3-gram, the transition from state 3 to state 4 can be used. An alternative is

represented by the transition from state 3 to state 7 passing through the intermediate state 5. State 5 and 6 are special states that allow the release of a symbol from history. This can be seen as a back-off mechanism that uses a lower order N-gram to decode the sequence.

One of the first works to use a Feed Forward Neural Network (FFNN) for SLM is the one carried out by [27]. A representation of the model used in this work is shown in Figure (2.2). The network takes as input at time step t a context of n words $\{w_{t-n}, w_{t-1}, w_t\}$ which are mapped into their one hot representation $\{\mathbf{o}_{t-n}, \mathbf{o}_{t-1}, \mathbf{o}_t\}$ using the $onehot$ function, followed by a mapping into a continuous value representation $\{\mathbf{e}_{t-n}, \mathbf{e}_{t-1}, \mathbf{e}_t\}$ by means of the $embedding$ function. Next the embedded representations are concatenated into a single vector $\mathbf{h}_t$ using the $concatenation$ function. Afterwards the $\mathbf{h}_t$ vector is mapped into a probability distribution over the words in the vocabulary V by means of the $output$ function. The $onehot$, $embedding$ and $output$ functions follow the same logic as those presented in Subsection (1.3.2).

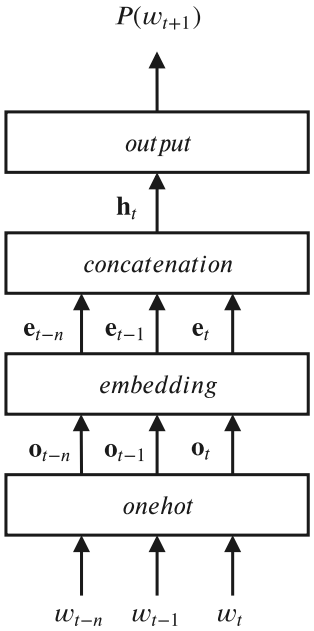


Figure 2.2: FFNN for SLM [27]

A further step in this direction was taken by the authors of [11], who overcame the limitation of the context to n words by proposing instead to exploit an architecture based on Recurrent Neural Networks (RNNs). They showed that connectionist LM are superior to traditional N-gram-based LMs. Details on the internal operation of an RNN are given in Subsection (1.3.2).

The problem of integrating NLMs into an ASR system by converting an RNN into a BLM has been addressed by several works. One of these is represented by [28], whose operation is shown in Figure (2.3). From the same text corpus, the authors created a BLM baseline and trained an RNN-based NLM. As the NLM is a generative model, Gibbs sampling [29] was used to generate new text data and create a new BLM. This new LM was then interpolated with the baseline BLM for the creation of the final model.

A similar strategy was also employed by [30], which used multi corpora for training the NLM. In addition to this, for the reduction of training time, the authors exploited the Noise Contrastive Estimation [31], which does not require the computation of the full $softmax$ function during training. The authors also used a strategy based on the quantization of weights and activations of the trained model to reduce inference time.

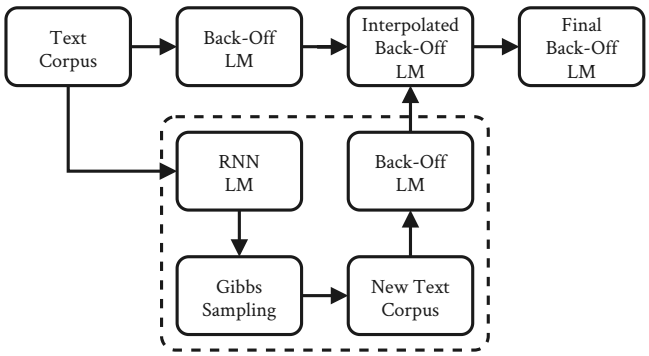


Figure 2.3: Conversion of an RNN into a BLM [28]

In the work done by [32], the model derived from the combination of the BLM and the NLM has been called the Background Language Model. This model involves the combination of the two models in a hierarchical way. For each order of the N-gram of interest, a BLM and a FFNN-based NLM are created. Since the FFNN model can become very large, the authors applied an entropy-based pruning. To deal with the case in which the NLM model is not able to provide a probability estimation for a specific N-gram, the authors propose to back-off to the BLM model and use its probability estimations. In the combination of the two models the N-grams are considered in a hierarchical way, that is a lower order N-gram defines which N-grams will be taken into account by a higher order.
The idea of entropy-based pruning has also employed in the work carried out by [33], where a strategy for the conversion of an RNN-based NLM into a Weighted Finite State Transducer (WFST) is proposed. This strategy is based on the discretization of continuous representations of the hidden states of the NLM through the application of clustering techniques. Chapter (4) will outline some experiments carried out in this direction, with a slightly different strategy than the one adopted by the authors of this work.
The idea of converting an RNN into a WFST was also addressed by [34], who proposed to represent the text corpus as a collection of so-called Soft Surface Patterns. An example of this kind of patterns is shown in Figure (2.4) where a specific class of sentences is encoded. The presence of epsilon transitions and wildcards allow the encoding of multiple sentences through a single pattern.

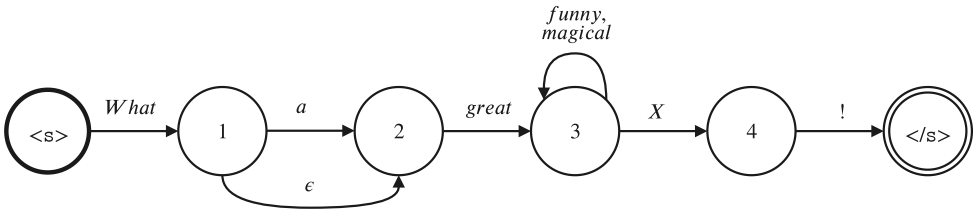


Figure 2.4: Soft surface patterns [34]

Spectral learning algorithms have also been used in this direction. For instance, the works carried out by [35] and [36] focus on the conversion of an RNN into a WFST, through the estimation of a so-called Hankel matrix. This matrix acts as a bridge between the two models and the values contained within it are estimated by the RNN and represent transitions between states. A low rank decomposition procedure such as the Singular Value Decomposition [37] is applied to the matrix for the creation of the WFST states.

Chapter 3

Task Formalization

This chapter is concerned with presenting the research area on which this work focuses and the final objective. In the following section, the task structure will be presented at a high-level. Next, the data set used for the SLMs training will be presented together with its analysis. Then the description of the N-gram BLM and the NLM used in this work will follow. Finally, the evaluation of the SLMs will be presented.

3.1 Task and Approach

The goal of this work is to incorporate an NLM into a real-time ASR system, where latency plays a crucial factor. As already mentioned in Chapter (1), this may pose a challenge as an NLM is computationally more expensive than a BLM, given that the search space of an NLM is much larger compared to a traditional BLM.

The structure of an BLM can be seen from the perspective of an SFSM, consisting of a series of states and transitions. On the other hand, this similarity is not so evident when considering an NLM since this model works on a continuous feature representation space, and consequently with an unbounded number of states and transitions. The goal of this work is the reduction of an NLM into a finite model to be compatible with the definition of an SFSM. Figure (3.1) outlines at a high level the various steps involved in achieving this goal.

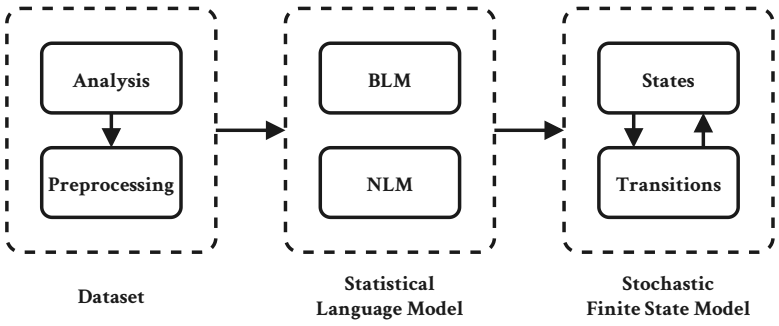


Figure 3.1: Task pipeline.
Building an SFSM from a data set via SLMs

The first component of the figure focuses on the data set. The description of the data set will be presented in Section (3.2), while details of the analysis and preprocessing will be provided in Subsection (3.2.1) and (3.2.2), respectively. The second component will be examined in Section (3.3) and (3.4), which will deal with the description of the BLM and NLM models used in this work, as well as the

training procedure of each model. The evaluation metric adopted for the two models will be described in Section (3.5). The description of the last component related to the construction of an SFSM from SLMs will be addressed in Chapter (4), which will describe in detail all the phases involved in the construction of the SFSM, together with the results obtained.

3.2 Data set

The data set used within this work is the Airline Travel Information System (ATIS), which was first introduced in the early 1990s. This data set was mainly used for Spoken Language Understanding system tasks, such as slot filling. The version adopted in this work is the one used in [38]. It consists of spoken queries made by users requesting flight information. Table (3.1) shows a small sample of users' information requests. The next subsection will conduct some analysis on the data set, to have a broader perspective on the type of sentences and words it contains, as well as highlighting the differences between the train and test set.

ATIS train set
what type of plane is an m80
i would like to fly from denver to boston
is fare code b the same as business-class
show departures from atlanta for american
i'd like the earliest flight from dallas to boston
information on flights from boston to washington
which airlines have flights from denver to pittsburgh
find a flight on delta from philadelphia to san-francisco
when is the first flight in the morning from boston to denver
what is your least-expensive fare between atlanta and boston

Table 3.1: ATIS train set.
Samples of users' flight information requests

3.2.1 Data Analysis

Since the development of an SLM is entirely data-driven, it is necessary to carry out some analysis to gain a better understanding of the data and to discover potential issues. A first analysis is shown in Table (3.2), which illustrates the characteristics of the sentences and words that compose them, both for the train and test set. The Natural Language Toolkit library [39] has been used for the removal of stop words. The percentage of unknown words which refer to words present in the test set and missing in the train set is equal to 15%, technically referred to as the Out-Of-Vocabulary (OOV) rate.

Description	**Train**	**Test**
number of sentences	4978	893
average sentence length	12.48	11.33
number of words	52170	8333
number of unique words	1045	484
number of stop words	23523	3493
number of unique stop words	90	59

Table 3.2: ATIS train and test set data analysis

A subsequent analysis consists of verifying whether the words in the two data sets follow a simple mathematical form known as the Zipf's Law [40], which claims that a small number of events occur with high frequencies and at the same time a large number of events occur with low frequencies. The distribution of words in train and test set are reported in Figure (3.2) and Figure (3.3), respectively. The objective of these plots is to highlight potential discrepancies between the two sets. If the word distribution in the test set was different from the word distribution in the train set, then the language model, which was built from the train data, would be ineffective. The two above-mentioned distributions are in accordance with the Zipf's Law and are very similar to each other.

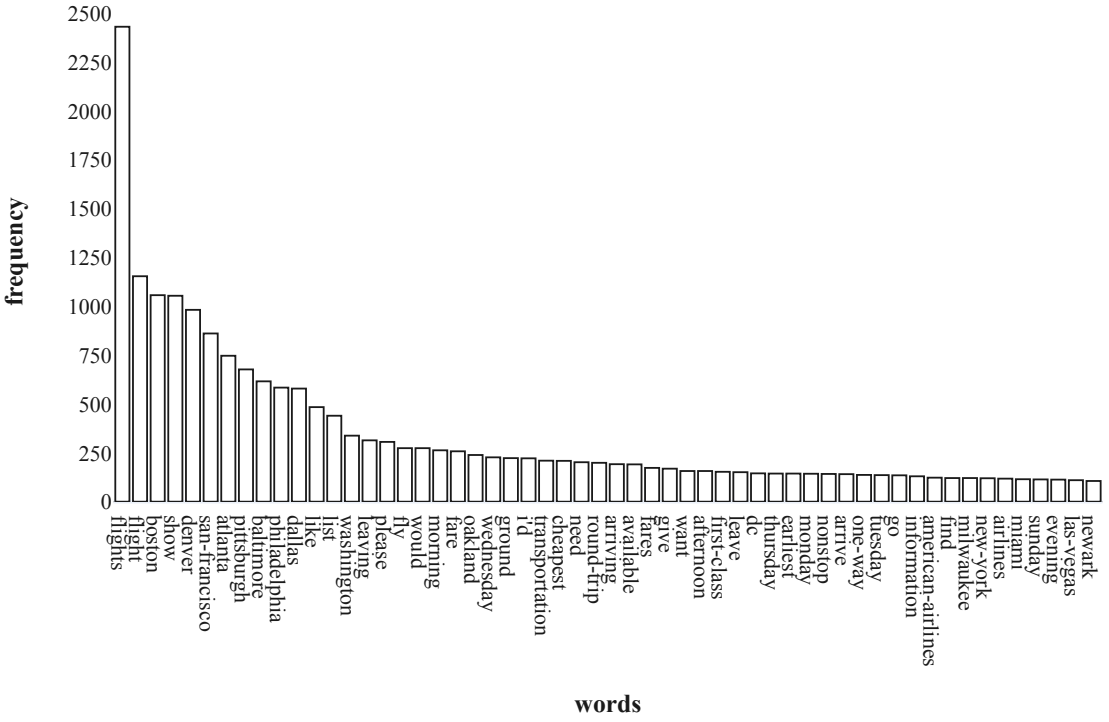


Figure 3.2: ATIS train set words distribution

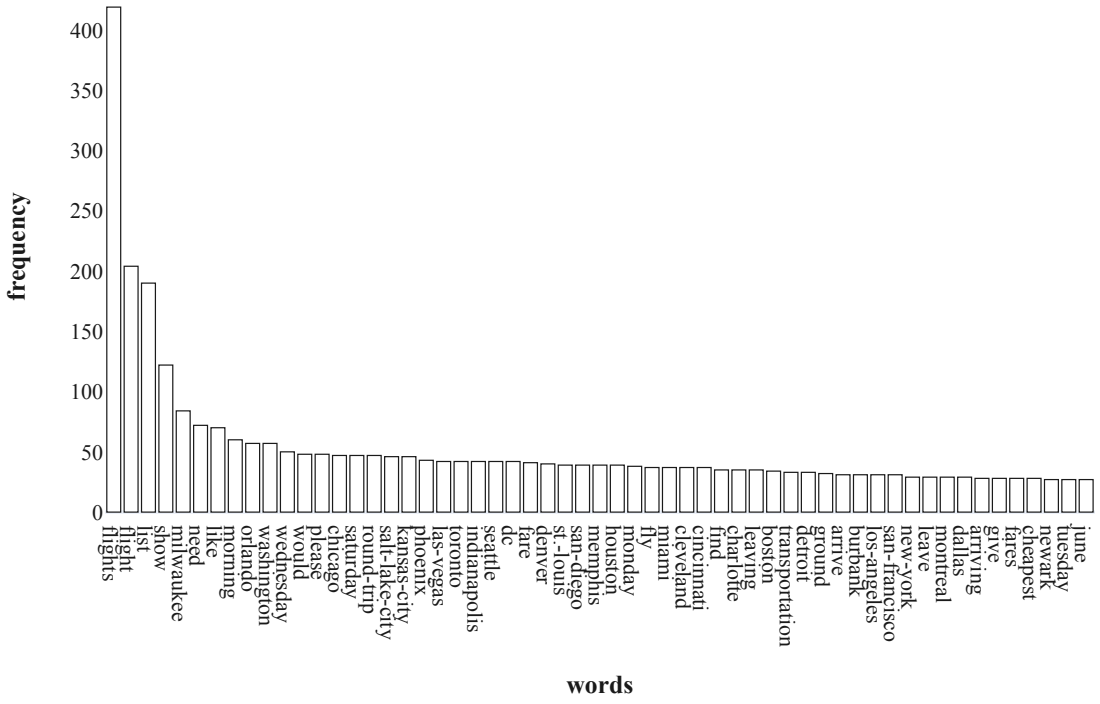


Figure 3.3: ATIS test set words distribution

3.2.2 Data Preprocessing

The equation used in Section (1.3) for estimating the probability of a sequence of words $P(W)$ does not take into account the case where i is equal to 1. To overcome this limit, delimiters indicating the start (<s>) and the end (</s>) of the sentence have been added to each sentence. The previous sentence "*i need to fly between philadelphia and atlanta*", therefore, becomes "<s> *i need to fly between philadelphia and atlanta* </s>". To reflect this change the previous probability computation will change in:

$$P(W) = \prod_{i=1}^{N+1} P(w_i|h_i)$$

The handling of OOV words is subject to different approaches in literature. To avoid any disagreement concerning the way these unknown words were handled, it was decided to exclude them completely from the experiments. Sentences from the test set containing unknown words have been removed, causing a reduction in the total number of test sentences from 893 to 802.

3.3 Building the N-gram Language Model

The introduction of the N-gram model in Subsection (1.3.1) highlighted the problem of estimating the probabilities of events that have never been observed within the train set. Since the counting function C, i.e. the function that counts the number of occurrences of a given event, returns a value equal to 0 for events that have never been observed, consequently the probability of these events will also be equal to 0. To overcome this problem, several smoothing techniques have been introduced. These techniques aim to redistribute the probability from observed events to unobserved ones. One of these techniques is represented by the Katz Back-Off method [19]. Considering that the experiments conducted in this work are limited to N-grams of order 2, the Katz method applied to the 2-grams, estimates the following probability:

$$P_{Katz}(w_i|w_{i-1}) = \frac{C_{Katz}(w_{i-1}, w_i)}{\sum_{w_i} C_{Katz}(w_{i-1}, w_i)}$$

where C_{Katz} represents a modified version of the counting function C and acts as follows:

$$C_{Katz}(w_{i-1}, w_i) = \begin{cases} d_r r & \text{if } r > 0 \\ \alpha(w_{i-1}) P(w_i) & \text{if } r = 0 \end{cases}$$

This counting function has several elements that must be analyzed individually. The first element to be taken into consideration is r, which represents the number of times the 2-gram (w_{i-1}, w_i) has been observed in the train set. Function C_{Katz} provides two alternatives, depending on whether the 2-gram has been observed at least once or not.

3.3.1 Case $r > 0$: Smoothing

In case the 2-gram has been observed at least once, i.e. $r > 0$, its count value will be equal to $d_r r$, where d_r is defined as:

$$d_r = \begin{cases} \dfrac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}} & \text{if } 1 \leq r \leq k \\ 1 & \text{if } r > k \end{cases}$$

The value of d_r is defined according to two subcases that depend on the value of r, which is the number of times the 2-gram has been observed and is bounded by a hyper-parameter k.

Subcase $r > k$

If the 2-gram has been observed at least k times, its number of occurrences r remains unchanged, thus $C_{Katz}(w_{i-1}, w_i) = d_r r = 1r = r$. Threshold k acts as a 2-gram reliability factor. Those who have a value greater than k are therefore to be considered more reliable than those who do not reach this value. Regarding the optimal value for k, [19] suggests that the most appropriate number should be 5. In Chapter (4) different values of k will be analyzed to find the best value that suits the train data set.

Subcase $1 \leq r \leq k$

If the 2-gram has been observed less than k times, its number of occurrences is smoothed according to the following formula:

$$\frac{\frac{r^*}{r} - \frac{(k+1)n_{k+1}}{n_1}}{1 - \frac{(k+1)n_{k+1}}{n_1}}$$

where n_k represents the number of the 2-grams that appear exactly k times. The value of r^* is equal to the Good-Turing frequency estimation [41], which is computed as follows:

$$r^* = (r+1)\frac{n_{r+1}}{n_r}$$

3.3.2 Case $r = 0$: Back-Off

In case the 2-gram has never been observed in the train set, it is necessary to find a method to assign a probability value different from 0. Katz proposes the so-called back-off strategy, which suggests to back-off and look at the weighted probability estimations in the immediately smaller order. In the case of a 2-gram, it consists of looking at the 1-gram weighted probability estimations. The 1-gram probability estimation is based on Maximum Likelihood Estimation (MLE), which corresponds to determining:

$$P(w_i) = \frac{C(w_i)}{\sum_{w_i} C(w_i)}$$

where C represents the usual counting function. The back-off weight is estimated according to:

$$\alpha(w_{i-1}) = \frac{1 - \sum_{w_i : C(w_{i-1}w_i)>0} P_{Katz}(w_i|w_{i-1})}{1 - \sum_{w_i : C(w_{i-1}w_i)>0} P(w_i)}$$

These two values are then combined to determine the value of $C_{Katz}(w_{i-1}, w_i) = \alpha(w_{i-1})P(w_i)$.

3.3.3 Building the Katz Back-Off 2-gram Language Model

For the creation of the Katz Back-off 2-gram LM the OpenFst [42] and OpenGrm [43] libraries have been exploited which results in an SFSM. A high-level representation of this model is shown in Figure (3.4). This 2-gram SFSM model consists of a set of states S and a set of transitions T between the states. The set of transitions T can be seen as $T = O \cup U$, i.e. the union between the observed transitions O and the unobserved transitions U. The set O corresponds to the observed 2-grams in the train set, while the set U must be derived using the smoothing and back-off techniques mentioned in Section (3.3). Transitions are weighted and the weights correspond to 2-gram probabilities. To guarantee that this model is a statistical one, it is necessary to ensure that the sum of the transitions from a start state w_{i-1} to each end state w_i, equals to 1. This corresponds to checking that:

$$\sum_{w_i \in V} P(w_i|w_{i-1}) = 1$$

The decoding of a sentence such as "<s> *i need to fly between philadelphia and atlanta* </s>" must use for each 2-gram, one of following transitions:

- $P(w_i|\texttt{<s>}) \wedge (\texttt{<s>}, w_i) \in O$: the set of observed transitions that have <s> as their start state, i.e. the start of sentence delimiter
- $P(w_i|\texttt{<s>}) \wedge (\texttt{<s>}, w_i) \in U$: the set of unobserved transitions that have <s> as their start state, i.e. the start of sentence delimiter
- $P(w_i|w_{i-1}) \wedge (w_{i-1}, w_i) \in O$: the set of observed transitions where the start and end states are different from delimiters <s> and </s>
- $P(w_i|w_{i-1}) \wedge (w_{i-1}, w_i) \in U$: the set of unobserved transitions where the start and end states are different from delimiters <s> and </s>
- $P(\texttt{</s>}|w_{i-1}) \wedge (w_{i-1}, \texttt{</s>}) \in O$: the set of observed transitions that have </s> as their end state, i.e. the end of sentence delimiter
- $P(\texttt{</s>}|w_{i-1}) \wedge (w_{i-1}, \texttt{</s>}) \in U$: the set of unobserved transitions that have </s> as their end state, i.e. the end of sentence delimiter

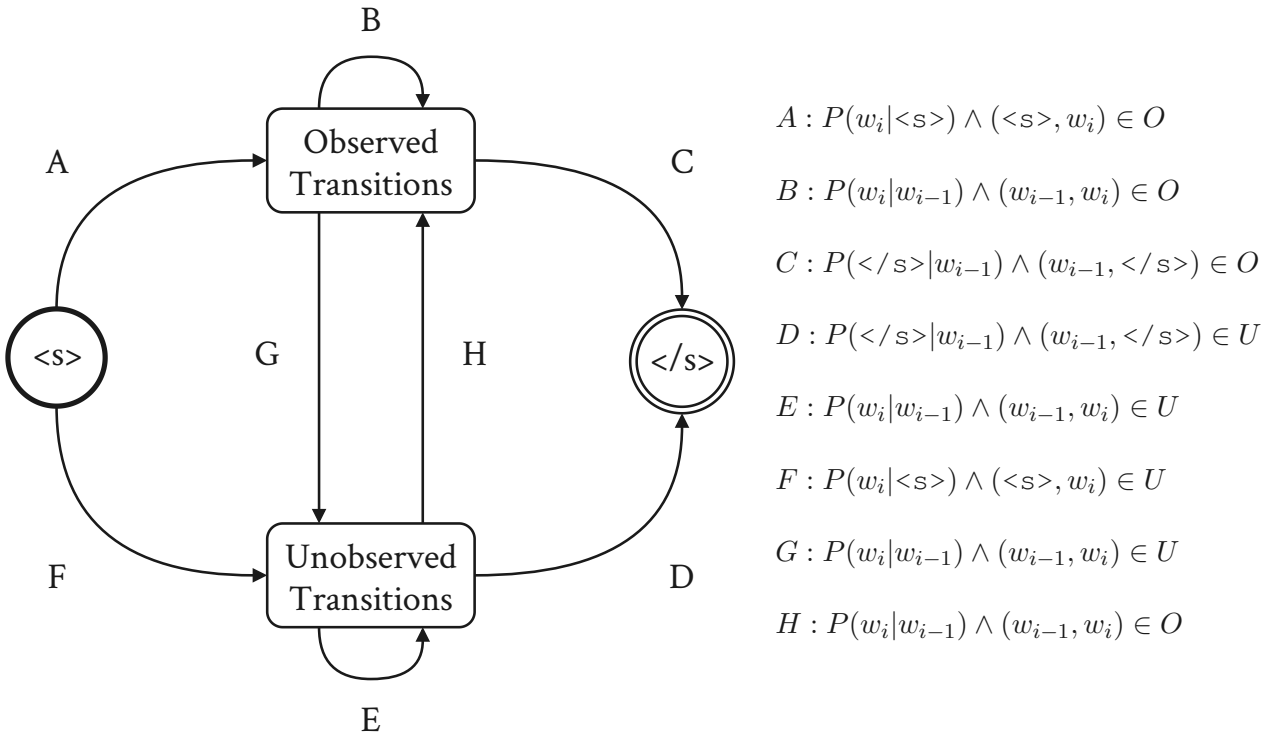
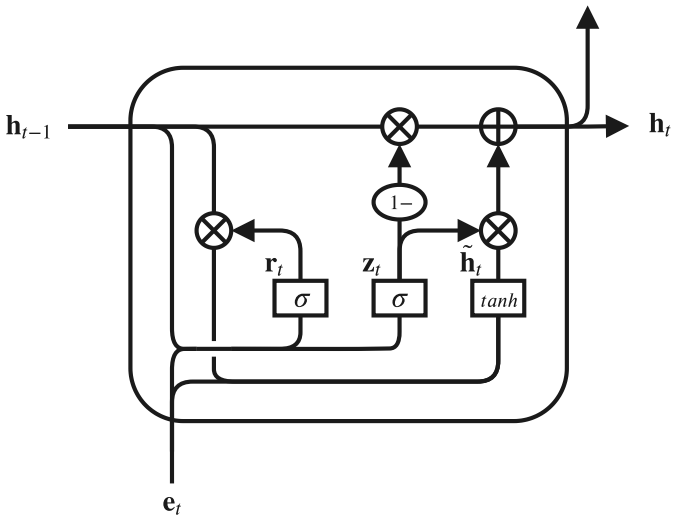


Figure 3.4: 2-gram SFSM.
States and transitions

3.4 Building the Neural Language Model

Subsection (1.3.2) introduced an NLM based on RNNs to solve the data sparsity problem. However, this model is not problem-free. Several studies have identified that this model suffers from issues related to vanishing [44] and exploding [45] gradients, which concern the way the weights of the Neural Network (NN) are updated and the gradients are propagated. Several RNNs architectures have been proposed to address these issues, such as the Long Short-Term Memory network [46] and the Gated Recurrent Unit (GRU) network [47]. This work focuses on the GRU variant, which is shown in Figure (3.5).



$$\mathbf{z}_t = \sigma(W_z \cdot [\mathbf{h}_{t-1}, \mathbf{e}_t])$$

$$\mathbf{r}_t = \sigma(W_r \cdot [\mathbf{h}_{t-1}, \mathbf{e}_t])$$

$$\tilde{\mathbf{h}}_t = tanh(W \cdot [\mathbf{r}_t * \mathbf{h}_{t-1}, \mathbf{e}_t])$$

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \tilde{\mathbf{h}}_t$$

Figure 3.5: Gated Recurrent Unit

The update of the hidden state $\mathbf{h}_t$ is done through a series of functions called gates. The first gate under analysis is called the *update* gate and works as follows:

$$\mathbf{z}_t = \sigma(W_z \cdot [\mathbf{h}_{t-1}, \mathbf{e}_t])$$

The concatenation of the previous hidden state $\mathbf{h}_{t-1}$ at time step $t-1$ and the embedding representation $\mathbf{e}_t$ of word w_t at time step t is linearly projected into a vector by means of the projection matrix W_z. This projection is then passed through the sigmoid non-linear activation function σ, which acts as follows:

$$\sigma(x) = \frac{1}{1 + e^{-x}} \in [0, 1]$$

The second gate under analysis is called the *reset* gate and works as follows:

$$\mathbf{r}_t = \sigma(W_r \cdot [\mathbf{h}_{t-1}, \mathbf{e}_t])$$

As can be noticed the $update$ gate and $reset$ gate have the same definition, except for the projection matrix each of them uses. The $reset$ gate plays a major role in the definition of the intermediate hidden state $\tilde{\mathbf{h}}_t$, which is computed as follows:

$$\tilde{\mathbf{h}}_t = tanh(W \cdot [\mathbf{r}_t * \mathbf{h}_{t-1}, \mathbf{e}_t])$$

The $\mathbf{r}_t$ vector contains values in range [0,1] and is multiplied element-wise with the previous hidden state $\mathbf{h}_{t-1}$. The $reset$ gate acts as a forgetting mechanism since elements of the $\mathbf{h}_{t-1}$ vector that are multiplied with values close to 0 from the $\mathbf{r}_t$ vector are to be considered less important, while those multiplied with values close to 1 will be preserved for further steps.

Finally, the GRU output consists of a new hidden state $\mathbf{h}_t$ at time step t, that is computed as follows:

$$\mathbf{h}_t = (1 - \mathbf{z}_t) * \mathbf{h}_{t-1} + \mathbf{z}_t * \tilde{\mathbf{h}}_t$$

The $output$ layer follows the same computation as the one defined in Subsection (1.3.2) for the RNN. In Chapter (4) different methods for training the E, W_z, W_r, W and O matrices will be outlined.

3.5 Language Models Evaluation Metric

As already mentioned, the objective of an SLM is to estimate the probability $P(W)$ of a sequence of N words $W = w_1, w_2, \ldots, w_N$. The sequence of words could be the test set, on which the quality of the model must be evaluated. However, instead of directly using the probabilities estimated by an SLM model as a measure of quality, in this work perplexity (PPL) is used, which is derived from information theory [7] and is defined as follows:

$$PPL(W) = 2^{H(Q,P)}$$

where $H(Q, P)$ represents the Cross-Entropy (CE) between distribution P and Q, where P is the probability model has learned from the train set, while Q represents the true probability distribution of words in the test set. CE is defined as follows:

$$H(Q, P) = -\sum_{w_i \in W} Q(w_i) log_2 P(w_i)$$

Since the true distribution Q of words in the test set is unknown, it can be approximated to $\frac{1}{|W|}$, that is $\frac{1}{N}$, which leads to the definition of PPL as follows:

$$PPL(W) = 2^{-\sum_{w_i \in W} \frac{1}{N} log_2 P(w_i)}$$

which can be rewritten as:

$$
\begin{aligned}
PPL(W) &= e^{-\sum_{w_i \in W} \frac{1}{N} lnP(w_i)} \\
&= e^{lnP(w_1)^{-\frac{1}{N}} + lnP(w_2)^{-\frac{1}{N}} + \cdots + lnP(w_N)^{-\frac{1}{N}}} \\
&= e^{ln(P(w_1)^{-\frac{1}{N}} \cdot P(w_2)^{-\frac{1}{N}} \cdots P(w_N)^{-\frac{1}{N}})} \\
&= P(w_1)^{-\frac{1}{N}} \cdot P(w_2)^{-\frac{1}{N}} \cdots P(w_N)^{-\frac{1}{N}} \\
&= \prod_{i=1}^{N} P(w_i)^{-\frac{1}{N}} \\
&= \sqrt[N]{\prod_{i=1}^{N} P(w_i)}
\end{aligned}
$$

and since there are dependencies between words, $PPL(W)$ can be rewritten as:

$$
PPL(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i|h_i)}} \qquad cosine(\mathbf{p}, \mathbf{q}) = 1 - \frac{\sum_i^n p_i q_i}{\sqrt{\sum_i^n p_i^2} \sqrt{\sum_i^n q_i^2}}
$$

cosine(**p**, **q**) = 1 - $\sum_i^n p_i q_i \frac{}{\sqrt{\sum_i^n p_i^2} \sqrt{\sum_i^n q_i^2}}$

where in case of a 2-gram **BLM**, $h_i = w_{i-1}$ and in the case of an **NLM** the creation of h_i is learned during the training phase. Perplexity, therefore, measures how well the P distribution learned during training can approximate the Q distribution of the test set.

It is important to note that the sequence of words W representing the test set, can be seen as the concatenation of test sentences where w_1 corresponds to the first word in the first sentence, while w_N corresponds to the last word in the last sentence. These two words correspond to <s> and </s>, respectively. It is evident that besides these two words, the start and end of the sentence delimiters will be encountered in different positions of the W sequence. In the perplexity computation it is, therefore, important not to count transitions from </s> to <s> where $w_{i-1} =$ </s> and $w_i =$ <s>.

Chapter 4

Experiments

This chapter will describe the experiments carried out with the two SLMs introduced in the previous sections. Section (4.1) and (4.2) will describe the training procedures adopted for the BLM and NLM, respectively. The following sections will describe different ways of building an SFSM from these two SLMs.

4.1 Back-Off 2-gram Language Model

As mentioned in Section (3.3), the OpenFst [42] and the OpenGrm [43] libraries were used for the construction of the 2-gram BLM. These libraries provide various smoothing methods, including Katz [19], Kneser-Ney [23] and Witten-Bell [20]. The results of these methods are reported in Table (4.1).

Smoothing Method	**PPL**
Katz	22.46
Kneser-Ney	21.41
Witten-Bell	22.26

Table 4.1: 2-gram BLMs performance comparison.
PPL values of Katz, Kneser-Ney, and Witten-Bell smoothing methods

Although there are no major differences between the results of the three smoothing methods, Kneser-Ney obtains the lowest PPL value. This work, however, focuses on the Katz smoothing method, as it is the default method used in the OpenGrm library. As already outlined in Section (3.3), the Katz method has a hyper-parameter k, which acts as a threshold on the number of times a 2-gram has been observed. The 2-gram that exceeds this threshold is to be considered more reliable compared to those that do not. For this purpose, a study on the threshold value k was conducted and is shown in Figure (4.1). The lowest PPL value obtained with this method is equal to 22.36.

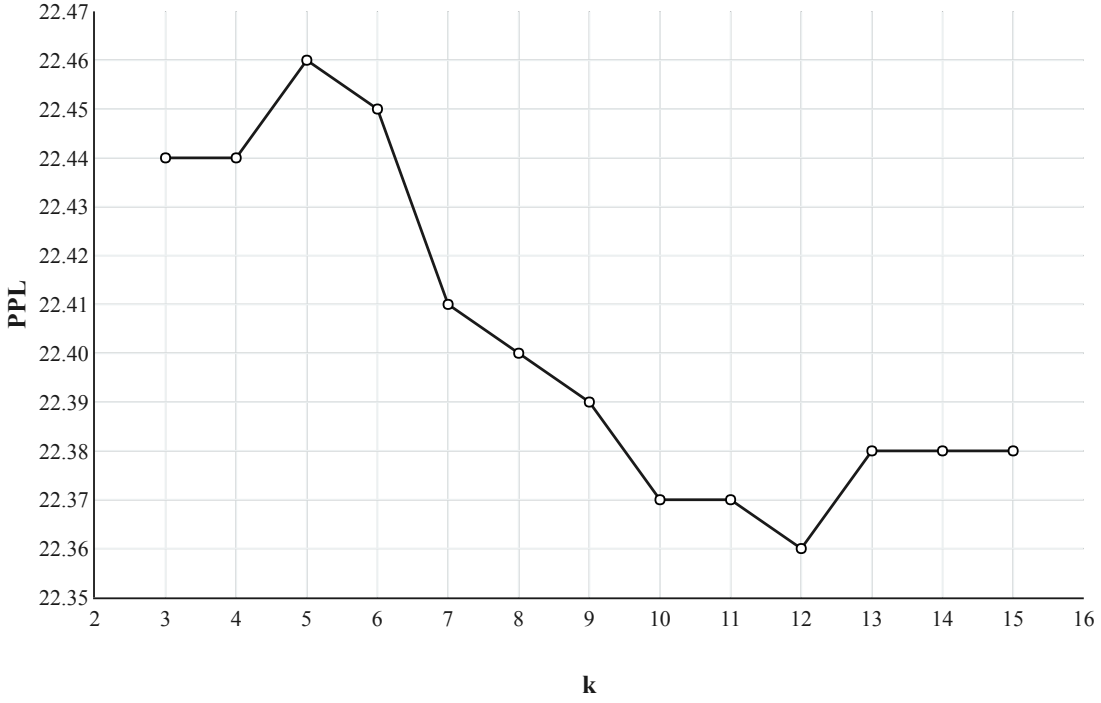


Figure 4.1: 2-gram Katz BLM.
Threshold value k vs PPL

The 2-gram Katz BLM can be seen as an SFSM, as shown in Figure (4.2). This representation follows the equations described in Section (4.1), where the set of observed transitions O contains the 2-gram transition probabilities defined as $P(w_i|w_{i-1})$, the set of unobserved transitions U contains the 2-gram transition probabilities defined as $P(\epsilon|w_{i-1}) \cdot P(w_i|\epsilon)$, where $P(\epsilon|w_{i-1})$ represents a back-off probability to an ϵ state, and $P(w_i|\epsilon)$ represents 1-gram probabilities.

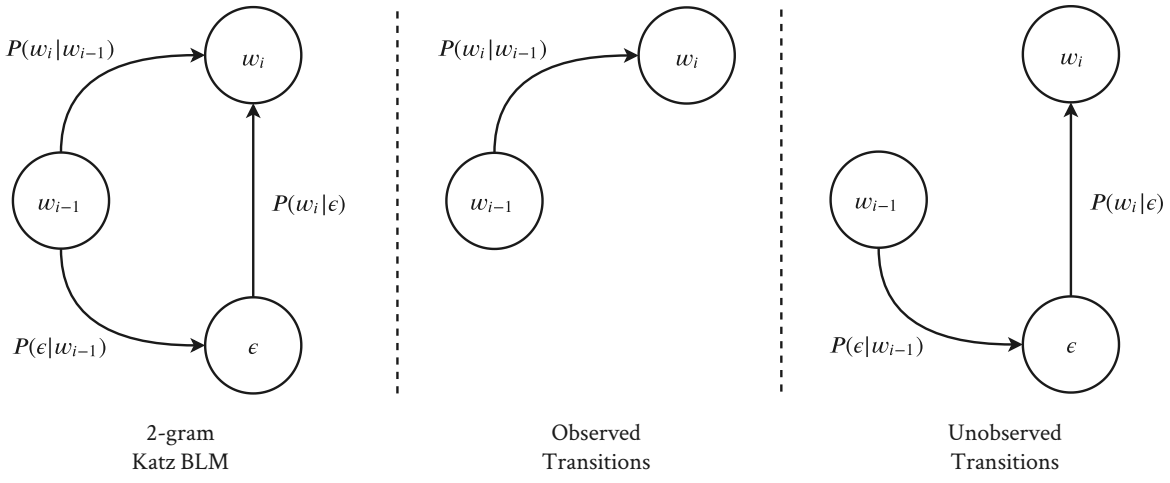


Figure 4.2: SFSM representation of a 2-gram Katz BLM

4.2 Neural Language Model

The construction and training of the NLM based on the GRU model presented in Section (3.4), was done through the PyTorch [48] library. The parameters of the model are reported in Table (4.2).

Parameter Name	Parameter Value
RANDOM_SEED	2910
TRAIN_DEV_SPLIT_PERCENTAGE	90%
PATIENCE	1
EMBEDDING_SIZE	100
HIDDEN_SIZE	200
CRITERION	Cross Entropy
OPTIMIZATION	AMSGrad
LEARNING_RATE	10^{-3}
TRAIN_BATCH_SIZE	16
EVAL_BATCH_SIZE	1
BACK_PROPAGATION_THROUGH_TIME	3

Table 4.2: NLM parameters

Parameter RANDOM_SEED is used as a seed initializer for the random number generator. It was decided to opt for a fixed value to ensure that the results are as reproducible as possible.

The second parameter TRAIN_DEV_SPLIT_PERCENTAGE is used to extract a data portion from the train set. This portion is called the development set and is used for the intermediate evaluation of the model performance before the final evaluation on the test set. A value of 90% corresponds to keeping 4480 sentences for the train set, while the remaining 10%, i.e. 498 sentences, are allocated to the development set for a total of 4978 sentences between train and development as reported previously in Table (3.2). The performance obtained on the development set is also used as a signal to decide the optimal stopping point of the training. The strategy governing the stop of the training is based on the so-called *Early Stopping* procedure. The procedure takes as input the PATIENCE parameter and together with the performance obtained on the development set, decides whether to stop the training or not. A value equal to 1 corresponds to stopping the training when the result on the development set has not improved after two evaluations.

The EMBEDDING_SIZE parameter, sets the dimension of the output vector $\mathbf{e}_t$ from the *embedding* phase. The HIDDEN_SIZE is analogous to the previous one but is related to the size of vector $\mathbf{h}_t$, i.e. the hidden state of the GRU cell. The CRITERION parameter sets as the objective function the Cross Entropy loss function. The OPTIMIZATION sets the stochastic optimization method to AMSGrad [49], while the LEARNING_RATE parameter controls the step of the gradient update.

The weights of the GRU module, i.e. $\mathbf{h}_t$, W, W_r and W_z are initialized with 0 values. Meanwhile, the weights of E and O matrices are initialized with values sampled from a uniform distribution over the range [-0.1, 0.1].

The last three parameters of the table will be discussed in the next subsection, which deals with the training and evaluation procedure of the NLM and how inconsistencies between those two phases can lead to different results in terms of PPL values.

4.2.1 Training and Evaluation of the NLM

It is well-known that in general NNs require a lot of data for training. The manipulation of massive amounts of data requires the efficient use of available hardware resources. Technological advancement and an ever-decreasing cost of Graphics Processing Units (GPUs) have made these devices the preferred medium for NNs training, as they reduce training time compared to Central Processing Units. An example of the use of GPUs in training large continuous space SLMs is the work done by [50], who used the so-called batch mode GPU training. This training method will be discussed in this subsection, and it will be compared with an additional less efficient training methodology. The aim is to show the differences that may exist between the training and evaluation phase, and how a mismatch between these two phases can lead to different results.

Batch Mode

Before training the NLM, the batch mode training procedure involves a further preprocessing phase in addition to the one already outlined in Subsection (3.2.2). The sentences in each data set are concatenated in a single vector. This corresponds to having three different vectors, one for train set, one for development set, and one for test set. Taking, for example, the vector of the train set, this will contain as first element the first word of the first sentence, i.e. <s>, and as last element, the last word of the last sentence, i.e. </s>. The vectors are then transformed into matrices, where the number of columns is decided according to parameters `TRAIN_BATCH_SIZE` and `EVAL_BATCH_SIZE` defined in Table (4.2), where the first parameter is applied to the train set vector, while the second is applied to the development and test vectors. The number of rows of the matrix is derived according to the length of each vector.

For instance, to roughly determine the size of the train set matrix, one can at first estimate the length of the train set vector, by taking the number of sentences reserved to the train set after the train-development split, i.e. a total of 4480 sentences, and multiplying it by the average sentence length in the train set, i.e. 12.48, thus reaching a total of $4480 \cdot 12.48 \approx 55910$ words. The start and end of sentence delimiters must be added to this value for each sentence, thus reaching $55910 + (4480 * 2) = 64870$ words. Once this value has been estimated, the number of rows of the batched version in the train set matrix can be determined by doing $64870/16 \approx 4054$ rows, where 16 corresponds to the `TRAIN_BATCH_SIZE` parameter value. Thus the final train set matrix has dimension 4054×16 for a total number of 64864 values/words. It must be noted that $64870 - 64864 = 6$ values are going to be discarded as they do not fill the number of columns of the matrix. The same procedure can be done with the development and test set matrices, which will have dimension 7211×1 and 11903×1, respectively where the value 1 corresponds to the `EVAL_BATCH_SIZE` parameter.

A synthetic example of a training set matrix is shown in Figure (4.3). This matrix is derived from a train set containing two sentences. The concatenation of the sentences leads to a vector containing 18 words. Assuming that parameter `TRAIN_BATCH_SIZE` has been set to 2, the train set matrix will then have dimension 9×2. The columns of this matrix are processed independently. Consequently, the dependency between the last word of the first column *"show"* and the first word of the second column *"me"* cannot be captured. If this dependency is not captured, the network captures the dependency between </s> and <s> which is unwanted. This is due to the fact that the train sentences have been concatenated, thus introducing the dependency between the end and start of sentence delimiters.

As shown in Figure (4.3), predictions are made for each row of the matrix. This means that the

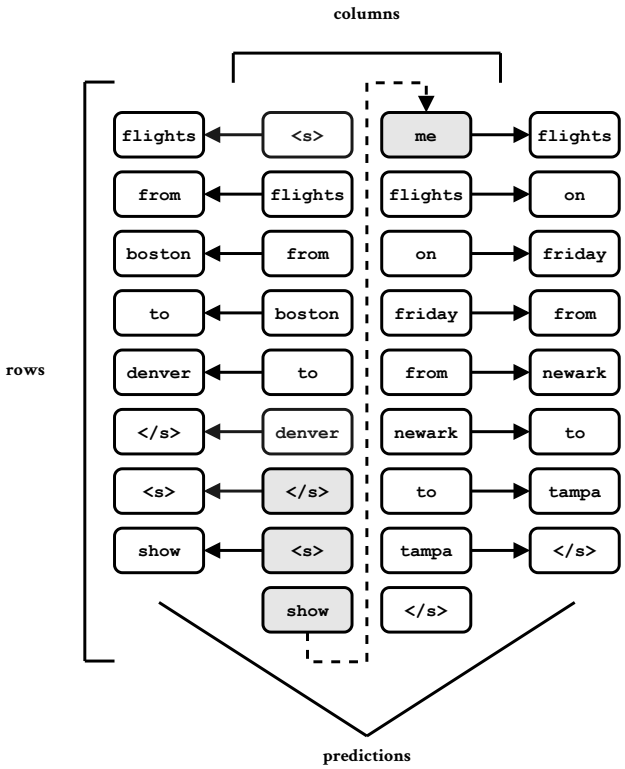


Figure 4.3: Synthetic example of a train set matrix

first row must predict the values of the second row, the second row must predict the values of the third row, and so on up to the second to last row of the matrix. Prediction errors are accumulated and back-propagated when the number of words that have been processed is equal to the `BACK_PROPAGATION_THROUGH_TIME` parameter.

As for the management of the hidden state $\mathbf{h}_t$, this vector is initialized only at the first row of the train matrix. The word sequences are processed until reaching the second to last row of the matrix, never resetting the hidden state. This implies that the model is not limited to capture dependencies only between words within a single sentence. It can also capture dependencies between words in different sentences. This corresponds to leveraging on cross-sentence dependencies for word predictions.

While this training method makes the best use of available hardware resources, it can introduce bias in the model evaluation phase. Given the way the NLM has been trained, during the evaluation phase, it expects a column vector containing a concatenation of sentences, be those related to the development set or the test set. Moreover, the hidden state $\mathbf{h}_t$ is initialized only at the beginning of the column vector. This is not compatible with the way the model will be used in a potential ASR system in a domain such as ATIS. Although the requests from different users can be represented as a concatenation of requests and consequently resembling the training conditions, the problem related to the management of the hidden state must be taken into account. Handling vectors of concatenated sentences larger than those of the train set one induces longer dependencies between words and sentences, that the model has not been trained for.

It is more simple to think of independent requests in the form of independent sentences from independent users. This corresponds to resetting the hidden state at the beginning of each request. However, a model trained in batch mode does not foresee the reset of the hidden state between one sentence and another, which introduces a mismatch between the way the model has been trained and the way

it will be used. The next paragraph will deal with introducing a training method that is less efficient but considers the several issues mentioned above.

Sentence Level

The sentence level training method is easier to understand since it involves considering each sentence in the set independent from the others. For each sentence, in fact, the hidden state $\mathbf{h}_t$ is initialized at the beginning of the sentence, and the words contained in the sentence are processed in sequence. The prediction error for each word is accumulated and back-propagated once the end of the sentence delimiter </s> is reached. Unlike the previous method, where the dependency between the last word in one column and the first word in the subsequent column could not be captured, this method is able to capture all dependencies between subsequent words. Moreover, this method does not capture the unwanted dependency between </s> and <s>. The last three parameters of Table (4.2) are therefore not used, as the training and evaluation conditions are identical. This model, therefore, complies with the ATIS domain use case, where requests are handled independently, without introducing dependencies between requests and especially between users.

Results: Batch Mode vs Sentence Level

Table (4.3) shows the results of the comparison between the two training methods mentioned above. This comparison also considers a mismatch between the training and evaluation phase. As outlined in Section (3.5), which deals with defining the procedure for the perplexity computation, it is necessary to exclude the $(\texttt{</s>}, \texttt{<s>})$ transition from the N count. This is particularly important in the case of the *GRU-batch-mode* model, which learns this transition dependency. The model is not penalized by the removal of the transition from the N count, as it still receives the token <s> in input, to be compliant with the training conditions. Only the probability that the model assigns to this specific transition is excluded. Besides being consistent with the perplexity definition, this way all the SLMs defined in this work can be compared with each other.

Model Name	Batch Model PPL	Sentence Level PPL
GRU-batch-mode	16.03	18.48
GRU-sentence-level	30.28	15.24

Table 4.3: NLM evaluation.
Batch Mode vs Sentence Level

It can be noticed that each model achieves its best result when the training and evaluation conditions are identical. This behavior has also been observed by the authors of [51] in a parallel work to the one carried out in this document. The *GRU-batch-mode* is the one that is least penalized by the mismatch as this model can exploit cross-sentence information during the training phase to outline better strategies for the construction of the hidden state $\mathbf{h}_t$. Consequently, when tested under conditions of isolated sentences, i.e. when the hidden state is reset at the beginning of each sentence, the model is still able to perform well.
Given the robustness of this model, it is interesting to understand whether the model needs all the previous words or a smaller number of words to construct the hidden state. For this purpose, Figure (4.4) shows the performance of this model when the hidden state is reset at discrete time steps.

This corresponds to limiting the context of words the model can process for the construction of the hidden state. As can be noticed, a limit of 50 previous words is enough to achieve almost the same performance as when all previous words are provided.

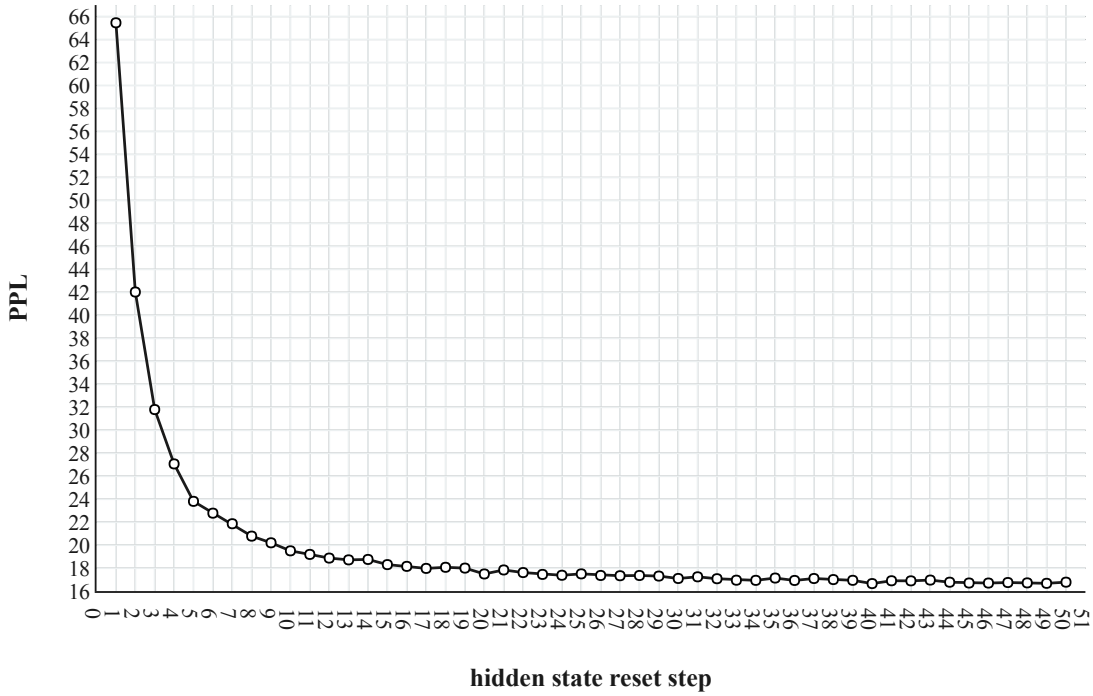


Figure 4.4: *GRU-batch-mode* test set evaluation.
Hidden state reset step vs PPL

Although Table (4.3) suggests that the *GRU-batch-mode* model is the one that suffered the least from the training-evaluation mismatch, the *GRU-sentence-level* model is the one that achieved the best result. Since this model is compatible with the ATIS domain use case, it will be used as the NLM reference model in later experiments and it will simply be indicated with *GRU*, thus omitting the way it was trained.

4.3 GRU-query

The NLM based on GRU can be seen as the composition of two components, namely the *Encoder* and the *Decoder*. The first component takes as input $\mathbf{h}_{t-1}$, i.e. the hidden state at time step $t-1$, together with w_t, i.e the word at time step t, and returns $\mathbf{h}_t$, i.e. the hidden state at time step t. This can be formalized as:

$$\mathbf{h}_t = Encoder(\mathbf{h}_{t-1}, w_t)$$

On the other side, the *Decoder* takes as input $\mathbf{h}_t$ and returns a probability distribution P over the words in the vocabulary V. This corresponds to:

$$P(w_{t+1}|\mathbf{h}_t) = Decoder(\mathbf{h}_t)$$

where $\sum_{w_i \in V} P(w_i|\mathbf{h}_t) = 1$. These two functions can be used for the first experiment to convert the GRU model into an SFSM. This model will consist of a set of states $S = V$, where the number of states is equal to $|V|$, i.e. the length of the vocabulary. The set of transitions $T = S \times S = V \times V$. As anticipated in Subsection (1.3.2), NLMs feature an implicit smoothing mechanism, as they work in continuous feature space representations. The *Decoder* is in fact able to return a probability distribution over the words in the vocabulary, regardless of how the hidden state has been built. The transition from one state to another represents the probability of a 2-gram. In principle, one could derive this probability through the following operation:

$$P(w_i|w_{i-1}) = Decoder(Encoder(\mathbf{h}_0, w_{i-1}))$$

where $\mathbf{h}_0$ represents the initial hidden state of the GRU model, prior to receiving any input word. As mentioned in Section (4.2) the $\mathbf{h}_0$ is initialized with 0 values. The procedure for the extraction of transition probabilities, i.e. 2-gram probabilities, can be seen as a sequence of queries submitted to the *GRU* model, hence the name *GRU-query* for the SFSM that is obtained as output from this procedure. The performance of this model is shown in Table (4.4), where it is also compared with the performance of previous models.

Model Name	**PPL**
Katz	22.36
GRU	15.24
GRU-query	44.86

Table 4.4: *GRU-query* performance comparison

As can be noticed, this SLM achieves much worse results than a traditional 2-gram Katz Back-Off Language Model (BLM). This suggests that the hidden state $\mathbf{h}_0$ plays a key role in estimating probabilities. This has been partly observed also in Section (4.2), where the hidden state for the *GRU-batch-mode* model was reset at different time steps. It is thus necessary to find better strategies for the extraction of probabilities from the *GRU* model. This will be investigated in the next sections.

4.4 GRU-contexts-probabilities

From the previous section it is evident that the *Decoder* component of the GRU is particularly susceptible to hidden state $\mathbf{h}_t$ vector. Recall that this vector can be seen as the history context of the words processed up to a specific time step t. The estimation of a 2-gram probability will depend on the context of the words taken into consideration prior to the estimation. However, since the ultimate goal consists of converting the *GRU* model into a static SFSM representation, it is necessary to provide a single probability estimation for each 2-gram. This implies several challenges, including the one depicted in Figure (4.5).

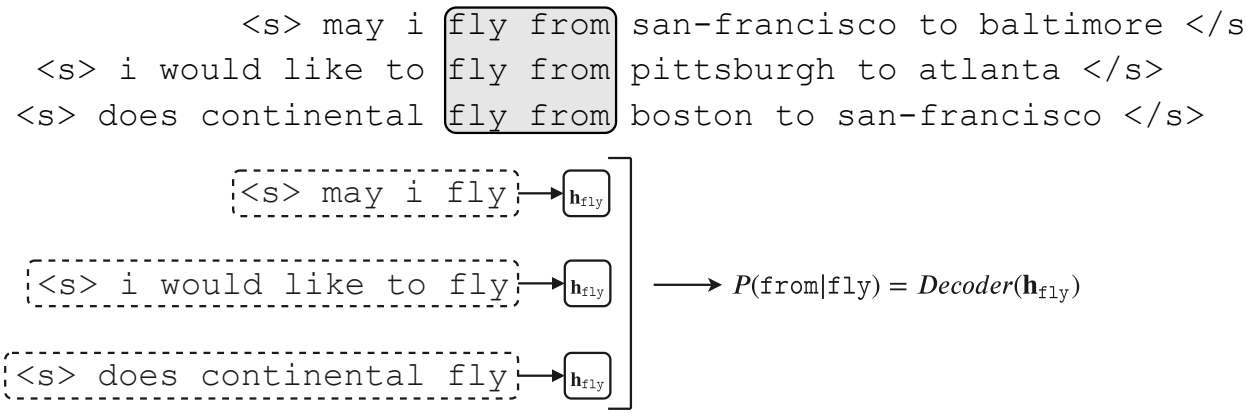


Figure 4.5: 2-gram probability estimation ambiguity.
Different $\mathbf{h}_{\texttt{fly}}$ can be used for estimating $P(\texttt{from}|\texttt{fly})$

Taking for example some sentences from the train set that contain the 2-gram $(\texttt{fly}, \texttt{from})$, the probability to be estimated is thus $P(\texttt{from}|\texttt{fly})$. This probability can be extracted from the *Decoder* since this component provides the probability distribution over the vocabulary V as follows:

$$P(w_i|\mathbf{h}_{\texttt{fly}}) = Decoder(\mathbf{h}_{\texttt{fly}})$$

where $w_i = \texttt{from}$. However there are multiple history contexts $\mathbf{h}_{\texttt{fly}}$ in the train set. This implies that there is an ambiguity about which history context to use for estimating the probability of the 2-gram. A methodology for this purpose is shown in Figure (4.6).

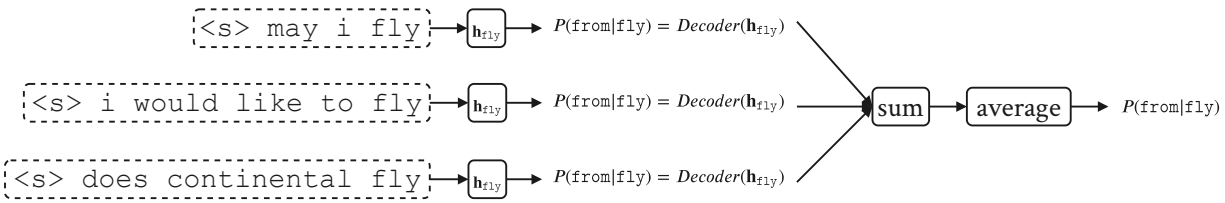


Figure 4.6: 2-gram probability estimation.
The sum and average of probability estimations method

This method takes into account all the different history contexts $\mathbf{h}_{\texttt{fly}}$ found in the train set for word `fly`, and for each of them retrieves $P(\texttt{from}|\texttt{fly})$ from the *Decoder*'s output. These probabilities are then summed and averaged. More formally $\forall w_i \in V$:

$$P(w_i|w_{i-1}) = \frac{\sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} P(w_i|\mathbf{h}_{w_{i-1}})}{|H_{w_{i-1}}|}$$

where $H_{w_{i-1}}$ represents the set of history contexts for word w_{i-1} and $|H_{w_{i-1}}|$ represents the length of this set, i.e. the number of sentence prefixes that contain word w_{i-1}. However, it is necessary to demonstrate that these probability estimations are actually legitimate probability estimations. For this purpose it is necessary to proof that:

$$\sum_{w_i \in V} P(w_i|w_{i-1}) = \sum_{w_i \in V} \frac{\sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} P(w_i|\mathbf{h}_{w_{i-1}})}{|H_{w_{i-1}}|}$$

that can be seen as:

$$1 = \sum_{w_i \in V} \frac{\sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} P(w_i|\mathbf{h}_{w_{i-1}})}{|H_{w_{i-1}}|}$$

and since the term $|H_{w_{i-1}}|$ at the denominator does not depend on the w_i in the summation term $\sum_{w_i} \in V$, it can be moved on the left side as follows:

$$1 = \frac{1}{|H_{w_{i-1}}|} \sum_{w_i \in V} \sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} P(w_i|\mathbf{h}_{w_{i-1}})$$

the same way the two summation terms can be swapped, thus reaching:

$$1 = \frac{1}{|H_{w_{i-1}}|} \sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} \sum_{w_i \in V} P(w_i|\mathbf{h}_{w_{i-1}})$$

which leads to:

$$1 = \frac{1}{|H_{w_{i-1}}|} \sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} 1$$

that can be seen as:

$$1 = \frac{1}{|H_{w_{i-1}}|} |H_{w_{i-1}}|$$

which results in:

$$1 = 1$$

This shows that the probability estimations are in fact legitimate probability estimations. The result obtained from this method are shown in Table (4.5), which reports also the results obtained by the previous models.

Model Name	**PPL**
Katz	22.36
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84

Table 4.5: *GRU-contexts-probabilities* performance comparison

As can be observed, better management of history contexts $\mathbf{h}_t$ leads to significant improvements over a simpler *GRU-query* model. This method succeeds in improving, even if only slightly, the results obtained by a traditional 2-gram Katz BLM. The results are still far from the *GRU* model, which suggests the need to explore further strategies in an attempt to reduce the performance gap.

4.5 GRU-contexts-centroids

This method partially replicates the *GRU-contexts-probabilities* model. The difference with the previous method is shown in Figure (4.7). As can be noticed, this method anticipates the estimation of the probability for each history context. Instead, it aggregates the different history contexts $\mathbf{h}_{\texttt{fly}}$ by calculating the so-called centroid $\tilde{\mathbf{h}}_{\texttt{fly}}$ and using the *Decoder* to retrieve the probability estimation $P(\texttt{from}|\texttt{fly})$.

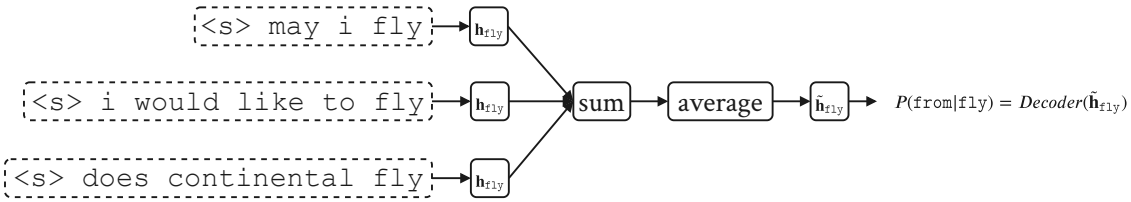


Figure 4.7: 2-gram probability estimation.
The sum and average of history contexts method

More formally this can be seen as computing the centroid $\tilde{\mathbf{h}}_{i-1}$ as follows:

$$\tilde{\mathbf{h}}_{w_{i-1}} = \frac{\sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} \mathbf{h}_{w_{i-1}}}{|H_{w_{i-1}}|}$$

This corresponds to the average of every history context for a specific word in vocabulary V. From a semantic point of view, $\tilde{\mathbf{h}}_{w_{i-1}}$ incorporates all the possible ways to reach the word w_{i-1} in the train set sentences in a single representation. This new history context can be used to extract the the 2-gram probability from the *Decoder* as follows:

$$P(w_i|w_{i-1}) = P(w_i|\tilde{\mathbf{h}}_{w_{i-1}}) = Decoder(\tilde{\mathbf{h}}_{w_{i-1}})$$

The result obtained from this method has been compared with those obtained by the previous models and is shown in Table (4.6). As it can be seen, this method obtains worse results compared to the previous *GRU-contexts-probabilities* model, which can be attributed to the fact that the *Decoder*, i.e. the NLM component that deals with extracting a probability distribution from a history context $\mathbf{h}_{w_{i-1}}$, has never encountered the new centroid representation $\tilde{\mathbf{h}}_{w_{i-1}}$. The model still manages to provide relatively good probability estimations since it works on continuous value vectors. However, the new centroid $\tilde{\mathbf{h}}_{w_{i-1}}$, in addition to containing the information of a specific history $\mathbf{h}_{w_{i-1}}$, it takes into account all the history contexts $\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}$ related to the word w_{i-1}. Next section will try to deal with this problem.

Model Name	PPL
Katz	22.36
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84
GRU-contexts-centroids	26.57

Table 4.6: *GRU-contexts-centroids* performance comparison

4.6 GRU-contexts-centroids-finetuned

The results obtained in the previous section highlight the difficulty for the *Decoder* in managing the new history contexts centroid vectors. Being $\tilde{\mathbf{h}}_t$ derived from multiple $\mathbf{h}_t$ vectors, implies that this new point in the feature space has not been taken into account during the training phase. The situation the *Decoder* has to deal with is shown in Figure (4.8).

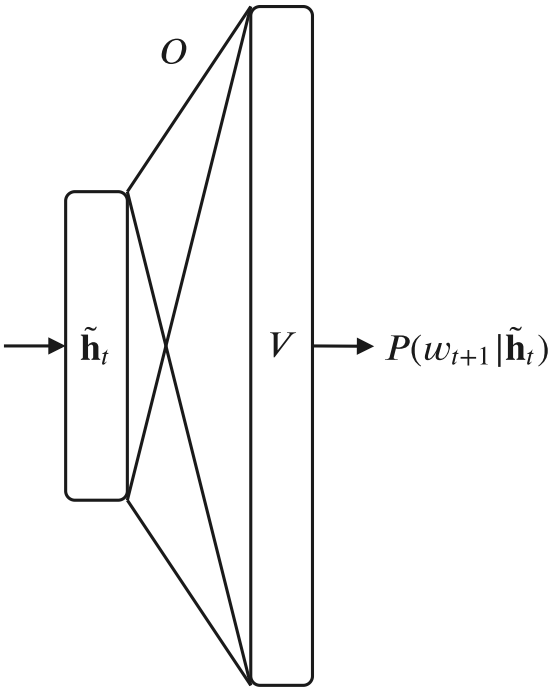


Figure 4.8: *Decoder* retrieves probability distribution P from history context centroid $\tilde{\mathbf{h}}_t$

As already reported in Section (4.2), a hidden state $\mathbf{h}_t$ is projected into a new vector via the O projection matrix. Then the *softmax* function is applied to obtain a probability distribution over the words of vocabulary V. To allow the *Decoder* to manage centroid $\tilde{\mathbf{h}}_t$ it is necessary to update the weights of the O matrix. This way, the model will learn to take into account the new points in the feature space. This procedure is called *fine-tuning*. To fine-tune this model, however, it is necessary to have train data. It is possible to generate synthetic train data by looking at the set of observed transitions within the train set. This set will contain 2-grams in the form of (w_{i-1}, w_i). The new synthetic training samples can be created as $(\tilde{\mathbf{h}}_{w_{i-1}}, w_i)$ tuples, where $\tilde{\mathbf{h}}_{w_{i-1}}$ is the usual centroid and acts as input to the *Decoder* component, while w_i is the target it has to predict. The total number of train samples obtained in this way is equal to 6929.

The parameters for fine-tuning the *Decoder* model are reported in Table (4.7) while the results obtained by the new model are reported in Table (4.8) together with the results obtained by the previous models.

Parameter Name	**Parameter Value**
RANDOM_SEED	2910
TRAIN_DEV_SPLIT_PERCENTAGE	90%
PATIENCE	1
CRITERION	Cross Entropy
OPTIMIZATION	AMSGrad
LEARNING_RATE	10^{-3}

Table 4.7: *Decoder* parameters

Model Name	**PPL**
Katz	22.36
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84
GRU-contexts-centroids	26.57
GRU-contexts-centroids-finetuned	24.91

Table 4.8: *GRU-contexts-centroids-finetuned* performance comparison

As can be noticed, the *fine-tuning* procedure has slightly improved the results of the previous *GRU-contexts-centroids* model. However, the results still leave room for further investigations.

4.7 Hybrid Back-Off Language Model

This section aims to combine the 2-gram Katz BLM with the GRU-based NLM, thus reaching a hybrid SFSM. The conversion of the GRU model into a BLM is not straightforward. The main problem is the estimation of the 1-gram probabilities. Recall that the output of the GRU model is always of the type $P(w_{i+1}|\mathbf{h}_t)$. There is, therefore, no estimation of the type $P(w_i)$ since the model always requires as input the current word w_t at time step t and the previous hidden state $\mathbf{h}_{t-1}$ at time step $t-1$. One possible way to convert the NLM model into an BLM is by using the structure of an already built BLM and attempt to re-estimate some of its probabilities. This section will present different strategies that aim to achieve this goal.

4.7.1 1-gram-Katz || 2-gram-GRU-contexts

As mentioned in Section (4.1) and shown in Figure (4.2), a 2-gram Katz BLM can be represented by a set of observed transitions O that contains 2-gram transition probabilities of type $P(w_i|w_{i-1})$, and a set of unobserved transitions U that contains 2-gram transition probabilities of type $P(\epsilon|w_{i-1}) \cdot P(w_i|\epsilon)$, where $P(\epsilon|w_{i-1})$ represents a back-off probability to an ϵ state, and $P(w_i|\epsilon)$ represents the 1-gram probabilities.

A conceivable hybrid model would consist of exploiting the 1-gram probabilities $P(w_i|\epsilon)$ provided by the Katz BLM model, and replacing probabilities $P(w_i|w_{i-1})$ and $P(\epsilon|w_{i-1})$ with those estimated by the GRU model. This corresponds to replacing the two probabilities estimations with $P_{GRU}(w_i|w_{i-1})$

and $P_{GRU}(\epsilon|w_{i-1})$, respectively. The estimation of $P_{GRU}(w_i|w_{i-1})$ can be done with the strategies defined in the previous sections, namely *GRU-contexts-probabilities*, *GRU-contexts-centroids*, and *GRU-contexts-centroids-finetuned*. Instead, the estimation of $P_{GRU}(\epsilon|w_{i-1})$ can be done as follows:

$$P_{GRU}(\epsilon|w_{i-1}) = 1 - \sum_{(w_{i-1},w_i)\in O} P_{GRU}(w_i|w_{i-1})$$

which corresponds to assigning to the ϵ state transition the probability mass that has not been covered by the set of observed transitions O. The results obtained with this strategy are shown in Table (4.9), together with the best results obtained by the previous models.

Model Name	**PPL**
Katz	22.36
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84
1-gram-Katz ǁ 2-gram-GRU-contexts-probabilities	24.20
1-gram-Katz ǁ 2-gram-GRU-contexts-centroids	29.16
1-gram-Katz ǁ 2-gram-GRU-contexts-centroids-finetuned	27.36

Table 4.9: *1-gram-Katz ǁ 2-gram-GRU-contexts* performance comparison

This hybrid model is not able to achieve better results compared to previous models. The next subsections will analyze whether the estimation of 1-grams from the Katz model affects the results negatively or not.

4.7.2 1-gram-uniform ǁ 2-gram-GRU-contexts

This subsection focuses on estimating the 1-gram transitions probability, leaving the GRU ϵ state transition and the 2-gram transitions part unchanged from the previous section. The aim is to understand to what extent the Katz 1-gram probability estimations affect the results. For this purpose, a relatively simple assumption is made, which consists of assigning the same probability value to each 1-gram. More formally, probability $P(w_i|\epsilon)$ corresponds to assigning a uniform probability value to each word in the vocabulary, which is estimated as follows:

$$P(w_i|\epsilon) = \frac{1}{|V|}$$

The results obtained with this strategy are shown in Table (4.10), together with the best results obtained by the previous models. As can be noticed, the use of a uniform probability as 1-gram probability estimation, produces slightly worse results. In addition, the same pattern is observed as in the previous block of experiments, with model *GRU-contexts-probabilities* as the leading one, followed by *GRU-contexts-centroids-finetuned* and *GRU-contexts-centroids*.

Model name	PPL
Katz	22.36
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84
1-gram-Katz ǁ 2-gram-GRU-contexts-probabilities	24.20
1-gram-uniform ǁ 2-gram-GRU-contexts-probabilities	24.69
1-gram-uniform ǁ 2-gram-GRU-contexts-centroids	29.75
1-gram-uniform ǁ 2-gram-GRU-contexts-centroids-finetuned	27.91

Table 4.10: *1-gram-uniform ǁ 2-gram-GRU-contexts* performance comparison

4.7.3 1-gram-GRU-<s> ǁ 2-gram-GRU-contexts

This subsection follows the structure of the previous subsection and focuses on a further strategy for the 1-gram probability estimation. This estimation is defined as follows:

$$P(w_i|\epsilon) = P_{GRU}(w_i|\mathbf{h}_{<s>}) = P_{GRU}(w_i|\tilde{\mathbf{h}}_{<s>})$$

The assumption is that the probability of a word w_i given the start of sentence delimiter <s> can be treated as a 1-gram probability since every sentence starts with the same delimiter. The GRU model thus learns the importance of a word w_i based on how often the word has been encountered in the train sentences after the start of sentence delimiter <s>. The results obtained from this strategy are shown in Table (4.11), together with the best results obtained by the previous models.

Model Name	PPL
Katz	22.36
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84
1-gram-Katz ǁ 2-gram-GRU-contexts-probabilities	24.20
1-gram-uniform ǁ 2-gram-GRU-contexts-probabilities	24.69
1-gram-GRU-<s> ǁ 2-gram-GRU-contexts-probabilities	29.63
1-gram-GRU-<s> ǁ 2-gram-GRU-contexts-centroids	35.70
1-gram-GRU-<s> ǁ 2-gram-GRU-contexts-centroids-finetuned	31.37

Table 4.11: *1-gram-GRU-<s> ǁ 2-gram-GRU-contexts* performance comparison

As can be observed, this method further worsens the results. This indicates that the creation of hybrid models does not seem to be a viable way. Next section will explore how the SFSM can be built from the GRU model from a completely different perspective.

4.8 GRU-KMeans

The models defined so far for the creation of an SFSM from a GRU have followed a top-down approach, where a state is defined a priori based on a guided aggregation of history context vectors, i.e. hidden states. In fact, $\forall w_i \in V$, set H_{w_i} has been been extracted, i.e. the set of sentence prefixes containing the word w_i. From this set, in Section (4.4), $\forall \mathbf{h}_{w_i} \in H_{w_i}$, the probabilities for the next word w_{i+1} were first extracted and the sum and average of these probabilities were made to define the final 2-gram probability. In Section (4.5) instead, these hidden states have been combined in a single representation $\tilde{\mathbf{h}}_{w_i}$, which has been used for the estimation of the final 2-gram probability. While this method makes it possible to map an SFSM state with a specific word w_i, this type of aggregation of the hidden states assumes that different sentence prefixes for the word w_i are similar, which is not necessarily true as different prefixes for the same word can have very different semantics.
This section focuses instead on the semantic representation of the hidden states, and proposes a bottom-up approach for their aggregation. Figure (4.9) shows the first step of this approach.

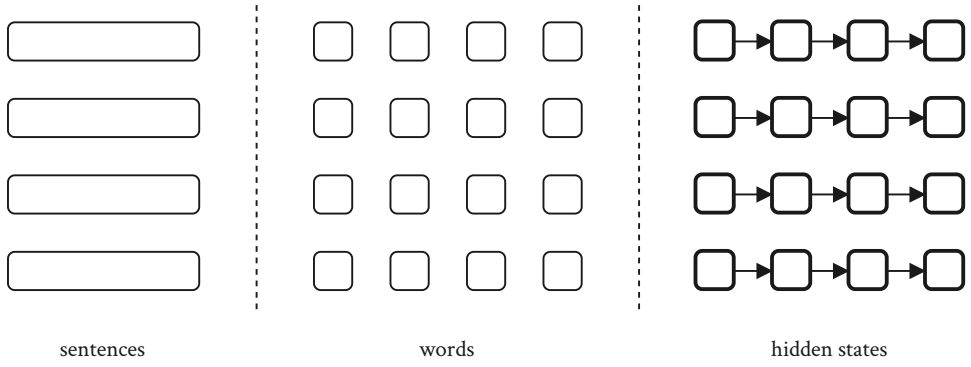


Figure 4.9: Sentences to hidden states conversion

The NLM model has been trained using a set of sentences contained in the text corpus. At the end of this process, the model is used to do a second pass on the words contained in each sentence of the data set to extract the hidden states relative to each word. The next step is shown in Figure (4.10) and consists of a clustering process which, given a finite number of k clusters defined a priori, maps each hidden state to one of the k clusters. This process takes advantage of the mini-batch version [52] of the k-means algorithm [53], implemented within the Scikit-learn [54] library. In addition to the hyper-parameter k, this algorithm requires the definition of two other hyper-parameters which are reported in Table (4.12).

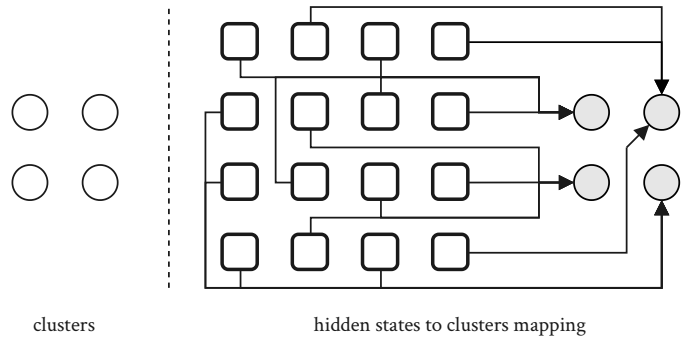


Figure 4.10: Hidden states clustering

Parameter Name	Parameter Value
RANDOM_STATE	2910
BATCH_SIZE	100

Table 4.12: Mini-batch k-means parameters

Each cluster k is identified by a centroid $\tilde{\mathbf{h}}_k \in K$, where K represents the set of centroids and each centroid is the hidden state that represents all the elements in the cluster. These centroids also represent the candidate states for the final SFSM. A new state $\tilde{\mathbf{h}}_{<s>}$ should be added to this set, representing the initial state, from which all the transitions to the other states must start. This state is defined as:

$$\tilde{\mathbf{h}}_{<s>} = Encoder(\mathbf{h}_0, <s>)$$

The back-off mechanism is included in the final SFSM by means of a new state $\tilde{\mathbf{h}}_\epsilon$, defined as:

$$\tilde{\mathbf{h}}_\epsilon = \frac{\sum_i^N \mathbf{h}_i}{|N|}$$

where N represents the total number of words in the train set, excluding the start and end of sentence delimiters. The ϵ state is thus represented by a hidden state computed as the global mean of the hidden states of the words in the train set.
As shown in Figure (4.11), once the K states has been created and the initial state $\tilde{\mathbf{h}}_{<s>}$ and the $\tilde{\mathbf{h}}_\epsilon$ state have been added, what remains for the creation of the final SFSM are the transitions between states.

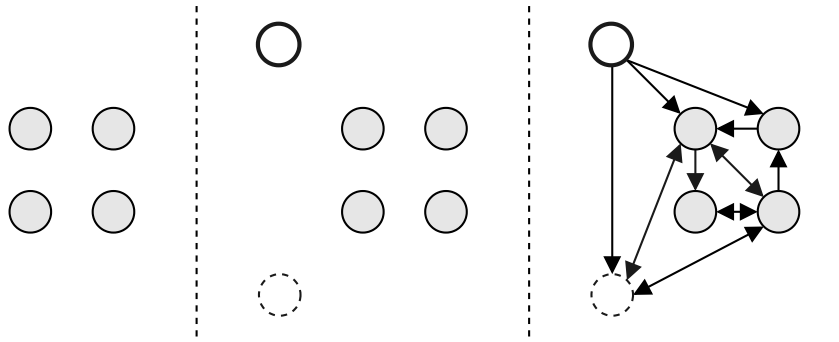


Figure 4.11: SFSM creation

The creation of those transitions is reported in Algorithm (1). The algorithm takes as input the set of observed transitions O, the set of centroids K and the two special <s> and ϵ states, represented by $\tilde{\mathbf{h}}_{<s>}$ and $\tilde{\mathbf{h}}_\epsilon$, respectively and outputs the SFSM with all the corresponding transitions between states. This algorithm is based on the incremental discovery of states, starting from the initial state $\tilde{\mathbf{h}}_{<s>}$. Each state $\tilde{\mathbf{h}}_i$ is linked to a word w_i with which the state was discovered. This word also determines the exit transitions from the state based on the observed transitions O that have the word w_i as the start state. The O set contains all the words w_{i+1} for the specific word w_i, that is all the transitions of type (w_i, w_{i+1}).

Data: O, K, $\tilde{\mathbf{h}}_{<s>}$, $\tilde{\mathbf{h}}_{\epsilon}$
Result: SFSM
SFSM = Dictionary();
queue = FIFO();
queue.put(($\tilde{\mathbf{h}}_{\epsilon}$, "epsilon"));
queue.put(($\tilde{\mathbf{h}}_{<s>}$, "<s>"));
discovered = List();
while *queue* ***not*** *empty* **do**
    $\tilde{\mathbf{h}}_{start}$, start = queue.get();
    P_{start} = $Decoder(\tilde{\mathbf{h}}_{start})$;
    P_O = 0.0;
    for *end* ***in*** *O[start]* **do**
        P_{end} = P_{start}[end];
        P_O += P_{end};
        $\tilde{\mathbf{h}}_{next}$ = *Encoder*($\tilde{\mathbf{h}}_{start}$, end);
        $\tilde{\mathbf{h}}_k$ = $distance\_minimum(\tilde{\mathbf{h}}_{next}, K)$;
        SFSM[($\tilde{\mathbf{h}}_{start}$, end)] = ($P_{end}$, $\tilde{\mathbf{h}}_k$);
        if *($\tilde{\mathbf{h}}_k$, end)* ***not in*** *discovered* **then**
            discovered.append(($\tilde{\mathbf{h}}_k$, end));
            queue.put(($\tilde{\mathbf{h}}_k$, end));
        end
    end
    P_ϵ = 1 - P_O;
    SFSM[($\tilde{\mathbf{h}}_{start}$, "epsilon")] = ($P_\epsilon$, $\tilde{\mathbf{h}}_\epsilon$);
end

Algorithm 1: Transitions between SFSM states

To decide the end state of a specific word w_{i+1} transition, the *Encoder* is used as follows:

$$\tilde{\mathbf{h}}_{next} = \textit{Encoder}(\tilde{\mathbf{h}}_i, w_{i+1})$$

This intermediate representation is compared via $distance\_minimum$ function with all centroids $\tilde{\mathbf{h}}_k$ contained in set K. The centroid $\tilde{\mathbf{h}}_k$ with the smallest distance is selected as the end state. If this state has not been discovered before with word w_{i+1}, it is added to the exploration First-In-First-Out (FIFO) *queue* along with word w_{i+1}. This corresponds to line *queue.put(($\tilde{\mathbf{h}}_k$, end))* in Algorithm (1), where *end* = w_{i+1}.
The probability of the transition (w_i, w_{i+1}) is obtained through the *Decoder* as follows:

$$P(w_{i+1}|w_i) = P(w_{i+1}|\tilde{\mathbf{h}}_i) = Decoder(\tilde{\mathbf{h}}_i)$$

It is important to note that this probability is dependent on state $\tilde{\mathbf{h}}_i$ and word w_i with which state $\tilde{\mathbf{h}}_i$ has been discovered. This means that $P(w_{i+1}|w_i) = P(w_{i+1}|\tilde{\mathbf{h}}_j) = Decoder(\tilde{\mathbf{h}}_j)$ where $\tilde{\mathbf{h}}_j \neq \tilde{\mathbf{h}}_i$ has been discovered with w_i, can also occur. This can be seen as different paths of reaching word w_i within the SFSM. Probability transitions $P(w_{i+1}|w_i)$ must be computed for each state that has been discovered with word w_i.

The probability mass that has not been covered by the observed transitions is therefore assigned to the ϵ transition as follows:

$$P(\epsilon|w_i) = 1 - \sum_{(w_i,w_{i+1})\in O} P(w_{i+1}|w_i)$$

The distance measure used in the $distance\_minimum$ function can be implemented using the *euclidean* distance, as follows:

$$euclidean(\mathbf{p},\mathbf{q}) = \sqrt{\sum_{i}^{n}(q_i - p_i)^2}$$

or by using the *cosine* distance, as follows:

$$cosine(\mathbf{p},\mathbf{q}) = 1 - \frac{\sum_i^n p_i q_i}{\sqrt{\sum_i^n p_i^2}\sqrt{\sum_i^n q_i^2}}$$

where $\mathbf{p}$ and $\mathbf{q}$ represent two centroids. Since the centroids once passed through the *Decoder* function encode a probability distribution over vocabulary V, it is possible to think of the distance measure as a measure of *divergence* between states. This corresponds to using the Kullback-Leibler divergence [55], defined as:

$$divergence(P,Q) = \sum_{x\in X} P(x) log(\frac{P(x)}{Q(x)})$$

where $P = Decoder(\mathbf{p})$ and $Q = Decoder(\mathbf{q})$. Even if this measure is not a proper distance measure since it is not symmetric, the reason it was adopted within this work is due to the fact that it takes into account the probability encoded by the hidden states and not their representation in the feature space. In Algorithm (1), $P = Decoder(\tilde{\mathbf{h}}_{next})$ and $Q = Decoder(\tilde{\mathbf{h}}_k)$. This function thus measures how much the probability distribution encoded by a centroid $\tilde{\mathbf{h}}_k$ diverges from the probability distribution encoded by the intermediate representation $\tilde{\mathbf{h}}_{next}$.

The creation of an SFSM based on k-means allows to manage the final dimension of the model and at the same time to trade-off between the number of states k and performance in terms of PPL value. A naïve algorithm for creating transitions involves creating a total number of $k \times V$ transitions. This corresponds to considering each centroid $\tilde{\mathbf{h}}_k$ as a start state, and for each word in the vocabulary decide the end state centroid. Functions $distance\_measure$ and $Decoder$ can be used for the centroid selection and probability estimation, respectively. It is clear that the number of transitions can easily become large. Recall that the back-off mechanism implemented in Katz's 2-gram BLM allows the number of transitions to be greatly limited, avoiding the creation of $V \times V$ transitions. In the same way, Algorithm (1) tries to limit the number of transitions by using the O set, which acts as a pruning mechanism, since for each word w_i contains the set of words w_{i+1} given that transition (w_i, w_{i+1}) has been observed in the train set. Moreover, given the way the algorithm has been defined, is not required for all k centroids to be discovered. There may be some very distant centroids which will not be discovered by any word transition.

The results obtained from this strategy are shown in Table (4.13), together with the best results obtained by the previous models.

Model Name	PPL
Katz	22.36
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84
1-gram-Katz ǁ 2-gram-GRU-contexts-probabilities	24.20
1-gram-uniform ǁ 2-gram-GRU-contexts-probabilities	24.69
1-gram-GRU-<s> ǁ 2-gram-GRU-contexts-probabilities	29.63
GRU-KMeans-k-250-euclidean	21.30
GRU-KMeans-k-500-euclidean	19.87
GRU-KMeans-k-750-euclidean	18.67
GRU-KMeans-k-1000-euclidean	18.87
GRU-KMeans-k-250-cosine	21.55
GRU-KMeans-k-500-cosine	20.10
GRU-KMeans-k-750-cosine	19.21
GRU-KMeans-k-1000-cosine	19.07
GRU-KMeans-k-250-divergence	20.87
GRU-KMeans-k-500-divergence	19.33
GRU-KMeans-k-750-divergence	18.62
GRU-KMeans-k-1000-divergence	19.38

Table 4.13: *GRU-KMeans* performance comparison

The strategy based on k-means proves to be the most effective, as it succeeds in considerably reducing the gap from the *GRU* model. Among the various distance measures adopted for the construction of transitions, the one based on *divergence* proves to be better than the classic *euclidean* and *cosine* methods. This means that the probability that a state encodes is more relevant than the hidden representation of the state itself. If on the one hand, this method succeeds in obtaining the best results, it is necessary to mark the fact that the final SFSM emerging from Algorithm (1), unlike previous strategies, loses the association of a word of the vocabulary to a specific state. In fact, states are created as an aggregation among hidden representations that are similar to each other, causing the loss of the label that identifies the state. It is therefore interesting to investigate to what extent the hidden states associated with each word contained in the train set are related to each other, and whether the method based on k-means can best aggregate them. The next section will try to provide more insight into this question.

4.9 GRU-dynamic

The main objective of this section is not directly related to the construction of an SFSM from a GRU. Instead, it aims to understand to what extent the hidden states relative to the words of the train set relate to each other and how they can be used to decode the sentences of the test set. The objective is, therefore, to understand if the method based on k-means presented in the previous section is able to aggregate in a proper way the hidden states of the train set. The decoding process of the test set in this case is dynamic, as it links the hidden states of the prefixes of the test sentences to a set of hidden states of the train set for the estimation of the 2-gram probabilities. In more formal terms, the

estimation of the probability is reduced to:

$$P(w_i|w_{i-1}) = \frac{P(w_i|\hat{\mathbf{h}}_{w_{i-1}}) + \sum_j^k P(w_i|\hat{X}_j)}{k+1}$$

where

$$\hat{X} = \{distance\_min(\hat{\mathbf{h}}_{w_{i-1}}, H_{w_{i-1}})\}_k$$

where $\hat{\mathbf{h}}_{w_{i-1}}$ represents the hidden state of a test sentence prefix up to word w_{i-1}, $\hat{X}$ represents the set of the k closest hidden states, with respect to $\hat{\mathbf{h}}_{w_{i-1}}$, that are part of the train hidden states set $H_{w_{i-1}}$ relative to same word w_{i-1}. In case $|H_{w_{i-1}}| < k$ all the hidden states contained in $H_{w_{i-1}}$ are taken into consideration. This procedure consists of aggregating a test set sentence prefix with the k most closely related sentence prefixes in the train set. However, as done in Section (4.4), it is necessary to demonstrate that these probability estimations are actually legitimate probability estimations. For this purpose it is necessary to proof that:

$$\sum_{w_i \in V} P(w_i|w_{i-1}) = \sum_{w_i \in V} \frac{P(w_i|\hat{\mathbf{h}}_{w_{i-1}}) + \sum_j^k P(w_i|\hat{X}_j)}{k+1}$$

which can be seen as:

$$1 = \frac{\sum_{w_i \in V} P(w_i|\hat{\mathbf{h}}_{w_{i-1}}) + \sum_{w_i \in V} \sum_j^k P(w_i|\hat{X}_j)}{k+1}$$

which reduces to:

$$1 = \frac{1 + \sum_{w_i \in V} \sum_j^k P(w_i|\hat{X}_j)}{k+1}$$

where the two summation terms can be swapped, thus leading to:

$$1 = \frac{1 + \sum_j^k \sum_{w_i \in V} P(w_i|\hat{X}_j)}{k+1}$$

which corresponds to:

$$1 = \frac{1 + \sum_j^k 1}{k+1}$$

that leads to:

$$1 = \frac{1+k}{k+1}$$

and finally to:

$$1 = 1$$

which shows that the probability estimations are in fact legitimate probability estimations. The result obtained from this method are shown in Table (4.14), which reports also the best results obtained by the previous models. The distance measures defined in the previous section, i.e. *euclidean*, *cosine*,

and *divergence* have been compared with a *random* selection of the k hidden states related for the $\hat{X}$ set. This was done to show that a random choice of hidden states is not enough, and that a choice based on distances gets the best results. Although the *random* choice does not get the best results, it is interesting to note that it manages to achieve competitive results when compared to *GRU-KMeans-k-750-divergence*.

Model Name	**PPL**
Katz	22.36
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84
1-gram-Katz ǁ 2-gram-GRU-contexts-probabilities	24.20
1-gram-uniform ǁ 2-gram-GRU-contexts-probabilities	24.69
1-gram-GRU-<s> ǁ 2-gram-GRU-contexts-probabilities	29.63
GRU-KMeans-k-750-divergence	18.62
GRU-dynamic-k-3-random	18.74
GRU-dynamic-k-5-random	19.64
GRU-dynamic-k-7-random	20.13
GRU-dynamic-k-3-euclidean	14.63
GRU-dynamic-k-5-euclidean	14.59
GRU-dynamic-k-7-euclidean	14.58
GRU-dynamic-k-3-cosine	14.65
GRU-dynamic-k-5-cosine	14.63
GRU-dynamic-k-7-cosine	14.63
GRU-dynamic-k-3-divergence	14.69
GRU-dynamic-k-5-divergence	14.63
GRU-dynamic-k-7-divergence	14.62

Table 4.14: *GRU-dynamic* performance comparison

Besides, the method based on dynamic decoding can achieve better results even when compared to the model based on the *GRU*, as the result of the fact that the *GRU-dynamic* model, when decoding a sentence, has access to external knowledge compared to *GRU*. This external knowledge is represented by the hidden states of the train set that are closer to the prefix of the sentence that is being decoded. The results obtained with the dynamic aggregation of the train hidden states, partially suggests that the k-means-based aggregation is not necessarily the best method.

To understand if the *GRU-dynamic* model is always able to get better results than the GRU model, it was decided to compare the two models under predictive conditions where the test sentences are not necessarily independent and identically distributed (i.i.d). To create these conditions, as the first step, train sentences have been clustered in such a way so that semantically similar sentences are mapped to the same cluster. However, to conduct the clustering procedure it is necessary to find a vector representation of the sentences. For this purpose the following procedure can be applied:

$$\mathbf{s}_j = \frac{\sum_{\mathbf{h}_{w_i} \in S_j} \mathbf{h}_{w_i}}{|S_j|}$$

where S_j represents the j-th sentence in the train set. The sentence is represented as a set of $|S_j|$ words w_i with their respective hidden representation $\mathbf{h}_{w_i}$. The vector $\mathbf{s}_j$ is thus computed as the average of the hidden representations of the words that compose sentence S_j. Once the vector representation for each sentence has been obtained, the train set can be represented as $T = \{\mathbf{s}_1, \mathbf{s}_2, \ldots, \mathbf{s}_N\}$. This set will then have to undergo the clustering process.

The choice of the number of clusters k in which to divide the train set has been made using the so-called Silhouette Score [56], which measures the degree of cohesion of an element with respect to the cluster to which it has been assigned compared to the other clusters. The output of the k-value selection via the Silhouette Score can be seen in Figure (4.12), where the value maximizing the score is equal to 52. For sentence clustering, the same algorithm of Section (4.8) has been used, together with the same initialization parameters reported in Table (4.12).

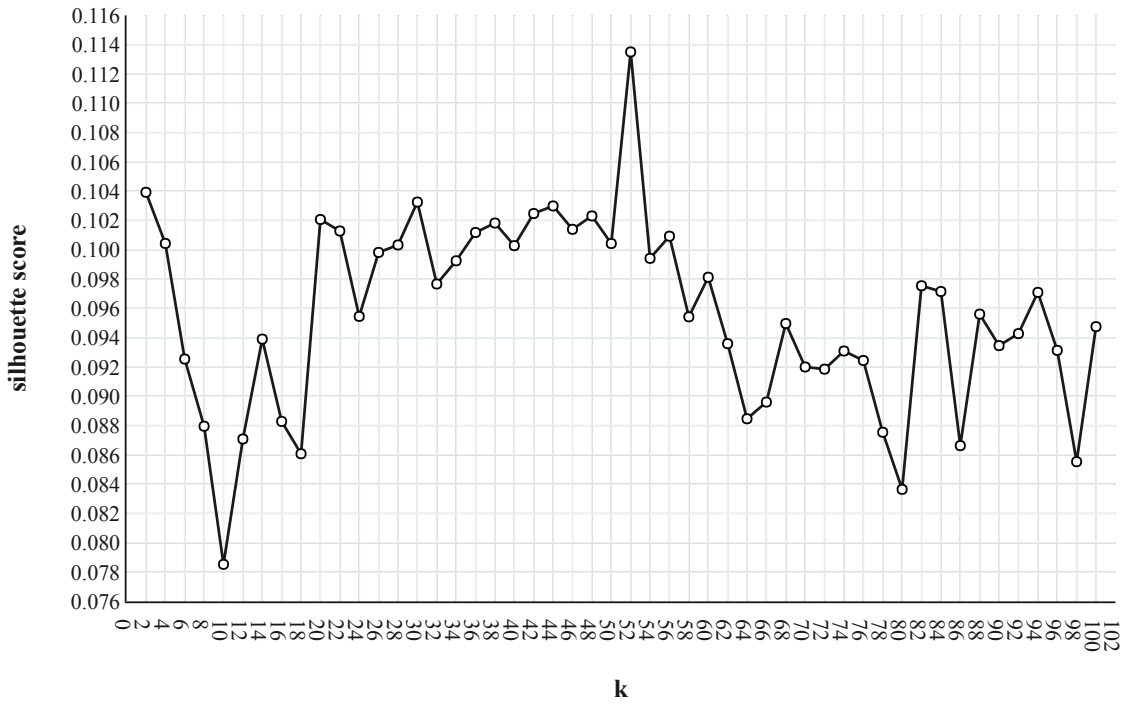


Figure 4.12: Train set sentences clustering.
k vs Silhouette Score

To verify the hypothesis according to which the *GRU-dynamic* model can obtain better results than the *GRU* model, even under non i.i.d. conditions, a k-fold cross-validation procedure was conducted. The k folds, in this case, correspond to the k clusters. This validation involves holding on rotation one fold as the test set and $k-1$ folds as the train and development sets. The same *GRU* model, i.e. the sentence-wise training model reported in the tables has been used for training on the $k-1$ folds. The dynamic model with which it has been compared to is its *GRU-dynamic-k-7-euclidean* derivation. The difference between the results obtained by the two models on the k-clusters, is shown in Figure (4.13). For reasons of readability, a maximum value of 4.25 has been set for the y-axis. Values related

to folds exceeding this threshold are instead reported in Table (4.15). It can be observed that, except for the fold 37 and 42, the *GRU-dynamic-k-7-euclidean* model manages to obtain better results than the *GRU* model.

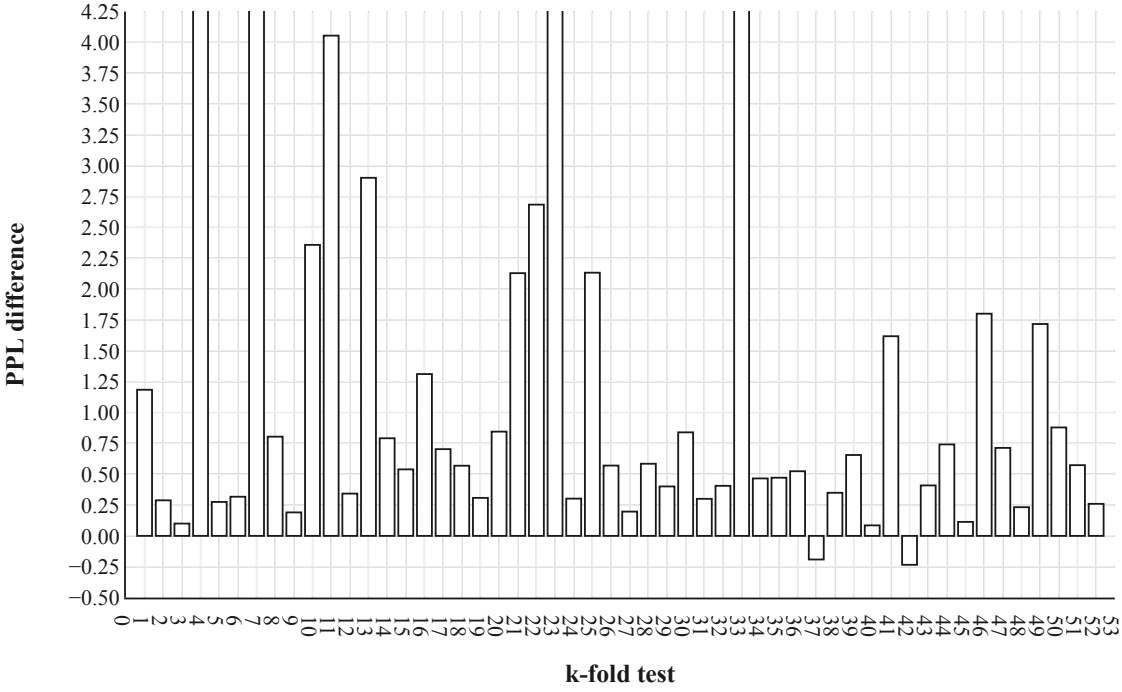


Figure 4.13: PPL difference.
GRU - *GRU-dynamic-k-7-euclidean*

Fold Number	PPL difference
4	7.01
7	6.98
23	35.53
33	230.76

Table 4.15: Out of scale PPL difference

Chapter 5

Conclusions

This work focused on the problem of incorporating a Neural Language Model (NLM) into an Automatic Speech Recognition (ASR) system where latency can be a crucial factor. One of the fundamental components of an ASR system is represented by the Statistical Language Model (SLM), which is often represented in the form of a Stochastic Finite State Model (SFSM), i.e. a finite state model where transitions between states are governed by probability values. Although in recent years the NLMs have proven to be superior to the traditional Back-Off N-gram-based Language Models (BLM), they have shown to be slower during the speech decoding phase. An NLM-based decoder has a larger search space than a BLM. This is due to the fact that NLMs can model longer history contexts, i.e. longer word sequences when compared to BLMs and consequently, the number of hypotheses to be taken into account during the decoding phase is much larger, which increases the processing time. To this end, the work carried out in this document aimed at reducing an NLM to an SFSM representation. This reduction represents a challenge since the number of history contexts that this model can encode is unbounded. Consequently, the reduction to a finite set of states represents only an approximation of the NLM.

This work exploited the Airline Travel Information System (ATIS) text corpus, and as the first step created an SFSM via a 2-gram BLM model, which implicitly encodes a finite state machine. The BLM model was considered as the baseline of comparison for subsequent experiments. Subsequently, the training of an NLM based on the Gated Recurrent Unit (GRU) was taken into consideration. It was shown how different ways of training and evaluation, and a mismatch between these two phases, can lead to different results. For the creation of an SFSM from the NLM model, firstly, a top-down strategy was proposed which involved taking into account all the history contexts related to a specific word, i.e. all the different prefixes of the sentences containing the word. Those contexts where then aggregated to form the representation of the word as a state of the SFSM and for the estimation of transition probabilities.

The second strategy involved a combination of the BLM and the NLM, where the first model was used as the structure of the final SFSM, while the second model was used to estimate the probabilities of transitions. The BLM model thus provided the set of states and transitions, while the NLM model estimated the probabilities of transitions.

The third strategy distinguished itself from the other two by proposing a bottom-up approach, where all the history contexts representing the words in the text corpus were considered on the same level, without defining an a priori strategy of how to aggregate them. Their combination has been left to a clustering procedure, which combined them based on their semantic similarity. The number

of clusters at the end of this process constituted the set of candidate states of the final SFSM. The connection between the states was determined by a states exploration algorithm based on the observed and unobserved transitions within the text corpus.
Finally, this work has also shown how in a dynamic text decoding context, it is possible to exploit the history contexts of the train set. This proved that this type of decoding can be useful even in conditions where sentences are not necessarily independent and identically distributed.
The results obtained from the various strategies are promising and establish a solid starting point for future research in this area. The next section will highlight several possible directions and strategies for future research.

5.1 Future Works

This work has focused mainly on 2-gram model languages, both for BLM and for the creation of the SFSM from the NLM. However, it would be interesting to extend this strategy to N-grams of a higher order. As Table (5.1) shows, BLMs with an order higher than 2 built with various smoothing strategies become competitive even when compared to a GRU-based NLM model.

Model Name	**PPL**
2-gram-Katz	22.36
3-gram-Katz	17.63
4-gram-Katz	18.19
5-gram-Katz	18.52
2-gram-Kneser-Ney	22.05
3-gram-Kneser-Ney	15.87
4-gram-Kneser-Ney	15.88
5-gram-Kneser-Ney	15.86
2-gram-Witten-Bell	22.26
3-gram-Witten-Bell	17.87
4-gram-Witten-Bell	20.04
5-gram-Witten-Bell	22.43
GRU	15.24
GRU-query	44.86
GRU-contexts-probabilities	21.84
1-gram-Katz ‖ 2-gram-GRU-contexts-probabilities	24.20
1-gram-uniform ‖ 2-gram-GRU-contexts-probabilities	24.69
1-gram-GRU-<s> ‖ 2-gram-GRU-contexts-probabilities	29.63
GRU-KMeans-k-750-divergence	18.62
GRU-dynamic-k-7-euclidean	14.58

Table 5.1: BLMs with order greater than 2.
Performance comparison

As can be noted, the *GRU-contexts-probabilities* model obtains better results than the 2-gram BLM alternatives. The equation described in Section (4.4), which governs the aggregation of probability estimates, could be extended and redefined as follows:

$$P(w_i|w_{i-1}) = \begin{cases} \frac{\sum_{\mathbf{h}_{(w_{i-1},w_i)} \in H_{(w_{i-1},w_i)}} P(w_i|\mathbf{h}_{(w_{i-1},w_i)})}{|H_{(w_{i-1},w_i)}|} & \text{if } |H_{(w_{i-1},w_i)}| \geq 1 \\ \frac{\sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} P(w_i|\mathbf{h}_{w_{i-1}})}{|H_{w_{i-1}}|} & \text{otherwise} \end{cases}$$

which instead of considering all the hidden states $H_{w_{i-1}}$ relative to word w_{i-1}, reduces the set of hidden states to $H_{(w_{i-1},w_i)}$, i.e. all hidden states relative to the sentence prefixes that contain the 2-gram (w_{i-1}, w_i). However, as done in Section (4.4), it is necessary to demonstrate that these probability estimations are actually legitimate probability estimations. For this purpose it is necessary to proof that:

$$\sum_{w_i \in V} P(w_i|w_{i-1}) = \sum_{w_i \in V} \begin{cases} \frac{\sum_{\mathbf{h}_{(w_{i-1},w_i)} \in H_{(w_{i-1},w_i)}} P(w_i|\mathbf{h}_{(w_{i-1},w_i)})}{|H_{(w_{i-1},w_i)}|} & \text{if } |H_{(w_{i-1},w_i)}| \geq 1 \\ \frac{\sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} P(w_i|\mathbf{h}_{w_{i-1}})}{|H_{w_{i-1}}|} & \text{otherwise} \end{cases} \quad t$$

which can be seen as:

$$1 = \begin{cases} \sum_{w_i \in V} \frac{\sum_{\mathbf{h}_{(w_{i-1},w_i)} \in H_{(w_{i-1},w_i)}} P(w_i|\mathbf{h}_{(w_{i-1},w_i)})}{|H_{(w_{i-1},w_i)}|} & \text{if } |H_{(w_{i-1},w_i)}| \geq 1 \\ \sum_{w_i \in V} \frac{\sum_{\mathbf{h}_{w_{i-1}} \in H_{w_{i-1}}} P(w_i|\mathbf{h}_{w_{i-1}})}{|H_{w_{i-1}}|} & \text{otherwise} \end{cases}$$

The equation in the *otherwise* branch can be proven in the same way as already done in Section (4.4). This results in:

$$1 = \begin{cases} \sum_{w_i \in V} \frac{\sum_{\mathbf{h}_{(w_{i-1},w_i)} \in H_{(w_{i-1},w_i)}} P(w_i|\mathbf{h}_{(w_{i-1},w_i)})}{|H_{(w_{i-1},w_i)}|} & \text{if } |H_{(w_{i-1},w_i)}| \geq 1 \\ 1 & \text{otherwise} \end{cases}$$

However, the same thing cannot be done for the first equation, as there is a dependency on word w_i. This equation needs further work to make sure the probabilities add up to 1.

A further way to improve the results could be done by fine-tuning the *GRU-Kmeans* model as done for the *GRU-contexts-centroids-finetuned* model. Each word w_i in the train set could be associated with its hidden state $\mathbf{h}_i$ to one of the k cluster centroids $\boldsymbol{h}_k$ and the next word w_{i+1} could then be used to fine-tune the *Decoder* as done in Section (4.6).

Bibliography

[1] Rob Kitchin. *The data revolution: Big data, open data, data infrastructures and their consequences*. Sage, 2014.

[2] Homer Dudley and Thomas H Tarnoczy. *The speaking machine of Wolfgang von Kempelen*. In: "The Journal of the Acoustical Society of America" 22.2 (1950), pp. 151–166.

[3] Noam Chomsky. *Syntactic structures*. Walter de Gruyter, 2002.

[4] Karin Müller. *Probabilistic Context-Free Grammars for Phonology*. In: "Proceedings of the ACL-02 Workshop on Morphological and Phonological Learning". 2002, pp. 70–80.

[5] Christopher D Manning, Christopher D Manning, and Hinrich Schütze. *Foundations of statistical natural language processing*. 1999.

[6] Fritz W Scholz. *Maximum likelihood estimation*. In: "Encyclopedia of statistical sciences" (2004).

[7] Dan Jurafsky. *Speech & language processing*. Pearson Education India, 2000.

[8] Andrej A Markov. *Theory of algorithms*. In: Academy of Sciences of the USSR Moscow. 1954.

[9] Venkat N Gudivada, Dhana Rao, and Vijay V Raghavan. "Big data driven natural language processing research and applications". In: *Handbook of Statistics*. Vol. 33. Elsevier, 2015, pp. 203–238.

[10] Joshua Goodman. *A bit of progress in language modeling*. In: "arXiv preprint cs/0108005" (2001).

[11] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Černock, and Sanjeev Khudanpur. *Recurrent neural network based language model*. In: "Eleventh annual conference of the international speech communication association". 2010.

[12] Jeffrey L Elman. *Finding structure in time*. In: "Cognitive science" 14.2 (1990), pp. 179–211.

[13] Reuven Y Rubinstein and Dirk P Kroese. *The cross-entropy method: a unified approach to combinatorial optimization, Monte-Carlo simulation and machine learning*. Springer Science & Business Media, 2013.

[14] Diederik P Kingma and Jimmy Ba. *Adam: A method for stochastic optimization*. In: "arXiv preprint arXiv:1412.6980" (2014).

[15] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. *Learning representations by back-propagating errors*. In: "nature" 323.6088 (1986), pp. 533–536.

[16] Peter F Brown, John Cocke, Stephen A Della Pietra, Vincent J Della Pietra, Frederick Jelinek, John Lafferty, Robert L Mercer, and Paul S Roossin. *A statistical approach to machine translation*. In: "Computational linguistics" 16.2 (1990), pp. 79–85.

[17] Kenneth Ward Church. *A stochastic parts program and noun phrase parser for unrestricted text*. In: "International Conference on Acoustics, Speech, and Signal Processing," IEEE. 1989, pp. 695–698.

[18] Lalit R Bahl, Frederick Jelinek, and Robert L Mercer. *A maximum likelihood approach to continuous speech recognition*. In: "IEEE transactions on pattern analysis and machine intelligence" 2 (1983), pp. 179–190.

[19] Slava Katz. *Estimation of probabilities from sparse data for the language model component of a speech recognizer*. In: "IEEE transactions on acoustics, speech, and signal processing" 35.3 (1987), pp. 400–401.

[20] Timothy Bell, Ian H Witten, and John G Cleary. *Modeling for text compression*. In: "ACM Computing Surveys (CSUR)" 21.4 (1989), pp. 557–591.

[21] Frederick Jelinek. *Interpolated estimation of Markov source parameters from sparse data*. In: 1980.

[22] Kenneth W Church and William A Gale. *A comparison of the enhanced Good-Turing and deleted estimation methods for estimating probabilities of English bigrams*. In: "Computer Speech & Language" 5.1 (1991), pp. 19–54.

[23] Hermann Ney, Ute Essen, and Reinhard Kneser. *On structuring probabilistic dependences in stochastic language modelling*. In: "Computer Speech & Language" 8.1 (1994), pp. 1–38.

[24] Stanley F Chen and Joshua Goodman. *An empirical study of smoothing techniques for language modeling*. In: "Computer Speech & Language" 13.4 (1999), pp. 359–394.

[25] Wenyang Zhang. *Comparing the effect of smoothing and n-gram order: Finding the best way to combine the smoothing and order of n-gram*. PhD thesis. 2015.

[26] Giuseppe Riccardi, Roberto Pieraccini, and Enrico Bocchieri. *Stochastic automata for language modeling*. In: "Computer Speech & Language" 10.4 (1996), pp. 265–293.

[27] Yoshua Bengio, Réjean Ducharme, Pascal Vincent, and Christian Jauvin. *A neural probabilistic language model*. In: "Journal of machine learning research" 3.Feb (2003), pp. 1137–1155.

[28] Anoop Deoras, Tomáš Mikolov, Stefan Kombrink, Martin Karafiát, and Sanjeev Khudanpur. *Variational approximation of long-span language models for LVCSR*. In: "2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)". IEEE. 2011, pp. 5532–5535.

[29] Alan E Gelfand and Adrian FM Smith. *Sampling-based approaches to calculating marginal densities*. In: "Journal of the American statistical association" 85.410 (1990), pp. 398–409.

[30] Anirudh Raju, Denis Filimonov, Gautam Tiwari, Guitang Lan, and Ariya Rastrow. *Scalable multi corpora neural language models for asr*. In: "arXiv preprint arXiv:1907.01677" (2019).

[31] Michael Gutmann and Aapo Hyvärinen. *Noise-contrastive estimation: A new estimation principle for unnormalized statistical models*. In: "Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics". 2010, pp. 297–304.

[32] Ebru Arısoy, Stanley F Chen, Bhuvana Ramabhadran, and Abhinav Sethy. *Converting neural network language models into back-off language models for efficient decoding in automatic speech recognition*. In: "IEEE/ACM Transactions on Audio, Speech, and Language Processing" 22.1 (2013), pp. 184–192.

[33] Gwénolé Lecorvé and Petr Motlicek. *Conversion of recurrent neural network language models to weighted finite state transducers for automatic speech recognition*. In: "Thirteenth Annual Conference of the International Speech Communication Association". 2012.

[34] Roy Schwartz, Sam Thomson, and Noah A Smith. *SoPa: Bridging CNNs, RNNs, and weighted finite-state machines*. In: "arXiv preprint arXiv:1805.06061" (2018).

[35] Guillaume Rabusseau, Tianyu Li, and Doina Precup. *Connecting weighted automata and recurrent neural networks through spectral learning*. In: "arXiv preprint arXiv:1807.01406" (2018).

[36] Stéphane Ayache, Rémi Eyraud, and Noé Goudian. *Explaining black boxes on sequential data using weighted automata*. In: "arXiv preprint arXiv:1810.05741" (2018).

[37] Gene H Golub and Christian Reinsch. “Singular value decomposition and least squares solutions”. In: *Linear Algebra*. Springer, 1971, pp. 134–151.

[38] Christian Raymond and Giuseppe Riccardi. *Generative and discriminative algorithms for spoken language understanding*. In: “Eighth Annual Conference of the International Speech Communication Association”. 2007.

[39] Edward Loper and Steven Bird. *NLTK: the natural language toolkit*. In: “arXiv preprint cs/0205028” (2002).

[40] Steven T Piantadosi. *Zipf’s word frequency law in natural language: A critical review and future directions*. In: “Psychonomic bulletin & review” 21.5 (2014), pp. 1112–1130.

[41] Irving J Good. *The population frequencies of species and the estimation of population parameters*. In: “Biometrika” 40.3-4 (1953), pp. 237–264.

[42] Cyril Allauzen, Michael Riley, Johan Schalkwyk, Wojciech Skut, and Mehryar Mohri. *OpenFst: A general and efficient weighted finite-state transducer library*. In: “International Conference on Implementation and Application of Automata”. Springer. 2007, pp. 11–23.

[43] Brian Roark, Richard Sproat, Cyril Allauzen, Michael Riley, Jeffrey Sorensen, and Terry Tai. *The OpenGrm open-source finite-state grammar software libraries*. In: “Proceedings of the ACL 2012 System Demonstrations”. 2012, pp. 61–66.

[44] Sepp Hochreiter. *The vanishing gradient problem during learning recurrent neural nets and problem solutions*. In: “International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems” 6.02 (1998), pp. 107–116.

[45] Xavier Glorot and Yoshua Bengio. *Understanding the difficulty of training deep feedforward neural networks*. In: “Proceedings of the thirteenth international conference on artificial intelligence and statistics”. 2010, pp. 249–256.

[46] Sepp Hochreiter and Jürgen Schmidhuber. *Long short-term memory*. In: “Neural computation” 9.8 (1997), pp. 1735–1780.

[47] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. *Learning phrase representations using RNN encoder-decoder for statistical machine translation*. In: “arXiv preprint arXiv:1406.1078” (2014).

[48] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. *PyTorch: An imperative style, high-performance deep learning library*. In: “Advances in Neural Information Processing Systems”. 2019, pp. 8024–8035.

[49] Sashank J Reddi, Satyen Kale, and Sanjiv Kumar. *On the convergence of adam and beyond*. In: “arXiv preprint arXiv:1904.09237” (2019).

[50] Holger Schwenk, Anthony Rousseau, and Mohammed Attik. *Large, pruned or continuous space language models on a gpu for statistical machine translation*. In: “Proceedings of the NAACL-HLT 2012 Workshop: Will We Ever Really Replace the N-gram Model? On the Future of Language Modeling for HLT”. Association for Computational Linguistics. 2012, pp. 11–19.

[51] Kazuki Irie, Albert Zeyer, Ralf Schlüter, and Hermann Ney. *Training Language Models for Long-Span Cross-Sentence Evaluation*. In: “2019 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)”. IEEE. 2019, pp. 419–426.

[52] David Sculley. *Web-scale k-means clustering*. In: “Proceedings of the 19th international conference on World wide web”. 2010, pp. 1177–1178.

[53] James MacQueen et al. *Some methods for classification and analysis of multivariate observations*. In: “Proceedings of the fifth Berkeley symposium on mathematical statistics and probability”. Vol. 1. 14. Oakland, CA, USA. 1967, pp. 281–297.

[54] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. *Scikit-learn: Machine learning in Python*. In: “Journal of machine learning research” 12.Oct (2011), pp. 2825–2830.

[55] Solomon Kullback. *Information theory and statistics*. Courier Corporation, 1997.

[56] Peter J Rousseeuw. *Silhouettes: a graphical aid to the interpretation and validation of cluster analysis*. In: “Journal of computational and applied mathematics” 20 (1987), pp. 53–65.

Acronyms

ASR Automatic Speech Recognition. 7, 10–12, 15–17, 20, 23, 37, 57

ATIS Airline Travel Information System. 7, 17, 24–26, 37–39, 57

BLM Back-Off Language Model. 7, 10, 11, 17, 19–21, 23, 32–34, 40, 43, 45, 51, 57–59

CE Cross Entropy. 15, 31

CFG Context-Free Grammar. 11

FFNN Feed Forward Neural Network. 20, 21

FIFO First In First Out. 50

GPU Graphics Processing Unit. 36

GRU Gated Recurrent Unit. 3, 5, 30, 31, 35, 38–48, 52, 54–59

LM Language Model. 10, 11, 13–17, 19, 20, 28

MLE Maximum Likelihood Estimation. 12, 13, 28

NLM Neural Language Model. 5, 7, 11, 15–17, 20, 21, 23, 30, 32, 33, 35–40, 43, 45, 48, 57, 58

NN Neural Network. 10, 14, 17, 19, 30, 36

OOV Out-Of-Vocabulary. 24, 26

PPL Perplexity. 31, 33–35, 51

RNN Recurrent Neural Network. 14, 15, 20, 21, 30, 31

SFSM Stochastic Finite State Model. 7, 10, 11, 16, 17, 19, 23, 24, 28, 29, 33, 34, 40, 41, 45, 47–52, 57, 58

SLM Statistical Language Model. 7, 11, 12, 14, 15, 19, 20, 23, 24, 31, 33, 36, 38, 40, 57

VNSA Variable N-gram Stochastic Automaton. 19

WFST Weighted Finite State Transducer. 21