SAPIENZA
UNIVERSITÀ DI ROMA

Postal Address Segmentation and Normalization Via Sequence Machine Learning Models

Facoltà di Ingegneria dell'informazione,informatica e statistica

Corso di Laurea Magistrale in Data Science

Candidate

Nthimbo Gift Tembo
ID number 1738522

Thesis Advisor

Prof. Sergio Barbarossa

Co-Advisors

Dr. Francesco Polimeni
Dr. Matteo Giacalone

Academic Year 2018/2019

Thesis defended on October 2019
in front of a Board of Examiners composed by:

Data Science Committee Members (chairman)

Postal Address Segmentation and Normalization Via Sequence Machine Learning Models

Master thesis. Sapienza – University of Rome

This thesis has been typeset by LaTeX and the Sapthesis class.

Author's email: nthimbo.tembo@gmail.com

Abstract

The Business Processes that make use of postal addresses, provided by humans in free text format, need a solution to convert them into a structured canonical form. Such solutions are traditionally realized by Service Providers that manage and keep update of an official address book provided by Authoritative Sources for name places.

The objective of this work was to understand if it is possible to normalize the postal addresses by means of a Machine Learning model trained on a Data set that contains partial and noisy addresses instead of full and clean dictionary of official addresses.

The Address Normalization has been divided in two different tasks that were undertaken with two different Machine Learning models: The Segmentation task, that assigns a tag to each word of the input address, was implemented by a Hidden Markov Model; the Standardization task, that convert the address into a canonical form, was implemented by a Recurrent Neural Network. The System provides a tool that learns the structure of input data and provides a highly probable segmented and standardized address.

Considerations on the feasibility of this approach and recommendation for further analysis are summarized at the end of the work.

Acknowledgements

I would like to take this time to offer my gratitude to my thesis supervisor Prof. Sergio Barbarossa for having allocated time on his busy schedule to supervise this thesis. The door to Prof. Barbarossa's office was always open whenever I ran into a trouble spot or had a question about my research. He consistently welcomed me and steered me in the right directions during my work and writing of this thesis.

I further offer many thanks to my co-advisers from INNAAS AskData Dr. Francesco Polimeni and Dr. Matteo Giacalone without which the thesis was not going to be possible. Without their passionate participation and input, the validation and advice, the implementation would not have been a success.

Contents

Chapter 1

Introduction

This Chapter describes the main goals of this dissertation and in particular we provide the objectives and motivation for carrying out our study. Segmenting of text into constituent elements presents key challenges such as the consistence of segmented text as regards to possibles tags, preservation of order of sentences as they appear in free text, reduction of redundancy in the generated tokens and provision of learning in long range dependency data.

1.1 Motivation

The problem of postal address Segmentation and Normalization seems easily explicable with regard to the standard rule based approaches. Anyway, there are a number challenges that are rather daunting, but can be solved with application of standard-yet powerful tools.

Firstly, there is the issue of having the same text address written in different formats. This being due to the fact that so many countries have different standards for postal addresses. For instance, the Italian standard postal address takes the format of “street type, house number, postal code, area, city, province, region” while for the United States the form is “street address, city, state or province, postal code”.

Then, secondly, there is the issue of misspellings, missing of some segments in the input text to be segmented, typos, and non-standard abbreviations. For the

sake of clarity given two Italian input addresses “Via G Mazzini Marsala” and “V.G Mazzini Trapani”, the standardized forms are "via Mazzini" if the city is Marsala and "via Giuseppe Mazzini" if the city is Trapani. In this case both the presented addresses contain abbreviations and uncompleted segments which are non-standard. The contextual mapping could be used to manage this issue.

The above stated challenges pose a level of difficulty when you are faced with a problem of segmentation and normalization of sequential data. Therefore, we have been motivated to apply Machine Learning Models to learn the structure and sequence of the postal addresses based on partial data-set given. The main idea is to infer the correct addresses based on the learned probabilities in the case of Hidden Markov Models and the Recurrent Neural Networks.

1.2 Objectives and Contributions

Our main objective is to provide an innovative online service for address segmentation and normalization realized by means of translating free text input address into structured and normalized format. The service is based on partial data-set of addresses in contrast to service they have available. Furthermore, the service is a restful application and accessible through user interface. in addition, the service has the ability for continuous improvement.

The system is based on two supervised machine learning models for sequential data. We applied Hidden Markov Models for segmentation tasks, and a probabilistic model (Statistical Corrector) and Recurrent Neural Networks for standardization tasks. The address segmentation phase is responsible for mapping the original input free text address into structured form whilst standardization phase is intended to convert the input free text address into canonical form where spelling mistakes, format and typos mistakes are corrected.

1.3 Thesis Outline

The rest of this dissertation is divided into different chapters. Each chapter presents the contents with focus on the title of the chapter. In Chapter 2, we present literature review of past research as relates to our research. Chapter 3 presents the details of the our research problem and detailed theory for working with sequential data-sets. The detailed implementation details of the approach we employed are presented in chapter 4. Furthermore, in chapter 4, we explain the reasons behind our adoption of the methodology used in our research and also the description of the experimental results for each methodology to provide a clear and concise picture of how each learning approach used scales with the data used. In chapter 5, we provide conclusions of our research as regards to the experimental results and we also provide details about our future research directions indicating the methods we intend to employ for future research.

Chapter 2

Literature Review

This Chapter describes some of the related research work in Natural Language Processing(NLP) pertaining to Name Entity Recognition. We however, put much focus on Name Entity Recognition especial to Address Segmentation and Standardization. Address Segmentation and Standardization is pretty much related to most of the problems in Natural Language Processing for sequential data.

2.1 Name Entity Recognition

Name Entity Recognition (NER) is one of the most important information extraction principles in Natural language Processing, which is principally concerned with associating names of entities such as people, organizations, locations and products. Name Entity Recognition has been well known to consists of two phases; Entity Detection and Entity Classification [31]. Entity Classification involves recognizing of snippets in free text and mapping them to actual real world objects. As a result of the classification nature in Name Entity Recognition, it has been widely adopted principle in many crucial areas of information management such as Semantic Annotation, Opinion Mining, Question Answering and Ontology Population [32].

With reference to historical information between 1966 and 2008, the evaluation metrics for NER data used was mainly through Message Understanding Conference(MUC), Computational Natural Language Learning(CoLL) and Automatic Content Extraction(ACE). The definition of tasks had close similarity as was the

semantics involved. Hence, given these semantics, the objective was to map snippets in free text to real world entities [32].

Contrary to the early years in Name Entity Recognition, nowadays NER systems developed use different types of Statistical models. These systems employ sophisticated Machine Learning algorithms and are optimized to provide better classification for different types of data. Some areas where Machine Learning for NER has provided reasonable and satisfying results include; Automatic Summarizing of Resume, Optimization of Search Engine Algorithms, Powering Recommender Systems etc.

2.1.1 Rule-Based Approach

This approach to NER was most used in the past prior to the advent of Machine Learning approaches to NER. Such systems use both internal and external evidence and word trigger dictionaries or hash tables for mapping text snippets to real world entities [33]. The dictionaries in Rule-Based approached are specifically used to speed up pattern recognition. In addition, it is not unusual in today's systems that one is able to encounter some form of Rule-Based systems augmented with Machine Learning methods. Such systems have been recognized as Hybrid NER systems since they incorporate both Machine Learning and Rule-based approaches. In Rule-Based NER systems, the rules to be followed by the system are manually built by an expert Linguist and are often augmented with look-up dictionaries or tables for information extraction from text. Since in Rule-Based NER systems the rules are manually configured by an expert Linguist, then it follows by intuition that such systems are to have high level of accuracy, robustness and coverage of obtained results. However, they also poses an excruciating caveat in relation to the time it takes to perform the annotations of the rules by an expert and also during updates it means the expert has to go through rules once more.

Nowadays, Rule-Based approaches are mostly used as a substitute for Machine learning methods if there is no ample annotated training data corpus for training Machine Learning Models.This approach consists mainly of two steps depending on the implementer of the system and the kinds of rules to be defined. The first

step is tokenization, whose objective is to split the sentences into tokens. In this stage, the sentences in a document are split into words, punctuation and numbers. The second phase involves Part of Speech Tagging(POS). To retrieve part-of-speech tagging, a Rule-Based part of speech tagger must be implemented as in the work of (Rayner Alfred, Leow Chin Leong, Chin Kim On, and Patricia Anthony) [35]. In their implementation for the Rule-Based system they defined a number of contextual features for the system. The rules were designed to detect three major types of named entities that include a person, an organization and a location. For instance, the tokenized words were initially evaluated using the POS tag dictionary[35]. Figure 2.1 shows the generic design flow for Rule-Based NER systems.

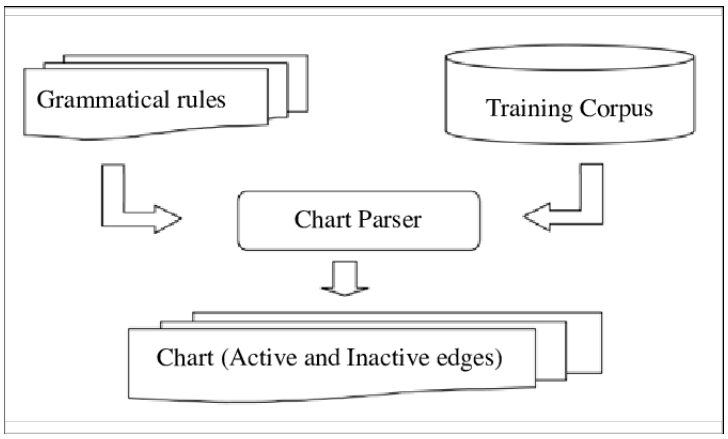

Figure 2.1. A generic design of Rule-Based NER

2.1.2 Machine-Learning Approach

Address Segmentation and Standardization approach belongs to large family called the Information Extraction. Information Extraction means finding a way to extracting information from a sequence of data according to the requirements of the information need. Finding useful information can be presented in two categories – first getting information from data by using the traditional rule based pattern matching systems and secondly using machine learning approaches. Our approach is to use machine learning algorithms for the discovery of the patterns in the presented data. Our focus is on segmentation using Hidden Markov Models and standardization using Recurrent Neural Network augmented with Statistical Corrector.

Hidden Markov Models are widely applied to many research fields concerned with information extraction. They mostly are applied to a document in two different ways. The first assumption is that the document can be partitioned in some defined way. The documents are then separated into different sentences [26][27]. In addition there is need to assume that the addresses in information extraction are already identified prior to the application of Hidden Markov Models [28]. This simply means having a full list or database of all the address prior to information extraction for standardization purposes. However, in our case we don't follow this approach since our system is based on partial data-set. The second approach to Hidden Markov Model is to apply the model on the entire document, and the Hidden Markov Model has to extract the features from the document in an unsupervised way. These are just some of the approaches that have taken course for Hidden Markov Models in the field of Natural Language processing.

Different models have been built for Natural Language Processing. There are now many open source libraries built that implement different kinds of algorithms for Natural Language Processing. For example, there is the open source Neural Machine Translation models [29][30]. In our case, we lay our focus on standard Neural Machine Translation and the Attention Mechanism in Neural Machine Translation implemented by Stanford and Harvard Universities [24][23] for research purposes.

Chapter 3

Problem and Methodology

This Chapter describes the visibility of learning patterns from Sequential data which can be represented in spatial or temporal forms. Temporal representation in this case is either explicit-where the dimension of time is apparently exhibited in the data or implicit-where the notion of time is not clearly exhibited in the sequence of data. The data used in the study stipulates an implicit notion of sequential data, hence, we base our focus tremendously on sequential data exhibiting implicit temporal behaviour.

3.1 Learning From Sequential Data

When working with data that presents poor assumptions for independence and identity in distribution, such as sequential data that often arises when dealing with time series data. For instance, heights of ocean tides, dialing close values of the stock markets, rainfall measurements on successive days at a specific location e.t.c. Sequential data can also arise in areas other than time series such like the sequence of neucleotides base pairs along a DNA strand or the sequence of words in any spoken language. Sequential data poses two types of distributions, stationary and non-stationary distributions. In the stationary case, the distribution of data evolves in time while the distribution from which the data was drawn is unchanging. In the non-stationary case, the generative distribution from which the data was drawn from evolves in time [1]. However, in this dissertation our focus is in learning from

stationary distribution perspective where the data generating process is assumed to be static.

In many applications that involve sequential data-sets such as financial forecasting, the objective is to be in a position to predict the next value in a series given the observations of previous values. By intuition, we expect the most recent observations to have more information than the historical values in predicting the future values. Furthermore, it would be impractical to consider a general dependence of future predictive values on all previous observations in time series since the complexity of such a model would go to infinite with an increase in the number of observations [1]. This theory leads to consider using Hidden Markov Models where the assumption that the future is independent of the past observations given the present observation holds dearly. However, contrary to hidden Markov Models, other complex models such as Recurrent Neural Networks provides a way to consider more of historical data (previous observations). Thus, Hidden Markov Models provides short range dependencies as opposed to the later model which provides long range dependencies putting it at an advantage in applications where loosing data dependence is critical [2].

3.2 Hidden Markov Models

The easiest way to model with sequential data would be to just ignore the sequential aspect and treat the data as $i.i.d$ (Independent and Identically Distributed). However, such a model would fail tremendously to discover sequential patterns in the data such as correlations for observations that are close in sequence (time) [1].

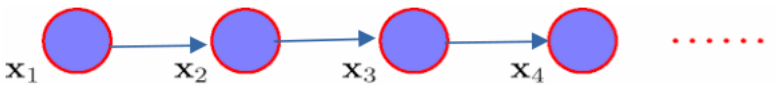

Figure 3.1. Simple graphical model for sequential data

If for instance we considered time series data to be a representation in figure 3.1 depicting a sequence of weather conditions in consecutive days either rainy or

sunny. The idea is that, we would like to predict the state of the next day given the states of the current day and the past states. Therefore, knowing the weather conditions for the current day is of significant help in predicting the conditions for next day. Such effects can be expressed in a probabilistic model by conditioning the future occurrence given the present and past occurrences. Hence, the need to relax the idea of independent and identically distributed (i.i.d), assumption and consider using Hidden Markov Models for such data. Without loss of generality we can use the product rule to express the joint distribution for a sequence of observation in the form;

$$p(X_1, ..., X_n) = \prod_{n=1}^{N} p(X_n|X_1, ..., X_{n-1}) \tag{3.1}$$

In the case that we assume that the conditional distributions on the right hand side are all independent of the previous observations except the most recent observation then we will have a first-order Markov Chain which is depicted in Figure 3.1 and conditioning two most recent observations gives second-order Markov Chain. Under such a model the joint distribution for a sequence of N observation is given by;

$$p(X_1, ..., X_n) = \prod_{n=2}^{N} p(X_n|X_{n-1}) \tag{3.2}$$

By the d-separation property the conditional distribution of observation X_n, given all other observations up to time step n, is given by;

$$p(X_n|X_1, X_2, ..., X_{n-1}) = p(X_n|X_{n-1}) \tag{3.3}$$

If we were to be constrained to use such a model as above, then our future predictions will just depend on the current observations and will be independent of earlier observations. Thus, loosing long range dependencies which is critical in some applications which depend highly on such a property. If we however, consider the previous observation to the current then the model becomes;

$$p(X_n|X_1, ..., X_{n-2}) = p(X_n|X_{n-2}, X_{n-1}) \tag{3.4}$$

We can extend such a model (3.4) to have as much dependence on the number of previous observation, however, the model complexity grows to infinity with number of observations conditioned upon [1]. We can represent the Hidden Markov Model's observations and states using a graphical model as postulated in figure 3.2.

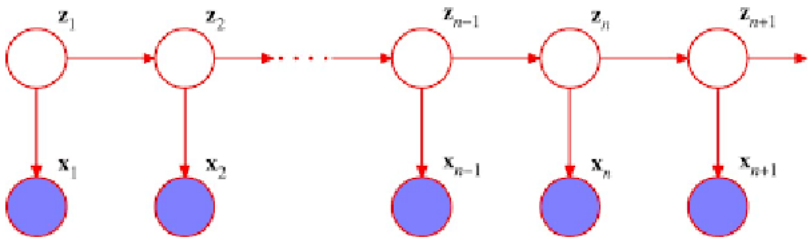

Figure 3.2. Representation of Sequential data using Markov Chain with latent variables

If in the graph the latent variables Z_i are discrete, then we obtain a Hidden Markov Model [3]. In a Hidden Markov Model or HMM the observed variables can either be discrete or continuous and different conditions can be used to model the observed variables. In this thesis, the discrete case assumption holds for working with the data.

3.2.1 Hidden Markov Models Architecture

A generic Hidden Markov Model is shown in Figure 3.2, with two parts, the Markov Processes Z_i and Observations X_i. The observations are denoted as sequence $\{1, ..., n\}$ which makes the notation simpler with no loss of generalization.The Markov process illustrated in the diagram is determined by the state and the state transition matrix often represented as A which consists of the probabilities of transitioning from one state to the other. we denote with symbol B the state emission matrix which consists of the probabilities emitted by each state given a particular observation.

The random variables X_i represents an observation at time $t \in T$ where T is the total time-space. The random variable Z_i represents the hidden state at time t where $Z_t \in (Z_1, Z_2, ..., Z_T)$. Hence, it is apparent from Figure 3.2 that the conditional probability distribution of the latent variable Z_t at time t, when all other values of the latent variables Z at all times are given (known), then Z^t depends only on the most recent value Z^{t-1}. In the first-order Hidden Markov Model, we assume that all latent values before the latent value at time $t-1$ have no influence on the latent value at time t. This is what is referred to as the Markov Property known commonly by an adage "The future value is independent of all the past values given the present value". In the same way, the value of the observed variable X^t depends only on the value of latent variable at time t which is Z^t. In this way, the hidden or latent state space is assumed to be constituent of N possible values, which are given a probability distribution. This in a proven way gives the notion that for each of the given states, the latent variable can be in any of the given states at time t. Therefore, there is clearly a transition for the latent variables from each state at time t to time $t+1$ given by transition probabilities. These transition probabilities are represented as a $N * N$ or N^2 matrix A called the transition probability matrix. These transition probabilities have a profound control in the way the hidden state at time $t-1$ is chosen to move to hidden state at time t . The matrix A is a row matrix where the rows must sum to 1. Overall there are a total of $N(N-1)$ transitions which are supposed to be made and not N^2, since one can only transition to the next state given they are in the current state [4].

In a similar fashion, for each of the hidden states there is a distinct emission probability associated with each of the possible observations. Therefore, this probability is represented in a $N(M-1)$ where M is the total number of possible observations which holds only when the observations are categorical. In this dissertation, we will always work with this assumption. These probabilities give distribution of each hidden state in relation to the possible observations. Figure 3.3 presents a table for the notations to be used for hidden Markov models.

Symbol	Description
π	Initial state distribution
X={ $x_1, x_2, \ldots\ldots, x_T$ }	Observation sequence
T	The total length of observation sequence
N	The number of hidden states or Latent values
n	The number of observations
S = { $s_1, s_2, \ldots, s_N$ }	Distinct states of a Markov Process
V = {1, 2, .., n}	A set of possible observations
A	State transition matrix
B	Emission probability matrix

Figure 3.3. Table of notations for Hidden Markov Models

3.2.2 Hidden Markov Models Inference

In a Hidden Markov Model the goal is essentially to pose and answer three fundamental questions as expected in real world applications, which are;

1. Given a sequence of observations say $X = (X_1, X_2, ..., X_T)$ and the model $\lambda = (\pi, A, B)$, we would like to find the most efficient way to calculate the probability $p(X|\lambda)$ of the sequence given the model in question.

2. Given a sequence of observations say, $X = (X_1, X_2, ..., X_T)$ and the model $\lambda = (\pi, A, B)$, how can we find the most likely sequence of states $Z = (Z_1, Z_2, ..., Z_T)$ that emitted the observation sequence in the order the sequence is in.

3. Give an observed sequence and corresponding sequence of states, how do we find the probability matrices of the model for each of our model parameters that maximize the probability $p(X|\lambda)$

In the scenario postulated above, the first task involves evaluation, meaning if we are given a model and an observation sequence how can we compute the probability that the observed sequence was indeed produced by the model. In this way, we can indeed view this problem as one way of checking how well the model matches the observation sequence. The solution here is especial to the event where we have different competing models. In such an incident, we will aim to choose the model that best matches the observation sequence [5].

3.2.2.1 The Probability of A Sequence

When we make an assumption that our data was generated by a process consisting of hidden states, then we are talking in terms of a series of states emitting observed symbols forming a sequence of observations while at the same time transitioning from one state to the other. Hence, such a sequence is produced by a Markov Model parameterized by the transition matrix A and the emission matrix B. During this process, we select at each time step t , an output X_t as a function of the state Z_t at that particular time. This means that to find the probability of the sequence we need to sum up the likelihood associated with the data $\bar{X}$ as a vector of observations given every possible combination of state series as given by the equation 3.5.

$$\begin{aligned} P(X|A,B) &= \sum_{\bar{Z}} P(X,Z;A,B) \\ &= \sum_{\bar{Z}} P(X|Z;A,B)P(Z;A,B) \\ &= \sum_{\bar{Z}} (\prod_{t=1}^{T} P(X_t|Z_t;B))(\prod_{t=1}^{T} P(Z_t|Z_{t-1};A)) \\ &= \sum_{\bar{Z}} (\prod_{t=1}^{T} B_{Z_t X_t})(\prod_{t=1}^{T} A_{Z_{t-1} X_t}) \end{aligned} \tag{3.5}$$

Even if the above derivations of the equation follows all the needed Markov assumptions, the sum of this computation is over all possible assignment of states. Hence, this computation is not efficient since the algorithm will run in time $O(S^T)$ operations when evaluating the sum. However, we can make faster computations using the more efficient dynamic programming called the forward procedure. Below is the algorithm for forward procedure required in dynamic programming;

Algorithm 1 Forward Procedure for computing $\alpha_i(t)$

1. Base case: $\alpha_i(0) = A_{0\,i}$, $i = 1..|S|$
2. Recursion: $\alpha_j(t) = \sum_{i=1}^{|S|} \alpha_i(t-1) A_{ij} B_{j\,x_t}$, $j = 1..|S|$, $t = 1..T$

In the forward procedure we make an assumption that we already have the values for the quantity $\alpha_i(T)$ which is given by $\alpha_i(t) = P(X_1, X_2, ..., X_t, Z_t = S_i; A, B)$. Hence, assuming we have the value for $\alpha_i(T)$ at each state, what remains is the

computation of the summation by states.

Therefore, given such a quantity the probability becomes;

$$P(\bar{X}|A,B) = \sum_{i=1}^{N} \alpha_i(t) \tag{3.6}$$

3.2.2.2 Maximum Likelihood of A Sequence

In the first problem for hidden Markov Models in finding the probability of a sequence, there is one unique solution which makes things easier. However, in the part of maximizing the likelihood of a sequence, there are several different ways of solving for such a problem. The optimal criteria in this case maximizes the expected number of individual correct states producing the sequence [5]. In simple terms, what we are seeking to establish here is the question of what is most likely sequence of states that has a higher probability of producing the observed sequence. Looking at all the possible combinations of state sequences that might have produced the observed sequence, we aim to choose the one having the highest probability among all the combinations as shown in equation 3.7;

$$\begin{aligned} \arg\max_{\overline{Z}} P(\overline{Z}|\overline{X};A,B) &= \arg\max_{\overline{Z}} \frac{P(\overline{X},\overline{Z};A,B)}{\sum_{\overline{Z}} P(\overline{X},\overline{Z};A,B)} \\ &= \arg\max_{\overline{Z}} P(\overline{X},\overline{Z};A,B) \end{aligned} \tag{3.7}$$

The simplification follows from the Bayes rule and secondly, also knowing that the denominator has no dependence on $\overline{Z}$. If we try all the possible combinations of state sequences and find the one with the highest probability, the computations will be computationally heavy hence, not feasible in this case. This way of calculation would require computational time of $O(S^T)$ increasing with root of the number of observations in the sequence. To make things easier we use a dynamic programming method like the earlier presented forward procedure.

3.2.2.3 Parameter Learning

In a Hidden Markov Models, the most important and the most difficult task is learning the parameters for the two matrices; the transition probability matrix A and the emission probability matrix B. Here, we are interested in answering a question where we have an observation sequence and a set of states. The question given in such a scenario is - how do we find the parameters of our two matrices - that is the probabilities of our matrices as pertains to all the states and the observed sequences.

The input in this learning or training part of a Hidden Markov Model is a sequence of unlabelled observations X and a vocabulary of potential hidden states Z. The standardized algorithms for learning are the Forward-Backward or the Baum-Welch algorithm which is a special case of the Expectation Maximization algorithm [6]. The algorithms makes it possible to train both the transition probabilities A and the emission probabilities B. However, though we mention the use of these algorithms, in this dissertation we calculated the transition probabilities and the emission probabilities directly from the data we had. Therefore, there was no need for us to use these algorithms. The calculations of these probabilities in our case was as presented in equation 3.8;

$$\hat{a}_{ij} = \frac{expected\ number\ of\ transitions\ from\ state\ i\ to\ state\ j}{expected\ number\ of\ transitions\ from\ state\ i} \tag{3.8}$$

Here the computation for numerator, we assume that we had some estimate of the probability given transition $i \rightarrow j$ was taken at particular point in time in the observed sequence. Knowing the probability for each particular time t, we now just sum over all times t estimate the total count for the transition from state i . Then, the denominator is just the total number of transitions from state i. This formula clearly shows that the value of $\hat{a_{ij}}$ is merely a probability which is between zero and one.

In a similar fashion, we can compute the probabilities of observations or emission probabilities of B. This is the probability of emitting a symbol V_k by the hidden state j from the observation vocabulary V . Therefore, the computation is easily done using equation 3.9.

$$\hat{b}_{j(V_k)} = \frac{expected\ number\ of\ times\ in\ state\ j\ and\ observing\ symbol\ V_k}{expected\ number\ of\ times\ being\ in\ state\ j} \tag{3.9}$$

Here the formula explains it in a simple and intuitive way, since we merely compute the probability of being in state j while observing a symbol V_k. We count the number of times we are in state j while observing a symbol v as the ratio of the total number of times we are in state j .

3.2.3 Smoothing Techniques

When working with algorithms that involve Maximum Likelihood Estimation(ML) for estimation of probabilities, smoothing techniques are handy to avoid getting zero or very low probability values that might give a negative impact on the computations involved giving a maximum likelihood estimation of zero which is not supposed to be the case [7]. In Hidden Markov Models where the vocabulary under consideration is very large, smoothing techniques have a profound importance in a scenario that we encounter cases where the probability is zero or extremely low that it adds insignificant value to the overall estimation of ML [8]. In such cases smoothing techniques adds little probabilities according to the type of technique employed in such a way that overall, we avoid getting wrong computations of the ML [9]. Here, we briefly describe the different types of smoothing techniques that can be employed in different scenarios. In our implementation we used Laplace smoothing technique which was easier to implement and achieved high accuracy.

1. Laplace Smoothing or Additive Smoothing

   This kind of smoothing technique employs Bayes Optimal in conjunction with the Dirichlet prior where the count of every word in the vocabulary is incremented with a “priming” occurrence [7]. Since we computed the emission and transition probabilities without employing the Forward-Backward or the Baum-Welch Algorithm we therefore, used this type of smoothing technique though it is not the most optimal according to theory.

2. Absolute Discounting

   Absolute discounting performs better that Laplace smoothing and is especial in situations where the number probabilities having zero-count words varies from state to state in the overall modelling process. In Absolute discounting, the core idea is to subtract a fixed value from all word counts having count greater than zero. The probability mass in this case is then distributed in a uniform way to the words that have zero counts [9]. The actual calculations for this type of smoothing procedure is presented in equation 3.10.

$$P(X \mid Z) = \frac{N(X,Z) - d}{N(Z)}, if\ N(X,Z) > 0\ or\ = \frac{d(|V| - |S_Z|)}{|S_Z|},\ if\ N(X,Z) = 0 \tag{3.10}$$

In the above equation V is the vocabulary size, X is the number of times observation X_i occurs, $N(Z)$ is the total number of observation occurrences for state Z across the possible vocabulary and S_Z denotes the total number of observations with zero count in state . There is no standard optimal value for d as reported by many theoretical papers [9] .

There are several other smoothing techniques which we did not consider such as; Good-Turing estimate, jelinek-Mercer smoothing, Katz Smoothing, Witten-Bell smoothing, Kneser-Ney smoothing, Church-Gale smoothing, Bayesian smoothing e.t.c [10].

3.3 Neural Networks

A Neural Networks can be defined in simple terms as interconnected assembly of simple processing units, nodes or elements emulating the behaviour of animal based neurons. The processing ability of such a network is stored in the inter-unit connection strengths, or weights obtained by learning from a set of training patterns [11]. The term network in this context will be used to refer to any system of artificial neurons since the principal idea behind neural networks is to imitate the way the human Neurons communicate via electrical signals that are short lived impulses or spikes in the voltage of cell membranes, with the synapses acting as the inter-neuron connections.

It is such a biological style that motivated the study of neural networks with the hope of emulating the exact operations in the biological setup. Artificially equivalents of biological neurons are the nodes. Synapses are modelled as weights, where inputs are multiplied by weights before being sent to the cell body consistent with the biological concept. The weighted signals are thereafter, summed together by some arithmetic function to supply an activation node. Figure 3.4 shows a simple work of artificial neuron called the threshold logic unit(TLU). After the activation is computed, it is then compared to the threshold; it produces a high-valued output, conventionally 1 otherwise it outputs zero.

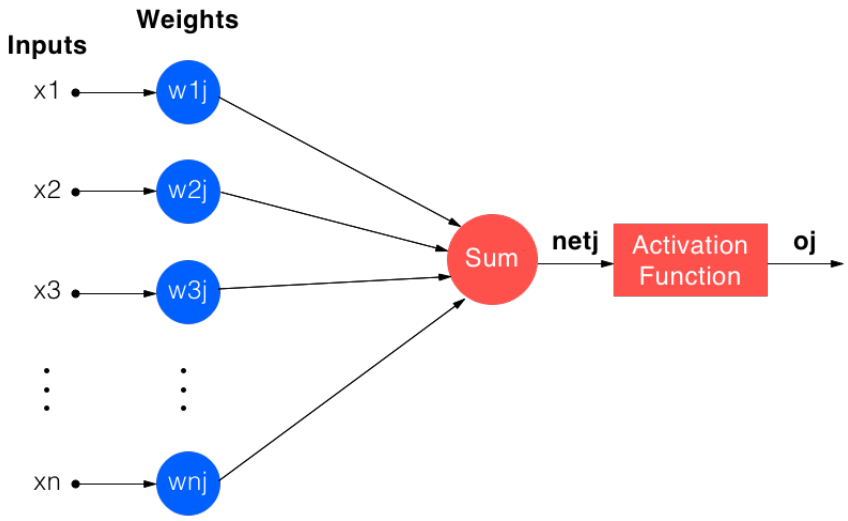

Figure 3.4. Example of an Artificial Neuron

3.3.1 Neural Networks Architecture

As already stated that in Neural Networks we try to emulate the structure of biological neuron. Here, we present the architecture of a Neural Network as seen in figure 3.5. Each node is shown by the circles while the weights are implicit to the connections shown by the arrows. In the structure the signals move from the inputs passing through the nodes in the hidden layer before reaching the output as indicated by the arrows. Such a structure of Neural Network is called a Feed-Forward Neural Network and is just one of the many different types of Neural Networks [11]. Neural Networks are sturdy learning models that achieve state-of-the-art results in a variety of supervised and unsupervised machine learning problems. Principally, they are most convenient for machine perception problems, where the features underlying the data are not interpreted individually. The success of Neural Networks pins down to their ability to learn Hierarchical representations as compared to traditional methods (supervised and unsupervised methods) that depend on hand-engineered features [12]. In recent years the problem of data storage is no longer an issue to be wary about as there has been much development in the field of data storage which has made data storage to become more affordable with the increase in the volumes of data-sets. Fitting large data-sets to simpler linear models tend to under-fit the data

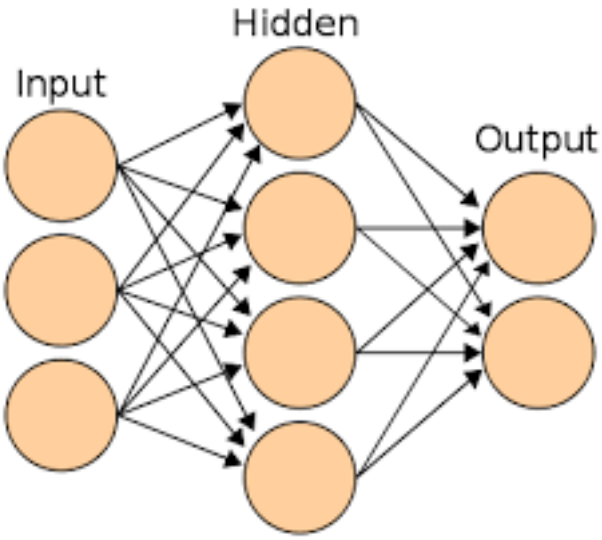

Figure 3.5. An example of neural network with 1 hidden layer and 4 nodes

and tend to under-utilize the computing resources. When working with huge data-sets, deep learning models especially those based on Deep Belief Neural Networks (DNNs), constructed by stacking restricted Boltzmann machines, and convolution neural networks, which exploit the local dependency of visual information, have demonstrated record-setting results on many important applications [2].

Despite their amusing and catching features of Neural Networks, they present a good number of limitations. Most notably, they rely on the assumption of independence as pertains to the training and test examples, hence not capturing patterns in data with long range dependency especially for data which are related in space or time such as time series data. We therefore, aim to extend this powerful learning tool to model data that has temporal or sequential structure and varying length of inputs and outputs in our data-sets. Recurrent Neural Networks(RNN) are connection extensions of Neural Networks and do not rely on the assumption of independence and consequently, are able to capture long range dependencies in sequential data. Recurrent Neural networks are not the only models with the capability to capture data dependence patterns in time or in space. As already accounted for earlier when presenting the Hidden Markov Models. Hidden Markov Models also work with temporal or spatial data. However, traditional Hidden Markov

Models are limited in that their states must be drawn from a smaller sized discrete state space S. Furthermore, the complexity of Hidden Markov Model increases with an increase in the number of hidden states if we are to include long range dependencies.

With the limitations of Hidden Markov Models, it makes sense that Connectionist Models, i.e., Artificial Neural Networks should achieve better results. Recurrent Neural Networks have a great advantage over the Hidden Markov Models and work in a subtle similar way as in Hidden Markov Models. Any state in a traditional RNN relies only on the state of the network at the previous time step. Contrary to the HMM, the hidden state in RNN can contain information from a nearly arbitrary long context window. While the expressive power of a RNN grows exponentially with the number of nodes, the complexity of both training and inference grows in at most quadratic time [2].

3.3.2 Recurrent Neural Networks Architecture

Recurrent Neural Networks are feed-forward Neural Networks that have been extended with inclusion of addition edges that span adjacent time steps which introduce the idea of time in the model. Just like traditional feed-forward Neural Networks, RNN may also not include cycles in their conventional edges. Contrary to the traditional Neural Networks, RNN have edges that connect adjacent time steps, called the Recurrent edges which may form cycles, including cycles that are self connection from a node to itself across time with a length of one. At any time step t , nodes with recurrent edges receive input from the current data point X^t and values $h^{(t-1)}$ from hidden node in a previous state of a network. The output $\overline{y}^{(t)}$ at each time step t is calculated from the node values $h^{(t)}$ at time t . Hence, the input $X^{(t-1)}$ at time step $t-1$ can affect the results of the output $\overline{y}^{(t)}$ at time t and in the future output by way of the recurrent connections [13].

There are mainly two equations that specify all calculations at each given time step on the forward pass in particular to a simple recurrent neural network in figure 3.6:

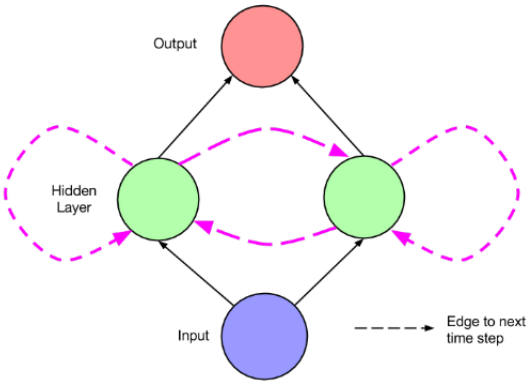

Figure 3.6. A simple recurrent neural network

$$h^{(t)} = \sigma(W^{hx}X^t \ + \ W^{hh}h^{(t-1)} \ + \ b_h) \tag{3.11}$$

$$\overline{y}^{(t)} = \ softmax(W^{yh} \ + \ h^{(t)} \ + \ b_y) \tag{3.12}$$

In this presentation, W^{hx} is a matrix of weights between the input and the hidden layer while, W^{hh} represents a matrix of re-current weights between the hidden layer and itself at adjacent time steps. The values b_h and b_y represents vectors consisting of the bias parameters to the network. Figure 3.7 shows the unfolded network of figure 3.6 having an apparent picture that the network can be trained in many time steps using back-propagation. In context with figure 3.7, recurrent neural networks use an algorithm called Back-Propagation Through Time(BPTT) for training.

The introduction of architectures for supervised learning on sequences was due to the work of Jordan [14]. As depicted in figure 3.6, which shows a simple feed-forward network with only a single hidden unit that can be extended depending on the requirements. An extension of a neural network to contain more than one hidden unit is what is termed the deep neural network.

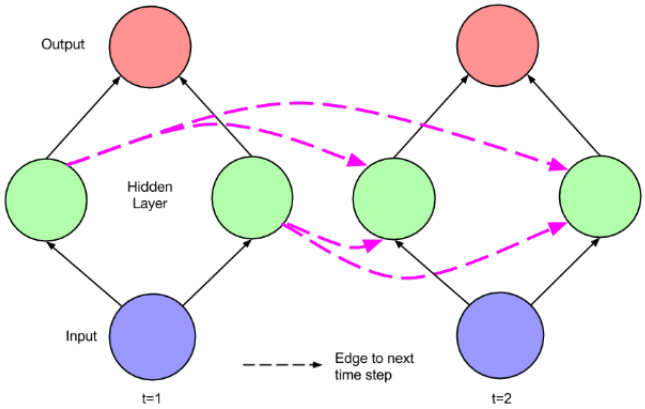

Figure 3.7. A recurrent Neural network of figure 2.6 in unfolded form

3.3.3 Recurrent Neural Networks Pattern Matching

Historically learning with Recurrent Neural Networks was always considered a challenging task, even learning with the standard feed-forward networks was viewed in a similar fashion due to the optimization problems [15]. However, the long range dependencies in Recurrent Neural Networks presents more challenging problems when learning long range dependencies using Gradient Descent Algorithm. There are mainly two problems associated with the task of learning in RNN - the vanishing gradients and the exploding gradients which occur when back-propagating errors across multiple time steps [16].

To understand more about the two problems in Recurrent Neural Networks we considered figure 3.7 of a simple Recurrent Neural Network with a single recurrent node. The network in figure 3.8 has a single input unit, single recurrent unit, and a single output unit. If we are to consider an input passed to the input layer and some error calculated at time t with the assumption that the input in the intervening steps is zero, then the binding of weights at successive time steps means that the recurrent edge at any hidden node j will always have the same weight. This therefore, means that the contribution of the input at time t will either explode or approach zero exponentially fast with an increase in the number time steps. The derivative of the error in this case is expected to vanish or explode with respect to the input[2].

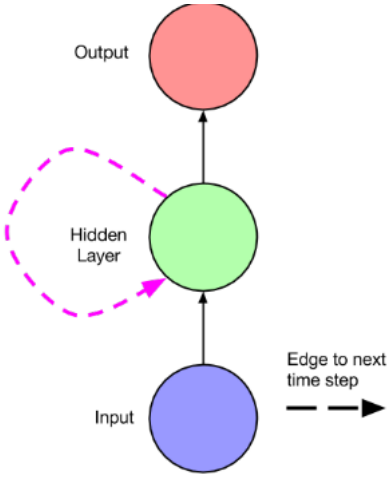

Figure 3.8. A simple recurrent net with one recurrent hidden unit

The occurrence of any of the two events depends solely on whether the weight of the recurrent edged $W_{jj} > 1$ or $W_{jj} < 1$ and also on the activation function in the hidden node. Recurrent neural networks use the rectified linear unit to avoid the problems of vanishing gradient unlike the Sigmoid function. In order to avoid the problem of exploding gradients, recurrent neural networks use a special type Back-Propagation called the Truncated Back-Propagation Through Time (TBPTT) especial to continuous running networks [2].

Truncated Back-propagation Through Time is arguably the most practical method for training recurrent neural networks [17]. Truncated Back-propagation Through Time was first used by Elman [18], since then TBPTT has been a success at training RNN on language level modelling [13]. One of the main problems of BPTT is its high cost of a single parameter update, which makes it impossible to use a large number of iterations. For instance, the gradient of a Recurrent Neural Network (RNN) on sequences of length 1000 costs the equivalent of a forward and a backward pass in a neural network that has 1000 layers. The cost can be reduced with a naive method that splits the 1000-long sequence into 50 sequences (say) each of length 20 and treats each sequence of length 20 as a separate training case.This is a sensible approach that can work well in practice, but it is blind to temporal dependencies that span more than 20 time steps [19]. It processes the sequence one time step at a time, and every K_1 time steps, it runs BPTT for K_2 time-steps, therefore, a parameter update can be cheap if K_2 is small. Consequently, its hidden states may have been exposed to many time steps and so may contain useful information about the far past, which would be opportunistically exploited. This cannot be done with the naive method. Below is an algorithm for TBPTT.

Algorithm 2 : The Truncated Backpropagation Through Time

1: **for** t **from** 1 **to** T **do**
2: Run the RNN for a single step, computing h_t and Z_t
3: **if** t **divides** k_1 **then**
4: Run BPTT from t to $t-K_2$
5: **end if**
6: **end for loop**

Figure 3.9. The Truncated Back-propagation Through Time Algorithm

3.3.4 RNN Long Short-Term Memory

The breakthrough in the use of recurrent neural networks were visibly seen after the publishing of two papers. The first paper was published on Long Short-Term Memory (LSTM) [20] and the other on Bidirectional Recurrent Neural Networks (BRNN) [21]. The main idea of the LSTM architecture is the introduction of a memory cell which is a unit of computation which replaces the traditional hidden nodes in the network. The two innovations work in mutual way and they are complementary in most implementations.

The Long Short-Term Memory was principally designed to overcome the problem of vanishing gradients as earlier explained. The architecture of this model is similar to the architecture of recurrent neural network with hidden layers. However, the architecture of LSTM consists of a memory cell in the hidden layer, replacing the nodes with the memory cell as opposed to the traditional recurrent neural networks. The nodes contained in the memory cell consists of self-connected recurrent edges fixed with weight one, which ensures that the gradient can pass over multiple time steps without vanishing or exploding. A memory cell is a composite unit, built from simpler nodes in a specific connectivity pattern, with the novel inclusion of multiplicative nodes. The LSTM memory cell therefore, provides an intermediate form of storage. Figure 3.10 shows a structure of LSTM model [2].

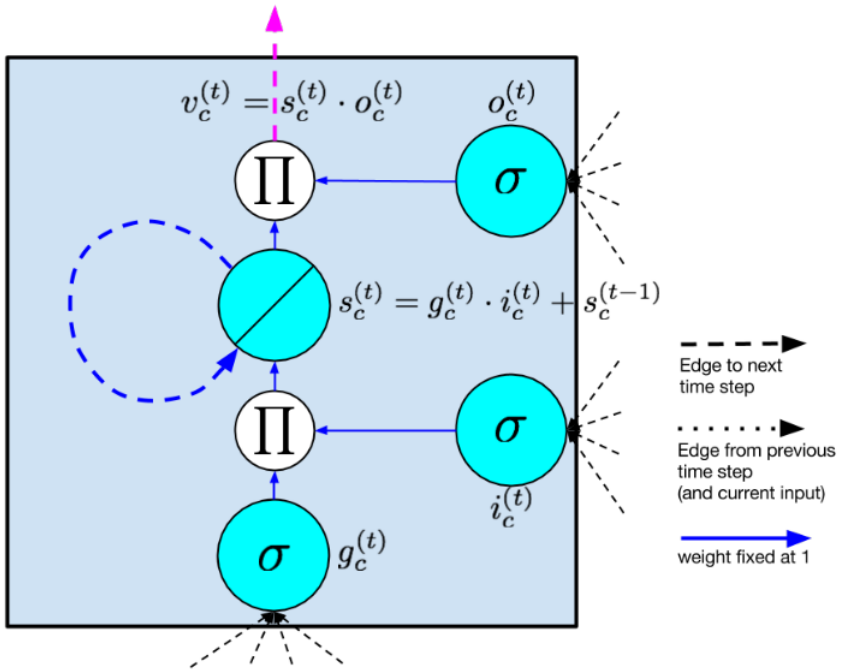

Figure 3.10. One LSTM memory cell. The self-connected node is the internal state

The calculations of values for all components of the LSTM model as proposed by the first published paper are stipulated below;

1. Input node: In the figure labeled g_c takes activation input from an input layer vector $X^{(t)}$ at time step t and activation $h^{(t-1)}$ from the hidden layer at time $t-1$. The summed weighted input will be then run through the activation function.

2. Input gate: labelled i_c , takes in activation like the input node from current data point $X^{(t)}$ and activation $h^{(t-1)}$ from the hidden layer at previous time step. Its main function is either to allow or disallow flow to pass through it. It is capable of acting as a bridge for the flow since it computes its value as either one which allows flow or the value of zero which disallows all flow from other nodes passing through it. It multiplies its value by the value of the incoming flow.

3. Internal State: labelled S_c and is at the heart of each memory cell with linear activation. Internal state allows error flow across multiple time steps hence,

avoiding vanishing or exploding gradients. The update of the state is given by; $S^{(t)} = g^{(t)} \odot i^{(i)} \ + \ S^{(t-1)}$ having point-wise multiplication

4. Forget gate: labelled f_c provides a way for the learn network to learn to send the contents of the internal state. This kind of gate is especial in continuous running networks. The calculation done on the forward pass is given as; $S^{(t)} = g^{(t)} \odot i^{(i)} \ + \ f^{(t)} \ \odot \ S^{(t-1)}$

5. Output gate: The output value V_c produced by the memory cell is the product value of internal state S_c multiplied by the value of the output gate O_c . However, in other neural network research, rectified linear units, which have a greater dynamic range, are easier to train. Thus it seems plausible that the nonlinear function on the internal state might be omitted.

The overall computation in the Long Short-Term Memory model proceeds as with the equations below. The calculations are performed at each time step . These steps which are augmented with the forget gates describes the Long Short-Term Memory algorithm in full.

$$i^{(t)} = \sigma(W^{iX} X_{(t)} \ + \ W^{ih} h^{(t-1)} \ + \ b_i) \tag{3.13}$$

$$f^{(t)} = \sigma(W^{fX} X_{(t)} \ + \ W^{fh} h^{(t-1)} \ + \ b_f) \tag{3.14}$$

$$o^{(t)} = \sigma(W^{oX} X_{(t)} \ + \ W^{oh} h^{(t-1)} \ + \ b_o) \tag{3.15}$$

$$S^{(t)} = g^{(t)} \odot i^{(i)} \ + \ S^{(t-1)} \ \odot \ f^{(t)} \tag{3.16}$$

$$h^{(t)} = \phi(S^{(t)}) \ \odot \ o^{(t)} \tag{3.17}$$

The actual value of the Long Short-Term Memory architecture at time t for its hidden layer is a vector given as $h^{(t)}$ and the values of the output from each memory cell is given by $h^{(t-1)}$ in the hidden layer at time step $t-1$. If we want to exclude the forget gates, then in the computation we set $f^t = 1$ at time step t [22].

3.3.5 Bidirectional RNN

The bidirectional recurrent neural network (BRNN) contrary to the LSTM consists of two layers of hidden nodes. The basic idea is that one node ought to capture the computations for the forward pass and the other for the backward pass in the negative direction [21]. Both computations involve two kinds of calculations - the computations for states either in the forward pass or backward pass and the computations for output neurons in either direction. During training of the BRNN, the gradients can be computed using the traditional back-propagation algorithm after unfolding across time. For instance given an input sequence and an output sequence the enumerations of the BRNN model can be represented by the equations stipulated below [2];

1. The forward pass: enumerations for the states and the output neurons

$$h^{(t)} = \sigma(W^{hx}X^{(t)} \ + \ W^{hh}h^{(t-1)} \ + \ b_h) \tag{3.18}$$

2. The backward pass: enumerations for the states and the output neurons

$$z^{(t)} = \sigma(W^{zx}X^{(t)} \ + \ W^{zz}h^{(t-1)} \ + \ b_z) \tag{3.19}$$

3. Final BRNN output:

$$\overline{y}^{(t)} = softmax(W^{yh}h^{(t)} \ + \ W^{yz}h^{(t+1)} \ + \ b_y) \tag{3.20}$$

The Bidirectional Recurrent Neural Networks fails in two ways. First it is not a plausible algorithm for running in continuous time as has to have fixed end points for both the forward and the backward pass. It also fails for online setting (future predictions in real time) scenarios due to its incapability to receive future information for unobserved data. Figure 3.11 shows an example of the described architecture for the bidirectional recurrent neural network [15].

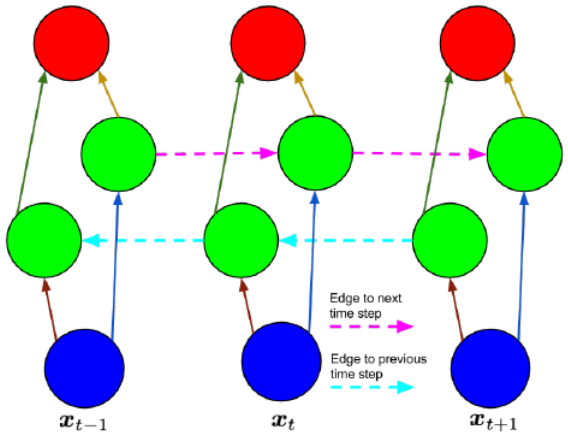

Figure 3.11. A simple BRNN with time steps give by dotted arrows. The green nodes represent the forward and backward pass at each time step of the hidden unit

3.4 Neural Machine Translator

Neural Machine Translation(NMT) is an extension built on top of the already defined architecture of the Recurrent Neural Networks. Principally, they involve two kinds recurrent neural nets - one for encoding and the other for decoding. The rudiment notion of NMT is that, given an input sequence and target sequence, the first step is to compute the probabilities of the input vector producing a vector of probabilities in the encoding phase. The output of the encoding phase which is a vector is passed to the decoding phase where the RNN in the decoding phase uses the passed vector to make predictions on the already known target sentences. The output of the decoder is then passed to a soft-max that takes the hidden state of the decoder at time step t and computes a probability vector [23]. Therefore, the role of the soft-max function is to squash the given values to produce real number in range [0 ,1] hence probabilities. The name Encoder-Decoder comes from the intuition that the first neural network running over the source sentence encodes its information as a vector of real-valued numbers, after which, the second neural network uses this information to predict the target sentence [24]. Figure 3.12 shows the architecture showing information flow in the Encode-Decode model [23].

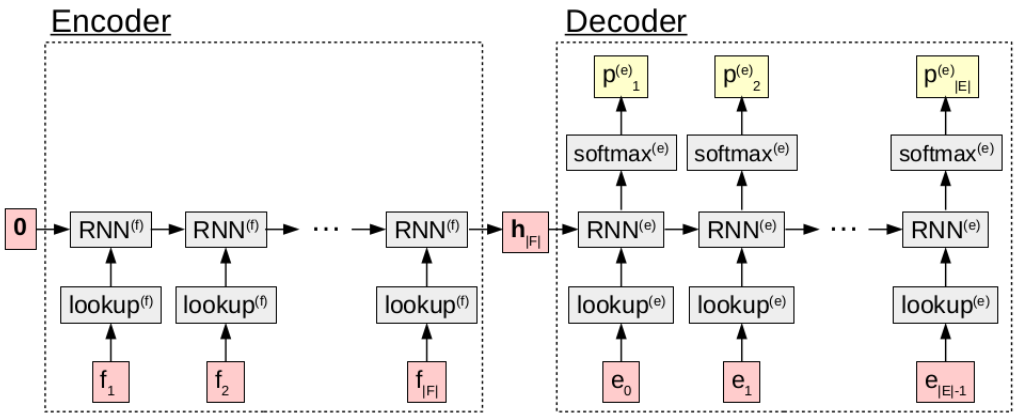

Figure 3.12. computation graph of the encoder-decoder model.

In the figure 3.12, the encoder is labelled as $RNN^{(f)}$, the decoder as $RNN^{(e)}$, and the soft-max that takes decoder output at time t and computes the probability. The computations are performed consecutively and are summarized in the equation (3.21). The first two lines of the equation looks for the embedding $m_t^{(f)}$ to calculate the encoder hidden state $h_t^{(f)}$ for the n^{th} word in the source sentence $\overline{X}$. The starting point is an empty vector $h_0^{(f)}$, and by the time we reach $h_{\overline{X}}^{(f)}$ simply means the decoder has seen all the tokens in the input sentence.

$$
\begin{aligned}
\boldsymbol{m}_t^{(f)} &= M_{\cdot,f_t}^{(f)} \\
\boldsymbol{h}_t^{(f)} &= \begin{cases} \mathrm{RNN}^{(f)}(\boldsymbol{m}_t^{(f)}, \boldsymbol{h}_{t-1}^{(f)}) & t \geq 1, \\ \mathbf{0} & \text{otherwise.} \end{cases} \\
\boldsymbol{m}_t^{(e)} &= M_{\cdot,e_{t-1}}^{(e)} \\
\boldsymbol{h}_t^{(e)} &= \begin{cases} \mathrm{RNN}^{(e)}(\boldsymbol{m}_t^{(e)}, \boldsymbol{h}_{t-1}^{(e)}) & t \geq 1, \\ \boldsymbol{h}_{|F|}^{(f)} & \text{otherwise.} \end{cases} \\
\boldsymbol{p}_t^{(e)} &= \mathrm{softmax}(W_{hs}\boldsymbol{h}_t^{(e)} + b_s)
\end{aligned}
\tag{3.21}
$$

In the decoder stage, we aim to predict the probability of token e_t at each time step. Prior to this computation, we search for $m_t^{(e)}$ using the previous token e_{t-1} . Afterwards, the decoder is run to compute $h_t^{(e)}$. The difference of the decoder from the encoder computations is that in the decoder $h_x^{(f)}$ is set as the initial value while the encoder take a vector of zeroes $h_0^{(f)}$. we finally compute the probability $P_t^{(e)}$ by using a soft-max on the hidden state $h_t^{(e)}$ [23].

Prediction in a neural machine translation follows the above defined steps. After computing the probability $P_t^{(e)}$ we can now make predictions. The probability model is hereby given by $P(\overline{Y} \mid \overline{X})$. There are several methods for making predictions from the probability model below are some few common methods [23].

1. Random Sampling: Random sampling involves randomly sampling an target sequence $\overline{Y}$ from the probability model $P(\overline{Y} \mid \overline{X})$.

2. Best Search or greedy best search: In 1-best search the notion is to find the target sequence $\overline{Y}$ that maximizes the probability $P(\overline{Y} \mid \overline{X})$ given as; .

3. N-Best Search or beam search: This algorithm is in principal similar to 1-best search with the difference here we get the first n outputs with the highest probabilities for a given model $P(\overline{Y} \mid \overline{X})$.

3.4.1 Encoder-Decoder Problems

As earlier pointed that a well trained encoder-decoder model over a large data-set should be able to make proper translations. However, in practice it cannot be guaranteed that we will often be presented with large enough data as per the model requirements in order to get a proper inductive bias - which is an appropriate model structure that enables the network to learn with limited size of data, just ample to learn from [23].

In general the encoder-decoder model presents two problems. The first is that there are long distances dependencies between tokens that are to be translated into each other. The second and the most critical problem is that the encoder-decoder models tries to store information of random, long length sentences into a hidden vector of fixed size. Therefore, since the hidden vector is of fixed size, means that all the sentences that the model learns have to be stored in this vector. For instance, if our network has the capacity to handle (process) 10 to 100 tokens then it means that all the processed tokens have to be stored in this vector and even sentences whose length are larger than 100. This can be overcome by increasing our network size. However, increasing the network size leads to another problem of needing more computational resources. In addition to the problem of increasing the network size, the number of parameters to be estimated also increases which may cause over-fitting during learning in the event of limited data.

3.4.2 Attention Mechanism in NMT

The motivation behind the attention mechanism in neural machine translation is to provide a way to align output words with the input words [25]. Consequently, instead of computing a single vector representation for a sentence, in the attention architecture each word in a sentence has its own vector. This means that long sentences will contain more vectors since the number of vectors is equivalent to number of words in the input sequence, which makes it easy for the expression of input sequences unlike the encoder-decoder mechanism which has only one fixed size vector [23].

The attention mechanism starts by creating a set of vectors having different lengths which are to be used in the model during computations. The creation of this set of vectors commences by first running the Recurrent Neural Network in both directions creating a vector for each word in the sequence [24] as:

$$\Rightarrow h_j^{(x)} = RNN(embed(x_j)\ , h_{j-1}^{(x)}) \tag{3.21}$$

$$\Leftarrow h_j^{(x)} = RNN(embed(f_j)\ , h_{j+1}^{(x)}) \tag{3.22}$$

After computations of vectors in the forward and backward direction, the single vectors are combined to form a single vector representing a single word. Subsequently, a matrix is computed containing vector representations for all the words in the input sequence.

$$H^{(\overline{x})} = concat_{vectors}(h_1^{(x)}\ , ...\ ,\ h_{\overline{x}}^{(x)}) \tag{3.23}$$

Given this generalized matrix, the aim is to compute the probabilities over the given language model vocabulary. In the attention mechanism, what happens is that we compute a value a_t which is used to computer a vector C_t by combining the columns of the matrix. Hence we get;

$$C_t\ = H^{(\overline{X})} * a_t \tag{3.24}$$

The value a_t is called an attention vector whose purpose is to make us get informed insights on the amount of attention we are putting to each word in the sequence. The larger values of attention vector indicates the importance of a word when predicting the next word in the output sequence.

The computation of the attention vector a_t is done in the decoder recurrent neural network, which is used to track the current state during the generation of output. Similar to the traditional encoder-decoder, the hidden state is a fixed continuous vector representing the previous target words $e_{(t-1)}$, initialized as $h_0^{(e)} = h_{\overline{x}+1}^{(x)}$. This then is what is used to calculate the context vector C_t which is used to make summaries of the source attention context used in choosing target word and is initialized as $C_0 = 0$.

The starting point is updating of the hidden state to h_t^e based on the word representation and context vector from previous time step as given in equation 3.25.

$$h_t^{(e)} = enc([embed(e_{t-1}) \; ; \; C_{t-1}] \; , \; h_{t-1}^{(e)}) \tag{3.25}$$

Then based on the value in the equation, we calculate the attention score as in equation 3.26;

$$a_{t,\,j} = attn \; - \; score(h_j^{(x)}, h_t^{(e)}) \tag{3.26}$$

The scoring function is some random function that takes in two vectors as inputs and outputs a score about how much we ought to focus on this particular word encoding $h_j^{(x)}$ at the time step of $h_t^{(e)}$. The attention score is then normalized by squashing with a soft-max function over the scores.

$$\alpha_t = softmax(a_t) \tag{3.27}$$

The final step is then, taking this attention vector to check the encoded representation of $H^{(x)}$ in order to create the context vector C_t for the current time step. The graph in figure 3.13 shows the graphical flow of the described computations [23].

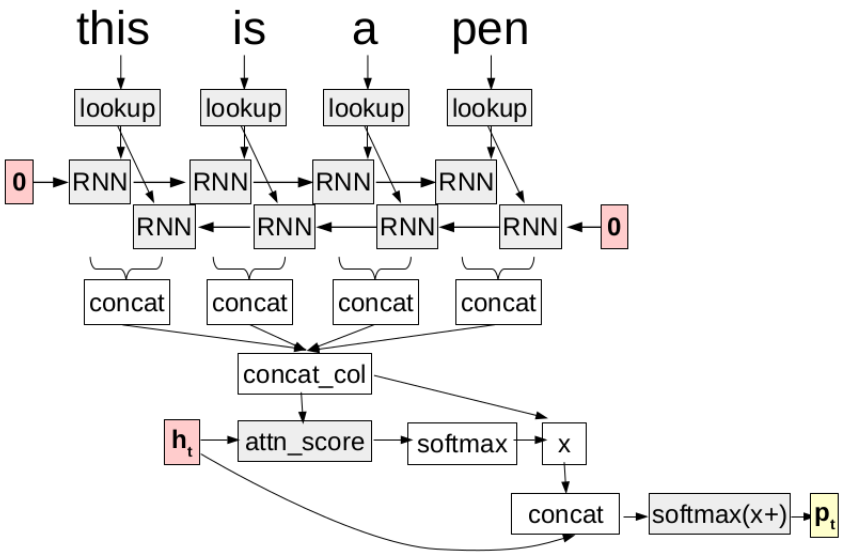

Figure 3.13. A computation graph for attention

The encoding of each word $h_j^{(x)}$ in the source sequence is treated much more directly when calculating the probabilities. This is in itself much more different from the architecture of encoder-decoder where the only thing to be considered is information about the first encoded word by passing it through in steps corresponding to the length of the input sequence. Fortunately, in the attention mechanism the source encoding is accessed through the context vector.

3.5 Statistical Corrector

Spelling correction is an important feature for any interactive service that takes input generated by users, e.g. an e-commerce web site that allows searching for goods and products. Misspellings are very common with user-generated input and the reasons why many web sites offer spelling correction [37]. Misspelled queries might be considered correct by a statistical spelling correction system as there is evidence in the data through frequent occurrences. Spellcheckers have become important due to the increase in the volume of text-based communication at work and in society on social media. Tri-grams learned from a corpus, their probabilities, minimum edit distance, and additional optimization are used in the error corrector [38]

In this dissertation, a Statistical Corrector is solely used during testing to check for typos, punctuation, and normalization of the input address tokens using some probabilistic rules. Given an input address, the Statistical Corrector checks the available vocabulary for similarity of the new tokens in the input address. This approach should be of particular interest for languages where very little annotated training data exists, although we also hope to use it as a baseline to motivate future research. The Spelling Corrector works in 4 steps. First given a word w. The spelling tries to choose the most likely spelling for the word w from a corpus of training data. For instance, given a word "hav", it should be corrected to either "Have" or "having" etc. The issue is that we are not certain for sure which is the correct word to correct to. We therefore, are trying to find the correction c, out of all possible candidate corrections, that maximizes the probability that c is the intended correction, given the original word w: Then by Bayes theorem we have to find;

$$\arg\max_{c \in candidates-space} P(c \mid w) \;\propto\; \arg\max_{c \in candidates-space} \frac{P(c)P(w \mid c)}{P(w)} \quad (3.28)$$

The above statistical expression has four parts [39];

1. The Selection Mechanism: $\arg\max$, We have choose from candidate space of words the word with the highest probability.

2. The candidate model: $c \in candidates - space$, The intuition is that we have

to consider all possible candidates in the space of candidates words. This gives us knowledge of which candidates to consider.

3. The Language Model: $P(c)$, This gives a probability the a word appears in our vocabulary. Basically this is just the probability;

$$probability(c) = \frac{The\ number\ of\ occurrences\ of\ c\ in\ the\ vocabulary}{The\ total\ number\ of\ words\ in\ the\ vocabulary} \quad (3.29)$$

4. The Error Model: $P(w \mid c)$, This is the probability of making a mistake either in spelling or typing where instead of the intended word c one types w. The error model uses a distance metric to check for the probability of each word w from the intended word. Therefore, the candidate will be the word which is close in distance from the typed word.

Chapter 4

Implementation

This Chapter describes our implementation details for system – for both the Hidden Markov Model and the Recurrent Neural Network. The details include diagrams for the system architecture design considerations.

4.1 Design

Earlier, we described that the system is divided into two tasks – address segmentation using Hidden Markov Models and address normalization using Recurrent Neural Networks augmented with Statistical Corrector. In our system, address normalization is considered a precursor of address standardization. This step generally consists of identifying the component parts or attributes of an address so that they may be transformed into some other desired format. A normalization algorithm must attempt to identify most likely attributes to associate with each component input address with respect to the gold standard for the system. That said, it is clear to see that address normalization is critical to the address cleaning process. Hence, without identifying which piece of text corresponds to which address attribute, it is impossible to subsequently transform them between standard formats or use them for feature matching. A range of normalization approaches may be used, however we chose to use Recurrent Neural Network augmented with a probabilistic Statistical Corrector. Figure 4.1 shows the overall flow with comparison to the traditional rule based approach and the machine learning approach.

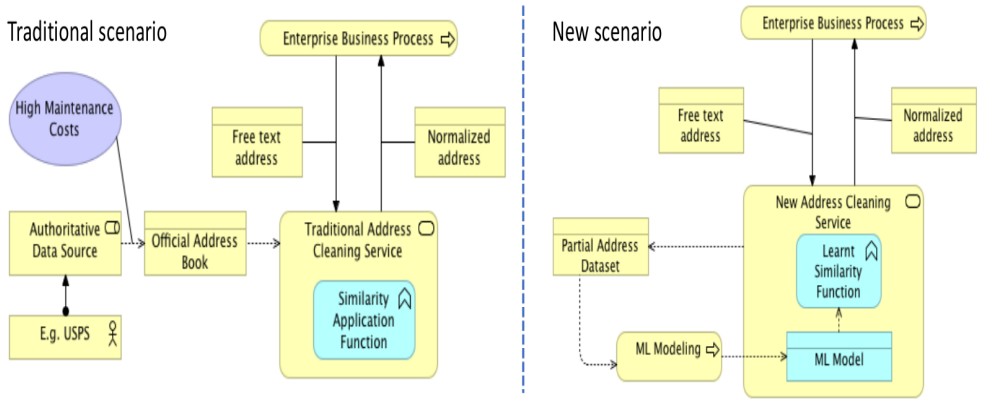

Figure 4.1. simple schematic flow of rule based in comparison to the ML approach

In our approach, the similarity function is realized by means of Machine Learning Model trained on the data-set of known addresses. We already articulated that this data-set is not complete and it does not cover all possible addresses and can be extended with the usage of the system. On the contrary, the traditional rule based approach consists of all possible addresses there can ever exist. Figure 4.2 gives details of the work flow of the our scenario (approach). The diagram shows three steps followed in the implementation of project, however, we excluded the Address Validity Check step which is intended to give a confidence on the existence of the address.

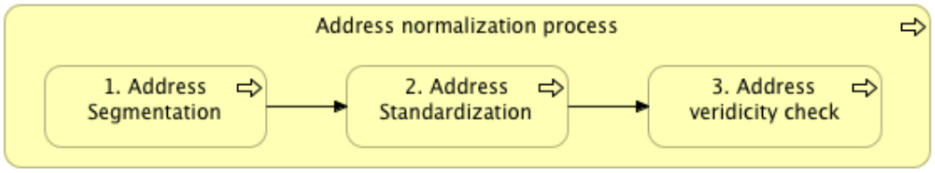

Figure 4.2. Flow of Address normalization project flow

4.2 Performance Measures

we applied different kinds measurement metrics for the Hidden Markov Models and the Neural Machine Translation models. In the Hidden Markov Models we used the standard performance measures – the precision, recall, accuracy and the f-measure. The details of implementations of these measures are given below with the abbreviations; TP defines the total number of true positive words, FN defines number of false negatives and FP as the total number of false positive words.

$$precision(P) = \frac{TP}{TP + FP} \tag{4.1}$$

$$Recall(R) = \frac{TP}{TP + FN} \tag{4.2}$$

For the calculation of F-measure, we use both the precision and the recall in order to provide a balanced summary of recall and precision.

$$F - Measure = \frac{(\beta + 1.0) * P * R}{\beta^2 * P + R} \tag{4.3}$$

The parameter, β represents the relative weight of recall to precision. However, in our approach we used the F_1, therefore, we had to set the value of β to one getting the formula as;

$$F_1 = \frac{2 * P * R}{P + R} \tag{4.4}$$

In the case where precision is equal to recall and both have value of zero then will give a value of zero.

In the case of translation, there might be multiple correct translations for a sequence [2], which may provide many flaws to the kind of performance or evaluation metrics to use in such circumstances. It might happen also that a labelled data set contains many multiple reference translations for each example. Comparing against such to a gold standard is more fraught than applying standard performance measure to binary classification problems.

The obvious metric used in natural language models with multiple references is the BLEU score and Perplexity which we used for evaluation of the Neural Machine Translations. BLEU score is a geometric mean of the n-gram for values of n between 1 and some upper limit N. The typical value of N in practice is 4 which has been shown to maximize the agreement with human raters [2]. The BLEU score includes a brevity penalty B to counter for the high precision scores incurred for short translations. The equations 4.6 shows a way to calculate the BLEU score. C in the equation represents the average length of the candidate translations and r the average rate of reference translations.

$$B = \begin{cases} 1, & \text{if } c > r. \\ e^{(\frac{1-r}{c})}, & \text{if } c \leq r. \end{cases} \tag{4.5}$$

The calculation of BLEU score then follows from brevity,

$$BLEU = B * exp(\frac{1}{N} \sum_{n=1}^{N} logP_n) \tag{4.6}$$

where p_n is the modified n-gram precision, which is the number of n-grams in the candidate translation that occur in any of the reference translations, divided by the total number of n-grams in the candidate translation. This is called modified precision because it is an adaptation of precision to the case of multiple references. BLEU scores are commonly used in recent papers to evaluate both translation and captioning systems. While BLEU score does appear highly correlated with human judgments, there is no guarantee that any given translation with a higher BLEU score is superior to another which receives a lower BLEU score.

4.3 Data Collection

The data we used was given as raw data consisting of free unstructured text and the structured text segmented into different token names. The structured addresses consisted of nine segments as; (“street type”, “street name”, “house number”, “postal code”, “city name”, “province”, “country”). In our approach, we opted to remove country in order to remove ambiguity, since we only used Italian addresses in the research work and accordingly the value for country is static.

The data collected had many forms and varying degrees of completeness. Low quality and incompleteness in training data presents one of the most critical problems when the system specification requires accurate address data. Thus, the aim of address data collection is to remove ambiguities presented as basic elements or attributes of address data. A good practice is to garner for the best possible information at the beginning of the process. In simple terms, the collection activity at the source of data makes other steps easier provided the data collected is in gold standard.

As already described, the features that we considered for a complete address, any address that contains a full list of the features we have mentioned will be considered to be a gold standard in our description. For example, an address given below is gold standard since it contains valid information in each of the possible attribute fields and indicates enough information to help with training of the system.

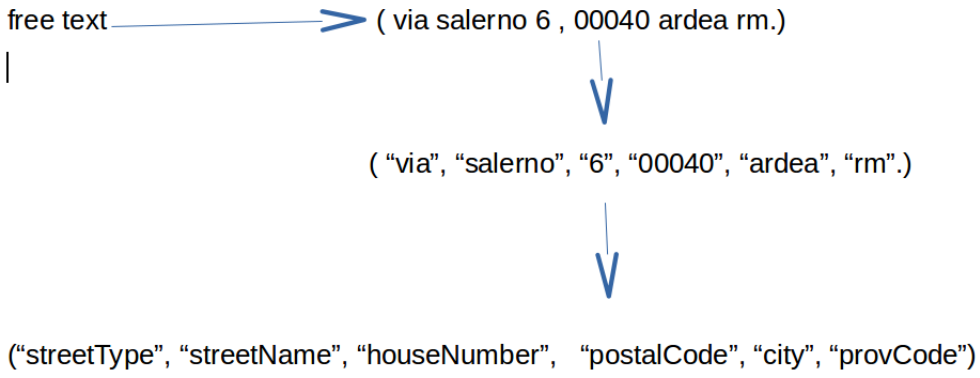

Figure 4.3. Sample work-flow for gold standard address

Figure 4.3 shows an example of gold standard data where each token of the free text is mapped to the corresponding segments. In such a scenario the address is considered to be complete.

4.4 Data Processing

The first data we received consisted of 124,000 addresses for training of our models. As described in data collection part, only well tagged data can be used to train the models to achieve reliable and better performance. We describe the schematic diagram for flow of data processing in figure 4.4.

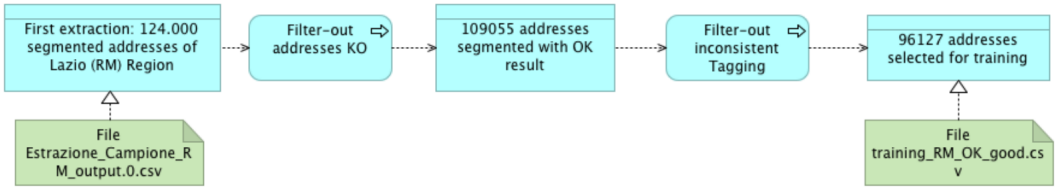

Figure 4.4. A description a flow for data processing

All samples tagged with word "OK" were excluded from the training examples and never used in implementation. Further inconsistent addresses were removed. For instance, text addresses with words not tagged in any other fields (Segments). In this stage the free text addresses where checked against there corresponding already segmented data to check for conformity. This means a reduction in the number of training examples that we further considered for training since data which had inconsistencies was removed.

4.5 From Data Generation to Model

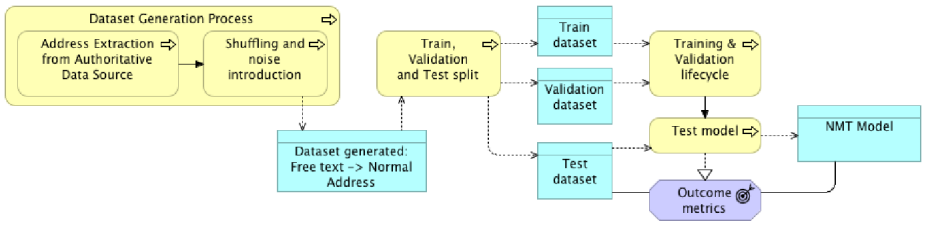

Figure 4.5. A description a flow from data generation to model

The implementation of the NMT model followed the process represented in figure 4.5 above. The Data-set Generation Process plays a key role for the achievements of the Outcome Metrics of the Model on the Test set. Starting from an extraction of standardized addresses from an Authoritative Source, we modified the free text form following a defined set of rules that simulate arbitrary human writing like: delete keywords, add abbreviations, switch the order of specific words, etc...

The high result achieved in the test demonstrates that the model was able to learn such rules. In order to demonstrate the applicability of this approach for real usage, the next steps should be:

- Test of the model on free text address really written by the end-users and collect additional data
- Adapt the model to the underlying patterns of real user writing with a scheduled re-training of the model

4.6 Experimental Results

In this section we present the experimental results as regards to the segmentation via the Hidden Markov Model and standardization via the recurrent neural networks augmented with a probabilistic corrector. The data-set was split into training and validation sets. 20% of the data was randomly selected as the validation set while the remainder was used for training of the systems. The splitting of the data was done after shuffling to facilitate randomness.

4.6.1 Hidden Markov models Learning and Inference

The training of Hidden Markov Model involved creation of states to tags and a dictionary of all the possible observations. Each state of the Hidden Markov Model was given a tag. The mapping of states to tags is shown below in simplified dictionary as;

{'houseNumber': 0, 'streetType': 1, 'streetName': 2, 'postalCode': 3, 'city': 4, 'provCode': 5, 'area': 6}

The possible observations were represented as numbers in a vocabulary dictionary. Each new observation not found in the vocabulary was given an "UNK" value while the vocabulary dictionary was incremented with the new observation.

The tags dictionary and the vocabulary dictionary facilitated the computation of the matrices A and B of the Hidden Markov Models as already explained. The experimental results for the Hidden Markov Models are shown in the tables that follow.

TAG	F1-score	Precision	Recall	Accuracy
houseNumber	0.9998383	0.99973865	0.99993797	
streetType	0.9988711	0.99834011	0.99940265	
streetName	0.9983859	0.99954109	0.99723337	
postalCode	1	1	1	
city	0.99690833	0.99394981	0.99988451	
provCode	1	1	1	
area	0.02241594	1	0.01133501	
Average Score	**0.859488509**	**0.998795666**	**0.858256216**	**0.9986238**

Figure 4.6. Results for training data for Hidden Markov Model

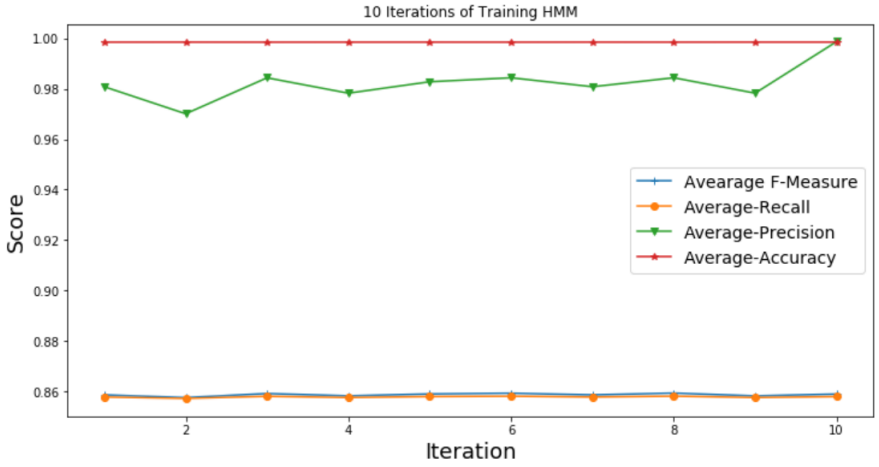

Figure 4.7. Plots of average train scores for F-Measure, Precision, Recall, and Accuracy

TAG	F1-score	Precision	Recall	Accuracy
houseNumber	0.99941002	0.99889106	0.99992952	
streetType	0.9989961	0.99857841	0.99941415	
streetName	0.99821832	0.99953988	0.99690025	
postalCode	1	1	1	
city	0.99678359	0.99369349	0.99989296	
provCode	1	1	1	
area	0.02752294	1	0.01395349	
Average Score	**0.858256216**	**0.99867183**	**0.85858434**	**0.99860726**

Figure 4.8. Results for testing data for Hidden Markov Model

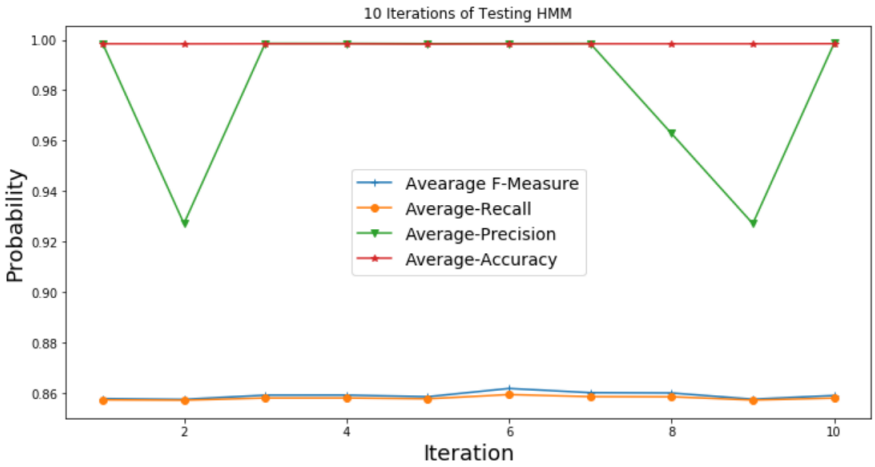

Figure 4.9. Plots of average test scores for F-Measure, Precision, Recall, and Accuracy

4.6.2 Discussion of HMM results

The overall results as shown in the figures 4.6 through to 4.9, shows a low score for *Area* tags. This low score can be attributed to the fact that there was missing values for the *Area* tag which resulted in data being highly unbalanced for the *Area* tags. Additionally, as can be seen in plots for training data in figure 4.7, it is clear that the HMM system maintains high score for Accuracy and Precision unlike for Recall and F-Measure scores. The scores for Recall and F-Measure have only subtle difference. Furthermore, looking at figure 4.8 for test data it is apparent that the system maintains its high accuracy in all the iterations but fails to maintain the same high score for precision similar to its score during training. However, the scores for Precision at some point during test reaches the maximum score contrary to the scores for the same during training phase. The high score is just maintained in the middle of testing phase and the curve goes steeply low after that and goes back to the maximum score. The scores for recall and F-measure shows a slight increase in the difference for testing phase when compared to the training phase difference in scores.

4.6.3 Recurrent Neural Networks Learning and Inference

For Recurrent Neural Networks model, we considered two types of model implementations – one with the standard RNN and the other with the attention mechanism. We already gave brief descriptions of these two types of models and there implementation details. Here, we give the results of the experiments conducted and show the flow of the experiments. Contrary to the Hidden Markov Models experiments, for Recurrent Neural Networks the notion was to standardize the addresses. Therefore, we only considered Natural Language Processing for street names and house numbers. These elements have a more complex standardized form that we decided to use natural language with a specific language model for postal addresses. The overall architecture for this implementation is shown in figure 4.10.

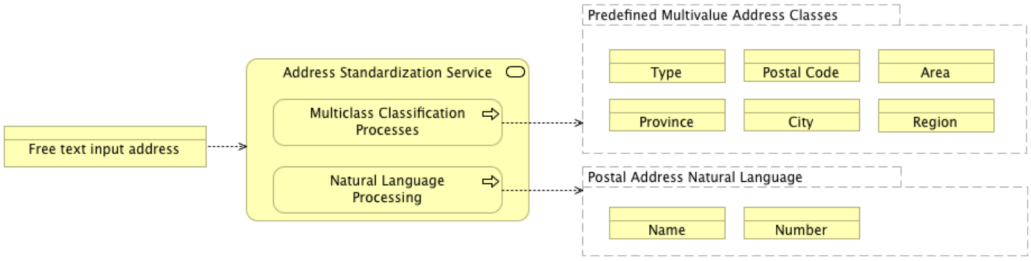

Figure 4.10. possible division of address standardization

The multi-value classification process works for elements that can be represented as belonging to a particular class with predefined set of values such as street type, postal code, area, province, city, and region. However, we did not consider this in the implementation of the system, hence we concentrate more on the Natural Language Processing mechanism.

The Recurrent Neural Network was implemented in parts two. One without the statistical corrector shown in figure 4.11 and the other with the statistical corrector shown in figure 4.12 which is important when learning of new input text.

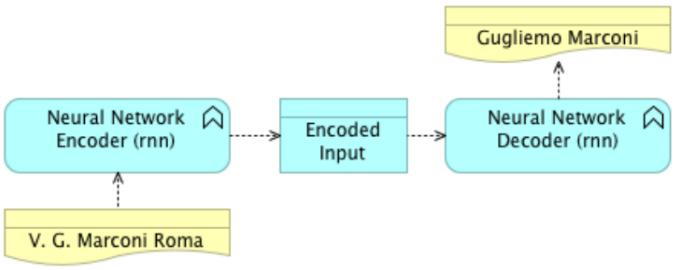

Figure 4.11. Standard Neural Machine Translation

The neural machine translation here has to learn the combined encoding and decoding functions that translate from input language, hereby - free text format of the address into target language which is the normalized version of the input text. We generated a broader set of examples from the data-set which was used for the training of the two models. We simplified the task as already explained by only including the street names and house numbers.

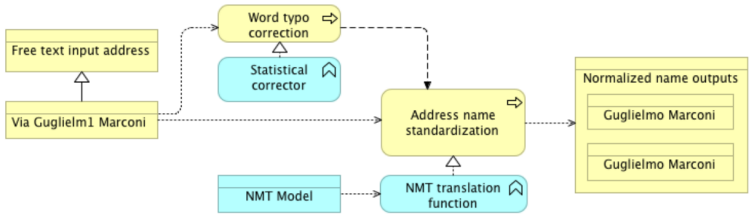

Figure 4.12. Neural Machine Translation augmented with statistical corrector

	Accuracy(%)	**Perplexity**	**Loss**	**Number of examples**
Train set	99.79	1.05	0.00072	238670
Validation set	97.26	1.33		60000

Figure 4.13. Results of the RNN with the Attention Mechanism

	Bleu	Perplexity	Number of examples
Train set	94.6	1.45	238670
Validation set	94.6	1.45	60000

Figure 4.14. Results of the Traditional RNN

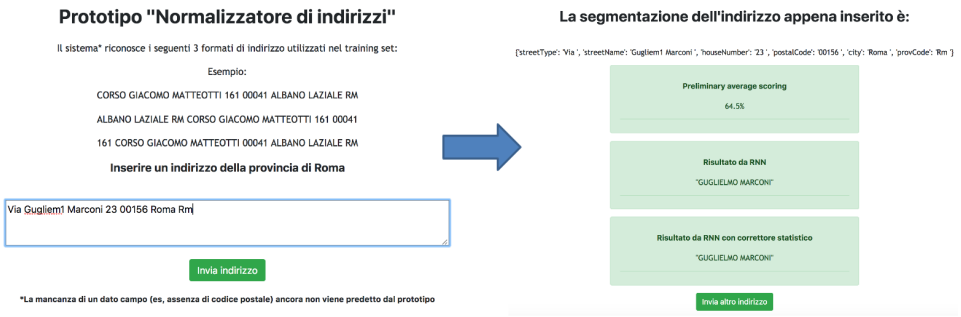

Figure 4.15. Sample of prediction results for a single unstructured text Address "Via Guglienn1 Marconi 23 00156 Roma Rm"

4.6.4 Discussion of RNN results

Looking at the figures for Recurrent Neural Networks in figure 4.13, it is clearly shown that the model performed with high scores in both the training and validation phases. The ***Perplexity*** evaluation for model (Attention Mechanism) is quite low as is expected for high accuracy results gotten in both the training and validation phases. The best case scenario for the ***Perplexity*** metric is suppose to be 1. However, the model is able to get to close to 1 in both the training and validation phases which is a good indication that model is able to reproduce the test data targets.

In figure 4.13, we take a look at the results from a traditional NMT model without attention mechanism using ***BLEU*** evaluation metric. It was observed during training that this model takes quite some time to converge as compared to the NMT model with Attention Mechanism and furthermore, the prediction scores are lower than that for the Model with Attention mechanism.

Figure 4.15 shows a sample test address for test the models. The test address is **'Via Guglienn1 Marconi 23 00156 Roma Rm'**. However, the corpus has name Guglienn under which the model was trained on. Therefore, the statistical Corrector checks in the corpus to find a similar word (token) which is close in distance to the

name **"Guglienn1"**. As can be seen in the figure the prediction of such an address using RNN with and without augmentation with Statistical corrector. Both types of RNN predict the same name **"Guglielmo Marconi"** which is correct. However, for complicated addresses the RNN without statistical correct fails to predict the correct address names as it does not check for word similarity in the corpus to choose the token which is close in distance to the test case. Additionally, it is apparent that the Hidden Markov Model is also able to segment the addresses into its constituent token names correctly as expected from the test scores in figure 4.15.

Chapter 5

Conclusions

This research project's objective was to address the problem of Postal Address Segmentation and Normalization as it relates to their use cases in organizations. The main objective was to understand if it is possible to normalize postal addresses by means of a Machine Learning model trained on partial data-set. We proposed two Machine Learning Models. Segmentation of Addresses by way of Hidden Markov Models and Standardization by way of Recurrent Neural Networks augmented with statistical Corrector for checking similarity of tokens in training corpus and new tokens for prediction. Two kinds of Recurrent Neural Networks were considered, the traditional Neural Machine Translation model and the Neural Machine Translation model with the Attention Mechanism. We tested the data on different open source Machine Learning libraries - Harvard's Open Neural Machine Translator (OpenNMT-py) and Stanford's Neural Machine Translator implemented in Tensor-flow.

From all the experiments conducted and all the evidence gotten, the models give expected results. It is evident from the results that between the traditional NMT and NMT with attention mechanism, the one with Attention mechanism shows superiority as pertains to efficiency, and accuracy. Furthermore, the models are much influenced by the features used in the models. For instance, it is explicitly evident from the results of the Hidden Markov Models that different features (elements of the vocabulary) have different scores for accuracy, precision, recall and F-measure. In particular, the element Location has low score for all evaluation metrics due to few cases present in the training examples for this particular label.

The results of segmentation through Hidden Markov Model reached an accuracy of greater than 99 percent and standardization through Recurrent Neural Networks reached greater than 96 percent in accuracy. The reason for such high scores is in particular to the data-set that we used in out experiments as already explained in data generation process for our research.

In future research, we hope to consider the use of other signal extraction methods for feature extraction such as graph processing. We believe by using graphs we can obtain some level of improvement for labels which tend to portray high levels of missing data. We would like to extract solely statistical features from graphs that are generated from time series. In particular, we would like to augment time series by means of their sequential approximations, which are further transformed into a set of visibility graphs.

Bibliography

[1] Christopher M. Bishop, "*Pattern Recognition and Machine Learning*", Cambridge, 2006 edition.

[2] Zachary C. Lipton, John Berkowitz and Charles Elkan, "*Critical Review Of Recurrent Neural Networks For Sequential Learning*",University of Carlifonia San Diego, June 2015.

[3] Elliot R.J, Aggoun .L and Moore J.B, "*Hidden Markov Models*", Estimation and Control, 1995.

[4] Wikipedia, https://en.wikipedia.org/wiki/Hidden_Markov_model

[5] Daniel Jurafsky and James H. Martin, "*Speech and Language Processing*", Hidden Markov Models, September, 2018.

[6] David G. Forney JR, "*The Viterbi Algorithm*", proceedings of IEEE, Vol. 61, March 1973.

[7] Joshua T. Goodman, "*A Bit of Progress in Language Modelling*", Computer Speech and Language, Microsoft Research, 2001

[8] Dayne Freitag and Andrew kachites McCallam, "*Information Extraction With HMM and Shrinkage*", AAAI Just Research, 1999.

[9] Ikechukwu I. Ayogu, Adebayo O. Adetunmbi†, Bolanle A. Ojokohand Samuel A. Oluwadar, "*A Comparative Study of Hidden Markov Model and Conditional Random Fields on a Yoruba Part-of-Speech Tagging Task"*, Federal University of Technology Akure, proceedings of IEEE, 2017

[10] Stanley F. Chen and Joshua Goodman, "*An Empirical Study of Smoothing Techniques for language Modelling*", Computer Speech and language. Carnegie Mellon University, 1999

[11] Kevin Gurney, "*An Introduction to Neural networks*", University College London, 1997.

[12] Clement Farabet, Camille Couprie, Laurent Najman and Yann LeCun, "*Learning Hierarchical Features for Scene Labeling*", Pattern Analysis and Machine Intelligence, IEEE Transactions on, 2013.

[13] Tomáš Mikolov, Martin Karafiát, Lukáš Burget, Jan Honza Černocký and Sanjeev Khudanpur, "*Recurrent neural network based language model*", Interspeech, Brno University of Technology and Johns Hopkins University, September 2010.

[14] Michael I. Jordan. "*A parallel distributed processing approach*". Technical Report 8604, Institute for Cognitive Science, University of California, San Diego, 1986

[15] Avrim L. Blum and Ronald L. Rivest, "*Training a 3-Node Neural Network is NP-Complete*", Neural Networks: Vol 5, Number 1, MIT Laboratory for Computer Science, January 1992.

[16] Yoshua Bengio, Patrice Simard and Paolo Frasconi, "*Learning Long-Term Dependecies With Gradient Descent is Difficult*", IEEE Transactions on Neural Networks, Vol.5, No.2, March 1994.

[17] Ronald J. Williams and Jing Peng, "*An Efficient Gradient-Based Algorithm for On-Line Training of Recurrent Network Trajectories*", Northeastern University, Appears in Neural Computation, Vol.2 , pp. 490-501, 1990.

[18] Jeffreyl Elman, "*Finding Structure in Time*", University of Carlifonia, Cognitive Science Vol.14, 179-211, 1990.

[19] Ilya Sutskever, "*Training With Recurrent Neural Networks*", University of Toronto, 2013

[20] Sepp Hochreiter and Jurgen Schmidhuber, "*Long Short-Term Memory*", Neural Computation 9(8):1735-1780, 1997.

[21] Mike Schuster and Kuldip K. Paliwal, "*Bidirectional Recurrent Neural Networks*", Transactions on Signal Processing, Vol. 45, No. 11, IEEE, November, 1997.

[22] Wojciech Zaremba and Ilya Sutskever, "*Learning to Execute*", New York University and Google Research, February 2015

[23] Graham Neubig, "*Neural Machine Translation and Sequence to Sequence Models*", Tutorial, Carnegie Mellon University, March 2017.

[24] Minh-Thang Luong Hieu Pham Christopher D. Manning, "*Effective Approaches to Attention Based Neural Machine Translation*", Stanford University.

[25] Philip Coen, "*Neural Machine Translation*", speech and language processing, Johns Hopkins University, september 2017.

[26] Borkar V., Deshmukh K., and Sarawagi S., "*Automatic segmentation of text into structured records*", in ACM SIGMOD 2001, (Santa Barbara, California, USA), 2001.

[27] Leek T., "*Information extraction using Hidden Markov Models*" Master's thesis, UC San Diego, 1997.

[28] Wojciech Zaremba and Ilya Sutskever, "*Learning to Execute*", New York University and Google Research, February 2015.

[29] Ilya Sutskever, Oriol Vinyals and Quoc V. Le, "*Sequence to Sequence Learning with Neural Networks*", Google Research, December 2014.

[30] Rico Sennrich, Barry Haddow and Alexandra Birch, "*Neural Machine Translation of Rare Words with Subword Units*", University of Edinburgh, June, 2016.

[31] Leon Derczynski, Diana Maynard, Giuseppe Rizzo, Marieke van Erp, Genevieve Gorrell, Raphaël Troncy, Johann Petrak and Kalina Bontcheva. "*Analysis of named entity recognition and linking for tweets*", Information Processing and Management, Science Direct 2015.

[32] Mónica Marrero, Julián Urbano, Sonia Sánchez-Cuadrado,Jorge Morato, Juan Miguel and Gómez-Berbís, "*Named Entity Recognition: Fallacies, Challenges and Opportunities*",University Carlos III of Madrid,

[33] Lhioui Chahira, Zouaghi Anis and Zrigui Mounir, "*A Rule-based Named Entity Extraction Method and Syntactico-Semantic Annotation for Arabic Language*", LaTICE Laboratory, The Third International Conference on Big Data, Small Data, Linked Data and Open Data, 2017.

[34] Yunita Sari, Mohd Fadzil Hassan and Norshuhani Zamin, "*Rule-based Pattern Extractor and Named Entity Recognition: A Hybrid Approach*", Universiti Teknologi PETRONAS, VOL.978-1-4244-6716-7 IEEE 2010.

[35] Rayner Alfred, Leow Chin Leong, Chin Kim On and Patricia Anthony, "*Malay Named Entity Recognition Based on Rule-Based Approach*", International Journal of Machine Learning and Computing, Vol. 4, No. 3, June 2014

[36] Christopher Bryant and Ted Briscoe, "*Language Model Based Grammatical Error Correction without Annotated Training Data*", Alta Institute, University of Cambridge.

[37] Saša Hasan, Carmen Heger and Saab Mansour, "*Spelling Correction of User Search Queries through Statistical Machine Translation*", Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, eBay Inc.

[38] Frida Mjari and Maria C. Keet, "*A statistical approach to error correction for isiZulu spellcheckers*", University of Cape Town, 2017.

[39] Peter Norvig, "*How to Write a Spelling Corrector*", https://norvig.com/spell-correct.html, Google, 2007.