UNIVERSIDADE FEDERAL FLUMINENSE

WEMERSON PASTOR DE OLIVEIRA MARINHO

Word101 - A Compact Encoding of Words using Character Level Information applied to Convolutional Neural Network Text Classification

NITERÓI

2018

UNIVERSIDADE FEDERAL FLUMINENSE

WEMERSON PASTOR DE OLIVEIRA MARINHO

Word101 - A Compact Encoding of Words using Character Level Information applied to Convolutional Neural Network Text Classification

Dissertation presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfilment of the requirements for the degree of Master of Science. RESEARCH AREA: Systems and Information Engineering.

Advisor:
LUIS MARTI OROSA

Co-Advisor:
NAYAT SANCHEZ-PI

NITERÓI

2018

WEMERSON PASTOR DE OLIVEIRA MARINHO

Word101 - A Compact Encoding of Words using Character-Level Information applied to Convolutional Neural Network Text Classification

Dissertation presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfilment of the requirements for the degree of Master of Science. RESEARCH AREA: Systems and Information Engineering.

Approved in July, 2018.

Ficha catalográfica automática - SDC/BEE

M337w Marinho, Wemerson Pastor de Oliveira
Word101 - A Compact Encoding of Words using Character Level Information applied to Convolutional Neural Network Text Classification / Wemerson Pastor de Oliveira Marinho ; Luis Marti Orosa, orientador ; Sanchez-Pi Nayat, coorientadora. Niterói, 2018.
69 f. : il.

Dissertação (mestrado)-Universidade Federal Fluminense, Niterói, 2018.

DOI: http://dx.doi.org/10.22409/PGC.2018.m.22111884863

1. Classificação de dados . 2. Algoritmo computacional . 3. Rede neural . 4. Produção intelectual. I. Título II. Orosa,Luis Marti , orientador. III. Nayat, Sanchez-Pi, coorientadora. IV. Universidade Federal Fluminense. Escola de Engenharia.

CDD -

Bibliotecária responsável: Fabiana Menezes Santos da Silva - CRB7/5274

WEMERSON PASTOR DE OLIVEIRA MARINHO

Word101 - A Compact Encoding of Words using Character-Level Information applied to Convolutional Neural Network Text Classification

Dissertation presented to the Computing Graduate Program of the Universidade Federal Fluminense in partial fulfilment of the requirements for the degree of Master of Science. RESEARCH AREA: Systems and Information Engineering.

Approved in July, 2018.

Niterói

2018

For each human being who devoted a moment of his life to teach me something.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my wife Karla Marinho for being my unconditional partner. Her love, patience and inspiration push me to finish this project.

I would not have completed this task without the valuable guidance and support of my advisors, Prof. Nayat Sanchez-Pi and Prof. Luis Martí. Every time that I came to you with a broken solution or was completely lost, I found some good ideas to try and encouragement to continue. I appreciate specially every moment that you dedicated to make this text and the papers that we wrote together a little less amateur despite the mess that were its first versions. It was an honour and a pleasure be guided by you.

Forever shall I remain indebted to my parents João Pastor e Maria Gorete, for being my examples of work ethic and dedication.

I would like to thank Prof. Cristina Nader Vasconcelos for introduce me to the Deep Learning field and present me the topics of this work in her classes.

I gratefully acknowledge the support of NVIDIA Corporation with the donation of a GPU used for this research and CAPES for the award of a research scholarship.

Resumo

Esta dissertação apresenta uma nova representação de texto em forma de tensores baseadas em técnicas de compressão de informação que assinalam códigos mais curtos aos caracteres mais frequentemente utilizados. Esta representação é independente de linguagem, não necessita de pré-treinamento e produz códigos sem perda de informação. Adaptada para se aproveitar da morfologia das palavras, é capaz de representar prefixos, conjugações e inflexões com vetores similares, sendo capaz de representar mesmo palavras não constantes no conjunto de textos treinamento. Por ser compacta, porém esparsa, é ideal para acelerar os tempos de treinamento utilizando-se de bibliotecas de processamento tensorial. Como parte deste trabalho de pesquisa, mostramos que esta técnica é especialmente eficiente se utilizada em conjunto com Redes Neurais de Convolução (CNN) para classificação de textos no nível do caractere. Resultados experimentais mostram que ela reduz drasticamente o número de parâmetros a serem analisados, resultando em uma acurácia de classificação competitiva em apenas uma fração do tempo que seria gasto em representações um por linha, possibilitando treinamento em equipamentos mais simples.

Palavras-chave: classificação de textos, codificação de palavras, redes neurais profundas.

Abstract

This dissertation puts forward a new text to tensor representation that relies on information compression techniques to assign shorter codes to the most frequently used characters. This representation is language-independent with no need of pre-training and produces an encoding with no information loss. It provides an adequate description of the morphology of text, as it can represent prefixes, declensions, and inflections with similar vectors and are able to represent even unseen words on the training dataset. Similarly, as it is compact yet sparse, is ideal for speed up training times using tensor processing libraries. As part of this research, we show that this technique is especially effective when coupled with convolutional neural networks (CNNs) for text classification at character-level. Experimental results show that it drastically reduces the number of parameters to be optimized, resulting in competitive classification accuracy values in only a fraction of the time spent by one-hot encoding representations, thus enabling training in commodity hardware.

Keywords: text classification, word embedding, word encoding, deep pyramidal network.

List of Figures

List of Tables

Abbreviations and Acronyms

BOW	:	Bag of Words;
CBOW	:	Continuous Bag of Words;
CNN	:	Convolutional Neural Network;
GLOVE	:	Global Vectors;
GPU	:	Graphics Processing Unit;
GRU	:	Gated Recurrent Unit;
LSTM	:	Long Short Term Memory;
SGD	:	Stochastic Gradient Descent;
SNAP	:	Stanford Network Analysis Project;
RESNET	:	Deep Residual Networks;
RNN	:	Recurrent Neural Networks;
TF-IDF	:	Term Frequency-Inverse Document Frequency;
WORD2VEC	:	Word to Vectors;

Contents

Chapter 1

Introduction

Document classification is one of the principal tasks addressed in the context of *Natural Language Processing* (SEBASTIANI, 2002). It implies associating a document —or any text fragment, for that matter— with a category or label relying on their content. The increasing availability of texts in digital form, especially through the *Internet*, has called for the development of statistical and artificial intelligence tools for automating this process. *Spam detectors*, *Sentiment analysis*, *News archiving*, among many others, demand high-quality text classifiers.

There is a broad range of approaches to document classification (see, for example, (SEBASTIANI, 2002; AGGARWAL; ZHAI, 2012; HOTHO; NRNBERGER; PAA, 2005; KOSALA; BLOCKEEL, 2000)). An important portion of them relies on a representation that handles words as the atomic element of text. Consequently, those methods carry out their analysis through statistics of words occurrence (ZHANG; ZHAO; LECUN, 2015). However, the variability of words and structures belonging to a language hinders the viability of this method. In this context, these models have a superior performance in specific domains and applications, where the vocabulary is or can be restricted to a relatively small number of words, possibly chosen by a specialist. Furthermore, such modeling becomes specific to a language, causing the replication process in another language to be carried out from scratch (ZHANG; ZHAO; LECUN, 2015).

In recent years, we have experienced a revolution in the machine learning with the advent of *Deep Learning* methods (GOODFELLOW; BENGIO; COURVILLE, 2016). The development of *Convolutional Neural Networks* (CNNs) (LeCun et al., 1998) coupled with the popularization of parallel computing libraries spearheaded by frameworks like *Theano* (BERGSTRA et al., 2010), *Tensorflow* (ABADI et al., 2015) and *Keras* (CHOLLET et al., 2015) that simplify general-purpose computing on graphics processing units (GPU) (MITTAL; VETTER, 2015) has been very successful in tackling image classification problem (KRIZHEVSKY; SUTSKEVER; HINTON,

2012) and is quickly becoming the state of the art of the field.

As it could be expected, these success has prompted the interest to extend the *Deep Learning* principles to the document classification domain. Some existing methods have been updated but the clear majority are still based on the tokenization of words and the inference of their statistics. *Bag of Words* (BoW) (SALTON; MCGILL, 1983) and *Word2vec* (MIKOLOV et al., 2013) are some of the most popular strategies.

It can be argued that the main challenge faced when replicating the success of image classification in the documents domain is representing text as numerical tensors. One popular alternative relies on the *one-hot encoding* technique (HARRIS; HARRIS, 2012) using a *Bag of words* strategy (SALTON; MCGILL, 1983). It represents the words in the text as vectors, where the term relative position is 1 and all other 0. However, in massive dataset scenarios, using this technique would generate a representation of thousands or even millions of coordinates, making its use impractical in terms of memory needed to store this vector, time required to compute its optimization and numerical issues because of the sparsity of the representation. Another problem is that typing errors or simple declensions of words in test data could not be codified if they are unseen on train data.

To address this issue, Zhang, Zhao & LeCun (2015) suggested a groundbreaking approach that considers the characters as the atomic elements of a text. In particular, they represented the text as a sequence of *one-hot encoded* characters. This encoding provides a robust, language-independent representation of texts as matrices to be used as inputs to different CNNs. Their experimental results showed that this approach was able to attain and, in some cases, improve the state of the art results in complex text classification problems. More recently, Xiao & Cho (2016) improved those results by combining CNNs with *Long Short-Term Memories* (LSTMs) (HOCHREITER; SCHMIDHUBER, 1997). In spite of that, the impact of this idea is hampered by the large computational demands of the approach, since its training can take days per epoch in relatively complex problems.

In this dissertation, building upon the work of Zhang, Zhao & LeCun (2015), we propose an efficient character-level encoding to represent texts using the knowledge of character frequency imbalance, derived from the *Tagged Huffman* (MOURA et al., 2000) information compression technique. This encoding takes into account the character appearance frequency in the texts in order to assign shorter codes to the most frequently used ones. This novel text encoding makes the idea put forward by Zhang, Zhao & LeCun (2015) more computationally accessible by reducing its training requirements in terms of time and memory.

The proposed encoding makes possible to represent larger portions of texts in a less sparse form, without any loss of information, while preserving the ability to encode any word, even those not present in the training dataset ones. In order to study the impact of this encoding, we coupled it with a deep architecture used by Johnson & Zhang (2017) on classifying texts using word representations. The experimental studies performed showed that we managed to achieve a superior accuracy than Zhang & LeCun (2015) and very competitive with others models that use words representation even using only character composition of words as input. Furthermore, we show that this strategy is more computational efficient, comparing with Zhang & LeCun (2015) and Conneau et al. (2016), that uses character as inputs.

Our main contribution is to show that this novel character-level text encoding produces a reduced input matrix, leading to a substantial reduction in training times while producing comparable or better results in terms of accuracy than the original approach by Zhang, Zhao & LeCun (2015). This opens the door to more complex applications, the use of devices with lower computational power and the exploration of other approaches that can be coupled with this input representation.

1.1 Motivation

The initial intent of this research was applying new advances in Deep Learning to the field of *Intelligent Tutoring Systems*(FREEDMAN; ALI; MCROY, 2000).

Online individualized and independent learning is an important aspect of modern Internet. *Massive online open courses (MOOCs)* have shown themselves as a revolutionary learning tool. One salient issue that hampers MOOCs is the lack of an adequate method for adapting the learning material and teaching pace to the particular needs and characteristics of a student. Studies carried out by BLOOM (1984) in different schools and levels of schooling have shown that the average performance of a student guided by an individual tutor is about 2 standard deviations above the average result of a conventional class of 30 students, that is, 50% of students under mentoring has results better than 98% of conventional students. However, since the costs of a one-to-one education are high and unfeasible for mass-scale use, the search for alternatives that approximate the performance of individual tutoring has become an area of intense research, particularly with the use of computer systems.(PIECH et al., 2015; PAVLIK; CEN; KOEDINGER, 2009; PANE et al., 2014)

Part of the success of individual tutoring can be attributed to the teacher's ability to tailor content to the student's current skill level and to insist on the subject until it is perceived that the student has mastered it. It can be stated that this is a rather intuitive process for an experienced teacher. However, simulating this competence with computational models is a complex task. In particular, it is necessary:

1. to infer the domain of a content
2. to infer the difficulty of that content for the general population,
3. to infer the ability of the student in that domain,
4. to seek strategies to adapt both aiming at the growth of the student's proficiency.

The main problem is that cognitive ability is a latent property of the individual, not directly measurable by its very nature. Throughout history, the most popular way of estimating this property has been through the application of exams and questionnaires. One of the objectives of the use of Intelligent Tutoring Systems is to allow the personalization of teaching, that is, to know the level of ability of the student and adapt the content in a way that does not waste time working already dominated contents or even frustrate the student presenting him items to which he is not yet ready for.

Traditionally, the technique used for this task is called *Knowledge Tracing*, a term used for the task of modeling the knowledge of a student using his errors and correct answers throughout the exercises, aiming to predict the performance of this student in future interactions. Such knowledge is used to individualize the training sequence of a student, especially in *Intelligent Tutoring Systems*. Formally, *Knowledge Tracing* is a task to predict performance on an interaction $x(t+1)$, given a sequence of interactions $x(0), x(1), ..., x(t)$. The interaction has the form $x(t) = (q(t), a(t))$ where $q(t)$ is the identifier of the subject evaluated and $a(t)$ is the result of the interaction being 0 to miss 1 and hit.

In Corbett & Anderson (1995) pioneering approach, a *Bayesian Knowledge Tracing (BKT)* algorithm is applied. It goes by aggregating the sequence of hits and misses of a student, a random apriori chance of success and a probability of accidental errors. Relying on that information it tries to predict in the hidden states the probability of success of a next learning item, considering that at a value above a threshold the content is dominated.

In a recent paper, a new technique was introduced with the use of Deep Learning: *Deep Knowledge Tracing (DKT)* (PIECH et al., 2015). DTK relies in on *Recurrent Neural Networks* (RNN) and *Long Short Term Memory (LSTM)* neural networks. In the reported experiments, DKT achieved significantly better results than the widely-used *Bayesian Knowledge Tracing (BKT)* method. The methodology is able even to uncover the latent structure between the concepts, which is very useful to facilitate the optimization of curricula.

In the DKT, for each question, the input is the student's performance and the identifier of the subject of the question. The encoding of this input is performed so that the coordinate of the vector corresponding to the question is given the value 0 or 1 and the remaining null (one-hot encoding). The prediction error for each of the answers in all other questions is calculated by minimizing the cross-entropy between the vectors. The algorithm iterates until all questions and all students are presented to the network.

The underlying idea is that the neural network approach learns the dependencies between two or more questions in a series of answers attempts made by a set of students, correlating subjects that are missed together and what kind of student misses which kind of question, beyond infer the question difficulty based on the general performance of students. It is even possible that the generality of the neural network architecture understands that the student has forgotten a subject once it starts to degrade his performance on the last attempts. Generally, the degree of freedom of these neural networks is quite large, which makes them learn much more complex representations than those mentioned, as long as they lead to a better prediction.

The main criticism of the use of DKT is that modifications in BKT would have similar performance, maintaining some interpretability of the data, something that is impracticable when observing the hidden layer of a neural network (KHAJAH; LINDSEY; MOZER, 2016). Another criticism is that structured models such as BKT variations may be easier to train, requiring less parameter tuning than DKT, and that hardware demands may become prohibitive when tens of thousands of items are used (WILSON et al., 2016). For further information, we did an extensive review of the literature available methods of infer student proficiency and published the results in (MARINHO; MARTÍ, 2016).

We understand the critics of using *Deep Learning* to *Knowledge Tracing* in the approach proposed by Piech et al. (2015) but we see on it a better opportunity to incorporate others meta-data relevant to the personalization process of *Intelligent Tutoring Systems*. By its own ability to deal with a variety of input data, uses of *Deep Learning* have demonstrated to be robust to a diversity of tasks, some of them very complex. We believe that it deserves more studies and could be a promising alternative to BKT or traditional strategies based on methodologies like Item Response Theory - IRT (BAKER, 1985).

But in our view, DKT has two main deficiencies:

- **DKT needs a extensive amount of human annotation to recognize a domain for each question before we even could inference any student proficiency.**

  Before we apply this methodology, we need to segment each question in a list of domain or competence, relying on human labor to classification it into disjoint sets. Different questions statements in the same class or domain are treated as equal. This approach has a disadvantage of needing huge and expensive human intervention before we could make inferences about the student performance. Furthermore, classifying questions in disjoint domains and treat them as equal could miss important cognitive nuances of a question and ignore interdisciplinary questions statements, a trend in modern education.

- **DKT need a massive pool of previous of attempts to fitting its parameters.**

  As usual in approaches based in *Deep Learning*, Neural Networks have strong results when they are trained in a massive amount of data, alleviating the overfitting problem. This characteristic prevents it application on small classrooms with less than a hundred students, because we will need many attempts to optimize its parameters. However, that is the reality of the majority of schools, making this approach suitable only for big conglomerates of schools with many students or to MOOCs.

We think that incorporating inputs of the statement of the question could help to alleviating both problems.

First, we believe that we do not need to restrict ourselves to a sequence of questions with human annotation subjects. Recent advances in the use of deep neural networks in *Natural Language Processing* make possible to classify texts using a supervised learning approach and maybe infer difficulty directly from the text of the statement of each these questions, allowing much more generality in its applications.

Second, by having the opportunity to compare similar statements, *Deep Learning* methods could extrapolate the difficulty levels of a new question statement or relate it to the difficulty to a group of similar questions. This could be determinant to allow uses in smaller classrooms, once we can train our Neural Network in massive available repositories of attempts, like public exams, and use its parameters to evaluate a new statement that we want to present to the student. Using information available in public exams have many benefits over using classroom data. The probability of cheating is reduced because usually there are some form of oversight. Furthermore, generally the high number of of applicants and its diversity of backgrounds represent more the population in general, giving more confidence into infer the level of difficulty a question. Differently, results obtained from attempts generated in advanced/deficient classrooms or that have the guidance of a teacher in its execution could produce completely biased outputs.

To make an attempt of attacking this problem, collecting data was essential step to this project.

Consequently, we have organized a comprehensive dataset of different examination results from public data. We have created a database of approximately 100,000 university entrance exam questions from several Brazilian universities annotated by area of domain. We also obtained a 10-year dataset of the FUVEST entrance exam data, which is annotated with the hit rate of each question by the candidates of this traditional entrance exam. Similarly, we obtained from the Brazilian Ministry of Education the sequence of responses of each individual evaluated in the *Exame Nacional do Ensino Médio (ENEM)*, the national examination of high school students that allows them to enter public universities, constituting of 19 years and assesses from 5 to 9 million candidates/year.

The original intent was using these data and techniques of *Deep Learning* to train neural networks able to predict from the text of the statement of the question its difficulty and domain area and use that information together with a sequence of responses of a student to related questions, measuring their ability to respond to this new question statement. These extend the approach of Piech et al. (2015) since the question presented to the student need not be in our training dataset, possibly a completely new statement in classic cold start problem. The idea is that the architecture could compare the properties of the text and infer the proficiency of the student based on the sequence of interactions that he already has with similar question statements or subjects in the past.

The main problem faced was extracting the relevant texts of these questions statements and represent it into a numerical tensor, allowing the use of *Deep Learning*. The language where they were written is Portuguese, and there are not good word embeddings for that language available for a *Word2vec* approach (MIKOLOV et al., 2013). The number of possible words related to these texts is extensive, so we could not use a one-hot representation like *Bag-of-words* (SALTON; MCGILL, 1983; JONES, 1972) without discarding many words. Furthermore, by its own nature, many of these questions have equations and formulas since Mathematics, Physics and Chemistry are disciplines of many of these exams and we could not discard that information since is relevant to measure the difficulty or recognizing the domains of subject.

We became aware of the work of Zhang, Zhao & LeCun (2015) and tried to apply it to our database of questions. The great advantage of this approach to our problem is that since it is character-based, so we could better deal with mathematical expressions, formulas and chemical equations and do not need a good word embedding repository in Portuguese. Furthermore, typos or even encoding errors present in our database could be deal by the architecture seamlessly.

In preliminary studies, we segmented our database of 100k questions in 62 classes, with subjects as diverses as Polynomial Equations in Mathematics, Hydrostatics in Physics, Atomic Theory in Chemistry, Word War II in History, Climate in Geography and Verbs in Portuguese. Using the approach of Zhang, Zhao & LeCun (2015) we could recognize the area of domain of a question based only on its text statement with 61.13% of accuracy. This was an encouraging result since the random choice have a $\frac{1}{62} = 1.61\%$ chance. But the time to execute this algorithm is painfully slow and we starting to think if there were not a better way to do this.

Since the dimensions of the input matrix is directly correlated with the time spent on training, we directed our efforts to find a smaller text representation into tensors. At first we thought to use some kind of compressed representation to reduce the size of each character in input to some number minor that the 69 in a one-hot encoding style used by Zhang, Zhao & LeCun (2015) . Studying classic compress encodings available in the literature, as Huffman (HUFFMAN, 1952), and more modern ones like Moura et al. (2000) and Brisaboa et al. (2003), we realized that if we could concatenate codes that were not a subset of other codes, we could represent a complete word directly. This could allow reducing even more the dimensions of the input matrix because documents have significantly less words then characters.

After many failed attempts, we developed the encoding strategy approach of this proposal, and to our delight, it works! We named it *Word101* since it uses digits 0 to represent the signal and digit 1 to tag the begin and end of a character-code. It presented a promising performance in preliminary tests, good properties to be applied in repository of texts, is stable in a specific language and similar across languages and mainly reduces substantially the size of text tensor representations.

For being based in character composition of words, with this encoding we could take advantage of morphological constructions of words and do not need trained embeddings. Furthermore, by representing a complete word, we could easily use all the techniques and architectures already developed for word embeddings and even concatenate word embeddings if available, inserting some semantic information into the representation. It could deal with typos, equations, formulas and words not present initially in our training dataset.

For the potential to help a wider audience of our community, we decided to concentrate efforts in studying its properties and comparing it with state-of-art approaches of the literature in the time due to this dissertation and postpone the initial objective to a later moment, being more sure of its viability. This dissertation is dedicated to report our findings regarding *Word101*.

Recently, Su et al. (2018) reported a similar attempt to our original idea of using the text of questions to enhance the performance of strategies using DKT with very encouraging results. This reinforces the belief that we could be in the right track in our intents.

1.2 Dissertation Outline

This work is structured as follows: In Chapter 2 we review the literature of text representations and contrast the choice of the token as characters, words and subwords representations. In the same chapter we review algorithms applied to the task classification texts, both linear as Deep Learning approaches. In Chapter 3 we describe our proposal of text representation and list some of its properties. In the same chapter, we detail the Neural Network architecture chosen to couple with this representation and earlier attempts done until we arrive at that particular architecture. Chapter 4 shows the results of an experimental study of our proposal in massive texts datasets and compare it against traditional methods and state-of-art methods, both in character as in word token parsing representation. Finally, in Chapter 5 we present our conclusions and suggested future research directions. In Appendix A there are listed two publications done in this period and we reproduce a paper done with details of our earlier attempt.

Chapter 2

Foundations

2.1 Preliminaries

Classify a document is a very basic task in *Natural Language Processing* (NLP) (MANNING; SCHÜTZE, 1999; JURAFSKY; MARTIN, 2014). The main idea is to use the text contents (words, suffix, part-of-speech, letters, symbols, equations, diagrams, etc.) and its compositional structure in an algorithm to make inferences about the implicit or explicit properties of these texts (NORVIG, 1987) as domain recognition, authorship, language, geographic location, etc. To model this problem, the main challenges could be split in two components:

1. **How to represent text contents and what the implications of this choice in the process?**

   To use the power of a computer and its algorithms, we have to represent text contents in some numerical format. The representation choice directly influences if we could represent the totality of the information, the speed of execution and the accuracy of our classification.

2. **Which computational approaches should we use to classify documents using these representations?**

   Written language is a tool that humans use to represent ideas that by its own nature, it has a very dimensional richness. Not only the vocabulary of tokens that represent these ideas is extensive, the particular combination of them could alter completely the meaning of the text. The power of the algorithm to model this variability and make inferences about it directly influence its accuracy.

In the following sections, we list a non-exhaustive list of the most usual answers to these questions available in the literature to better contextualize our proposal.

2.2 Text Representations

Usually, to represent text two choices are available:

The first choice is defining what is least piece of that structure, the token. For the western languages, the usual choice of the atomic constituent of a text is a word or a character, but we could use ideograms as in Eastern Languages like Japanese, Chinese or even be hieroglyphs, like in ancient Egypt. There are also choices of N-grams, which are combinations of n neighbor characters or words. In theory, we could represent a complete phrase as a token that could define a text. The problem with this approach is that the variability of expressions possible is enormous. Even if we could deal with a vector that attributes a single dimension to a particular expression, we will need a massive amount of data with that expression in different contexts to have some statistical pattern validation.

The second choice is how to represent each token of that structure as a numerical value, or in a more formal way, how to transform a categorical variable into a distinct vector. In the literature, the common choice is a very sparse one-hot encoding or a continuous dense representation using some embedding strategy.

2.2.1 Word One-hot encoding

In the case of word one-hot encoding, for representing each word in a vocabulary of size N, a digit 1 is placed in the correspondent position of that word in a $1 \times N$ vector, all others positions remaining with digit 0. The problem with this approach is that the number of words in a language is unknown, but surely is very big. The Second Edition of the 20-volume Oxford English Dictionary (OXFORD..., 2012) contains full entries for 228,132 words but these numbers exclude inflections, and words from technical and regional vocabulary, slang and neologisms. Depending on how we define what is considered a word, a language could easily has more than billions of words. Glove (PENNINGTON; SOCHER; MANNING, 2014), a repository of embeddings of words using Wikipedia (Wikipedia contributors, 2004) list at least 6 billions of different words.

The dimensionality of the vectors has a direct effect on memory requirements and processing time (GOLDBERG, 2015) in many algorithms. To enable processing, usually the vocabulary is restricted to some small set with thousands of entries, with only the main significant words of a domain. It is common to use the help of a specialist or some form of statistical frequency to judge which words should be on the vocabulary. This step is very important because we could only represent words that are in our restricted vocabulary set, all others being ignored.

One very common way to reduce the dimensionality of the word vector representation is discarding the most and the least frequent words. The idea is that very frequent words are usually stop words or not so important words. Words that have very low frequency should have very little power in classifying a document.

One problem with using word one-hot encoding is that it only can represent words that are contained in its vocabulary, or in other terms, we are a hostage of our training dataset. If new words appear in our dataset of interest, being them a neologism or a combination of known words, we have not a representation for them. Another problem is that each word is independent from one another (GOLDBERG, 2015). The word *'dog'* is as dissimilar to word *'cat'* as to word *'thinking'* (GOLDBERG, 2015)

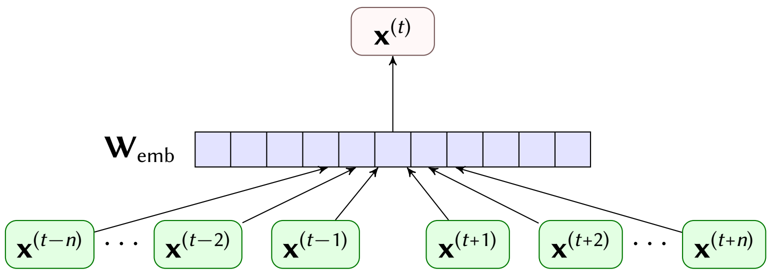
2.2.2 Word Embedding Representations

The strategy of representing words as a dense vector was introduced by Bengio et al. (2003) and popularized by *Word2vec* strategy of Mikolov et al. (2013).

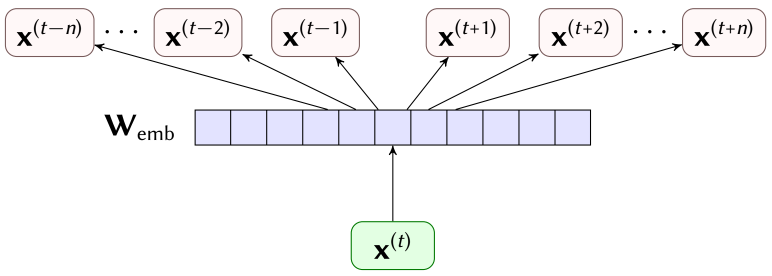
The main problem of using word one-hot representation to encode words is the extreme amount of words in a language. Even if we have enough memory to store vectors with the same dimension of available words, some words will appear few times in our training dataset, becoming difficult to establish a significant statistical pattern recognition inference about the presence of that word in a text. This problem is recognized as the curse of dimensionality (BENGIO et al., 2003).

To reduce this dimensionality without discarding some word occurrence in a document some pre-training could be used to convert that big word vector to a smaller and more computational efficient embedding. That strategy has two flavors: *Skip-Gram* and *Continuous Bag of Words* (CBOW).

The idea behind *skip-gram* is simple: train a single layer of neurons in a neural network to predict near words given a single word in a massive amount of text documents, then use the weights of that training as a word representation. Usually, the strategy is performed using a window of 5 words. The input is a central word and the network tries to predict the 2 near words before and the 2 near words after. It is very common use 100, 200 or 300 weights in a fully connected neural network, that will be used as the vector representation of that word.


$\mathbf{x}^{(t)}$
$\mathbf{W}_{\text{emb}}$
$\mathbf{x}^{(t-n)}$ ... $\mathbf{x}^{(t-2)}$ $\mathbf{x}^{(t-1)}$ $\mathbf{x}^{(t+1)}$ $\mathbf{x}^{(t+2)}$ ... $\mathbf{x}^{(t+n)}$

(a) Continuous bag of words (CBOW).


$\mathbf{x}^{(t-n)}$ ... $\mathbf{x}^{(t-2)}$ $\mathbf{x}^{(t-1)}$ $\mathbf{x}^{(t+1)}$ $\mathbf{x}^{(t+2)}$ ... $\mathbf{x}^{(t+n)}$
$\mathbf{W}_{\text{emb}}$
$\mathbf{x}^{(t)}$

(b) Skip-gram.

Figure 2.1: Schematic representation of the CBOW and Skip-Gram word embeddings as explained in Mikolov et al. (2013).

The intuition behind this strategy is that words with similar meaning will be surrounded by similar words, so words that appear in the same context should have similar vector representations.

Another flavor of that strategy is doing the opposite, input in a network the context words and try to predict the centered word. This strategy is named *Continuous Bag of Words* (CBOW).

In fact, the strategy is using sparse, one-hot vectors as input and them dedicating the first layer of a network architecture to learning a dense embedding vector for each word based on the training data (GOLDBERG, 2015). As the input is a one-hot encoding representation of words, the same problem of dealing with words not present in the training dataset is inherited. So neologisms, slang or words with some misspellings will not have a vector representation.

The great advantage of this strategy is that word representation are not equally dissimilar. In the dense vectors representation, the learned vector for 'dog' may be similar to the learned vector from 'cat', since it appears in similar contexts, allowing the model to share statistical strength between the two words (GOLDBERG, 2015). So, the main benefit of the dense representations are in generalization power, representing similar ideas as similar vectors, independently of the word used to represent that idea, like in the case of *'cat'* and *'dog'* that are related to the context of domestic pets.

Skip-Gram and CBOW strategies could be applied to the documents that we intend to classify, but have a better general performance if trained in a massive amount of texts. *Global*

Vectors (Glove) (PENNINGTON; SOCHER; MANNING, 2014) is repository maintained by *Stanford University* of words vector representations trained on *Wikipedia* (6 billion words) or *Twitter* (27 billion words). This alleviates the problem of not having a word representation for words not present in the training dataset if that word is present in this repository.

On another hand, since the majority of written content of the internet is in English, the meaning of words is very biased to that language.

There is a variant of this strategy called *Fasttext* (JOULIN et al., 2016) developed by *Facebook Inc.* that uses all n-grams of a word as inputs to the vector embedding strategy. The main advantage is that it could deal better with rare words, or smaller datasets because it can use part of the word that is common to other words to infer some similarity. *Facebook* released pre-trained word-vectors in more than 150 languages using texts available in *Wikipedia* (JOULIN et al., 2016).

The repository of vector representations is a good general representation of a word in general contexts. If your data is related to a specific context, like a technical area, some words could have a better representation if were trained only on related documents. For example, the word *"Set"* could be related to *"adjustments"*, but could be defined as a collection of well defined and distinct objects in *Mathematics*, a sequence of games played with alternating service and return roles in *Tennis* sport or even a deity on ancient Egyptian texts.

As a starting point, it could be a good idea using pre-trained data and them fine-tune the representation with new data, especially if there are not so many documents available at your selected training dataset.

2.2.3 Character One-Hot Representation

Representing words as vectors using word *one-hot encoding* have the problem of big dimensionality. Using embeddings of words pre-trained in a massive amount of texts alleviate this problems, but the representation will be a hostage of the generality of that massive training set that could or not produce a vector representation that favor your dataset of texts. Equations, neologisms, slang, emoticons and other common aspects of internet texts could or not be represented in pre-trained datasets.

These techniques work well enough when applied to a narrowly defined domain, but the prior knowledge required is not cheap – they need to predefine a dictionary of words of interest and the structural parser needs to handle many special variations such as word morphological changes and ambiguous chunking. (ZHANG; ZHAO; LECUN, 2015)

These requirements make text understanding more or less specialized to a particular language and if the language is changed, many things must be engineered from scratch (ZHANG; LECUN, 2015). If your dataset is not in English the problem may become more important, since the majority of words in these training texts databases are biased to that language. In Chinese, tokens are not separated by spaces or other typesetting conventions. For most NLP applications, German compounds should be split. Tokens in agglutinative languages like Turkish are difficult to process as unanalyzed symbols (BLUNSOM et al., 2017).

The success of deep learning in speech, vision and machine translation demonstrates the potential of giving models direct access to the data as opposed to through the intermediary of human-designed features. This last model the data as it comes in without any alteration through manually designed features. Especially with *Deep Learning* models, it is possible that the models learn their own representation of the data without the need of tokenizing the character sequence into words (BLUNSOM et al., 2017).

If we give up the notion that a token is an opaque symbol and instead model the sequence of characters it is made up of, then we can in principle learn all morphological regularities: inflectional and derivational regularities as well as a wide typological range of morphological processes such as vowel harmony, agglutination, reduplication, and nonconcatenativity (BLUNSOM et al., 2017)

Addressing these issues, Zhang, Zhao & LeCun (2015) suggested considering the characters as the atomic elements of a text. The problem of the dimensionality of the vector is alleviated, because usually there are not so many of distinct characters in an alphabet of a language. Even in languages that use ideograms, the number of them is much lower than the amount of possible words.

This simplification of engineering could be crucial for a single system that can work for different languages, since characters always constitute a necessary construct regardless of whether segmentation into words is possible (ZHANG; ZHAO; LECUN, 2015).

Zhang, Zhao & LeCun (2015) used only 69 characters as an alphabet to represent a document in datasets of texts in English and romanized Chinese. Using a *one-hot* character encoding with a dictionary of characters, an alphabet, at each line of the matrix representing the document, the corresponding character position of the vector is considered 1 and all others positions 0. The non-space characters are letters, numbers and punctuation.To allow processing this approach, Zhang, Zhao & LeCun (2015) limited the number of characters in a document to 1014. So, each document is represented in a matrix of dimension 1014×69

Table 2.1: Architectures of the 'large' and 'small' CNNs used by Zhang, Zhao & LeCun (2015).

CONVOLUTIONS

Layer	Large Feature	Small Feature	Kernel	Poll
1	1024	256	7	3
2	1024	256	7	3
3	1024	256	3	—
4	1024	256	3	—
5	1024	256	3	—
6	1024	256	3	3

FULLY CONNECTED

Layer	Large Feature Out	Small Feature Out	Dropout
7	2048	1014	0.5
8	2048	1014	0.5
9	Depends on the problem		

The big challenge was to show that such an approach could be learned by an algorithm, since the great amount of variability imposed by trying to understand texts from scratch. To classify texts using this strategy, they applied a deep *Convolutional Neural Network* (CNN).

The model is composed of 9 layers, 6 of convolutions and 3 fully connected. Their architecture is described in Table 2.1.

They used *Stochastic Gradient Descent* (SGD) with a mini batch of size 128, using a momentum of 0.9 and initial step size 0.01 which is halved every 3 epochs for 10 times. So, their results were obtained in at least 30 epochs.

This encoding provides a robust, language-independent representation of texts as matrices, that are then used as inputs of different CNNs. Their experimental results showed that this approach was able to attain and, in some cases, improve the state of the art results in complex text classification problems.

The drawback of this strategy is that the training is more complex since the network has to learn to identify some constructions belonging to certain classes of texts, requiring more epochs to converge and have to deal with matrices of bigger dimensions, since a document have many more characters than words. This entire process is slower than using word representations.

2.2.4 Sub-word Representations

Deriving representations of words from the representations of their characters is motivated by the out-of-vocabulary words problem. Working on the level of characters alleviates this problem to a large extent, as the vocabulary of possible characters is much smaller than the vocabulary of possible words. However, working on the character level is very challenging, as the relationship between form (characters) and function (syntax, semantics) in language is quite loose(GOLDBERG, 2015).

Restricting oneself to stay on the character level may be an unnecessarily hard constraint. Some researchers propose a middle-ground, in which a word is represented as a combination of a vector for the word itself with vectors of sub-word units that comprise it(GOLDBERG, 2015).

These approaches also have the benefit of producing very small model sizes (only one vector for each character in the alphabet together with a handful of small matrices needs to be stored), and being able to provide an embedding vector for every word that may be encountered. (GOLDBERG, 2015; SANTOS; ZADROZNY, 2014) model the embedding of a word using a convolutional network over the characters and found state of art results in part-of-speech (POS) tagging.

One of the great advantages of this strategy is that you could use word embeddings available from *Glove*(PENNINGTON; SOCHER; MANNING, 2014) or *Fasttext* (JOULIN et al., 2016) to known words and use the sub-word representation to guess the meaning of similar words. Dismissing any handcrafted feature decision is another advantage.

On another hand, there is a need to use training stage to achieve the character-word representation, and although it can represent out of vocabulary words, this training representation is dependent on the dataset used. Different training datasets achieve different representations.

2.3 Algorithms

2.3.1 Traditional Linear Algorithms

A traditional strategy to classify text documents using *one-hot* word representation is Bag of Words (SALTON; MCGILL, 1983). In this strategy, a vector composed of a sum of all word-vectors present in the text represents a document. This approach only takes into account the presence or not of a word, so its position in the text is ignored, the analogy is that each document is considered a *bag* of mixed words.

To allow statistical relevance and be able to store word vector representation on memory, the number of words should be restricted to thousands of words. The selection of words in a vocabulary is determinant on the accuracy of the algorithm. Usually is applied an specialist in this stage or applied some kind of statistical filter, where most frequent words (stop words) and rare words are discarded.

Another variation of this strategy is weighting the importance of words in a document by its *Term Frequency-Inverse Document Frequency* (TF-IDF) (JONES, 1972). In this technique, the *Term Frequency* is the number of times a word is present in a certain document and the *Inverse Document Frequency* is the frequency of that word in all documents. The TF-IDF of a word in a document is the *Term Frequency* divided by the *Inverse Document Frequency*. The idea of the technique is that words that are too much frequent will appear in many documents, so its importance is lower than a certain type of word that usually only appear in a specific class of documents.

To compare or classify each document, usually a *Multinomial Logistic Regression* or a *Naive Bayes algorithm* is applied.

In the *Naive Bayes algorithm* approach, is assumed that words are independent from one another and the conditional probability of a document to belonging to a particular class could be calculated using the frequency of words on the document and frequency of the same words in a particular class using the *Bayes Theorem*. (See (JURAFSKY; MARTIN, 2000))

In the *Multinomial Logistic Regression* (also called *MaxEnt* or the *Maximum Entropy classifier*) approach, the frequency of occurrence of a set of words is compared with the frequency of the class using an exponential normalization and using linear combinations that maximize entropy. (See (JURAFSKY; MARTIN, 2000))

The great advantage of these linear models is that they are very fast do compute. The number of classes should be small, ideally binary. In *Bag of Words* strategies, the position of a word does not matter, so it works well when some set of words are characteristic of a kind of document. It has very success in tasks like spam filters (SAHAMI et al., 1998), where few words are very common in many junk messages.

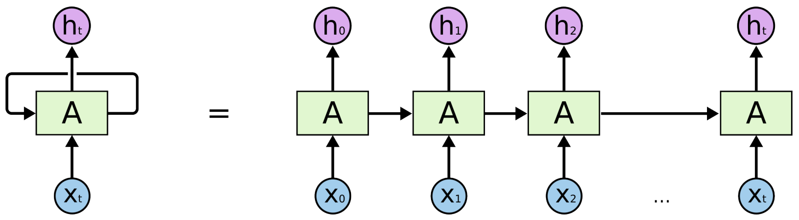


Figure 2.2: Schematic representation of an unfolded Recurrent Neural Network. Taken from (OLAH, 2015).

2.3.2 Recurrent Neural Networks

In language, the sequence of words in a sentence and the sequence of sentences in a document are important in the structure of the ideas that a text is delivering.

One way to model the influences of past events in a decision using neural networks is a *Recurrent Neural Network*. In this architecture, an internal hidden state takes into account some information of what was already presented to the model and use such an event to take decisions. We can think that this hidden state act like a primitive memory because it have not access to what happened before, but everything that already happened modifies the behavior on the next decision.

The architecture where the recurrence is represented by a set of hidden states is called *Vanilla RNN*. A schematic representation is presented in fig. 2.2. It is very efficient to take into account recent events. The problem of using this kind of architecture is that sometimes, the important event happened in a long distance of the decision. In the meantime, a bunch of others inputs modified the model, and that critical information has its power diluted.

One solution is using more hidden states to represent more information from the past, but this usually makes the gradient vanish and unable the training (BENGIO; SIMARD; FRASCONI, 1994). Another solution is to develop architectures that are more sophisticated and less prone to vanishing. Some of them use as a strategy deciding which information taking into account and what are the ones that should or not interfere in the next decision. The most popular are *Long Short Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997) and *Gated Recurrent Unit* (GRU) (CHO et al., 2014) networks.

The *Long Short-Term Memory* (LSTM) (HOCHREITER; SCHMIDHUBER, 1997) architecture was designed to solve the vanishing gradients problem. The main idea behind the LSTM is to introduce as part of the state representation also “memory cells” (a vector) that can preserve gradients across time. Access to the memory cells is controlled by gating components – smooth mathematical functions that simulate logical gates. At each input state, a gate is used to

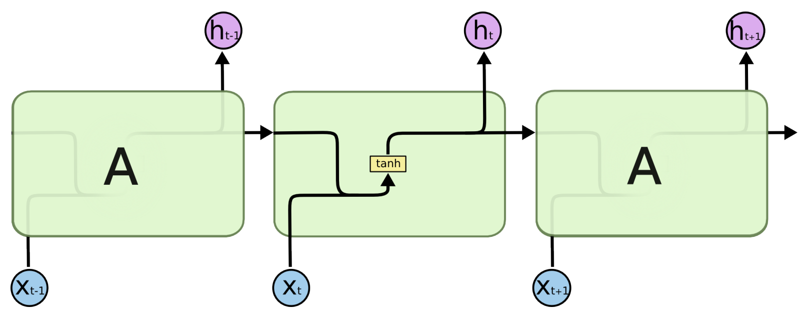


Figure 2.3: Schematic representation of an unfolded Vanilla Recurrent Neural Network. Taken from (OLAH, 2015).

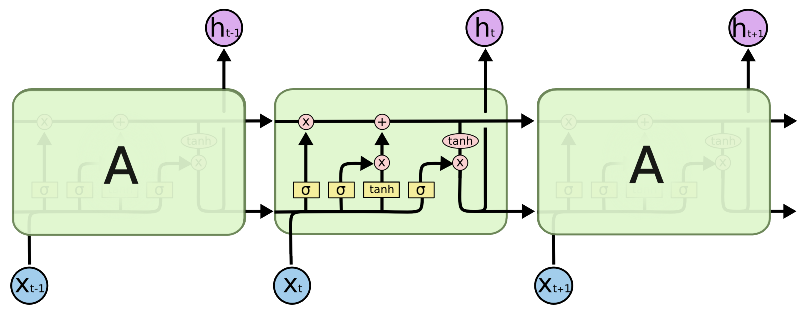


Figure 2.4: Schematic representation of an unfolded LSTM. Taken from (OLAH, 2015).

decide how much of the new input should be written to the memory cell, and how much of the current content of the memory cell should be forgotten (GOLDBERG, 2015).

Unlike the traditional recurrent unit which overwrites its content at each time-step, an LSTM unit is able to decide whether to keep the existing memory via the introduced gates. Intuitively, if the LSTM unit detects an important feature from an input sequence at an early stage, it easily carries this information (the existence of the feature) over a long distance, hence, capturing potential long-distance dependencies (CHUNG et al., 2014). In Figure 2.3 there is an illustration of a simple RNN in contrast with LSTM cells in Figure 2.4. We can see the gates indicated by $\otimes$. The first one decides if the cell will or not will forget something, the second one decides if the cell will store that information and the last one decide what the cell will output (OLAH, 2015).

LSTMs are very powerful to model sequences but have a downside that there are too many parameters to optimize, so it is slower to train. Some variations were proposed. *Gated Recurrent Unit* proposed by Cho et al. (2014) is one of this variations.

Similarly to the LSTM unit, the GRU has gating units that modulate the flow of information inside the unit, however, without having a separate memory cell. One feature of the LSTM unit that is missing from the GRU is the controlled exposure of the memory content. In the LSTM unit, the amount of the memory content that is seen, or used by other units in the network is

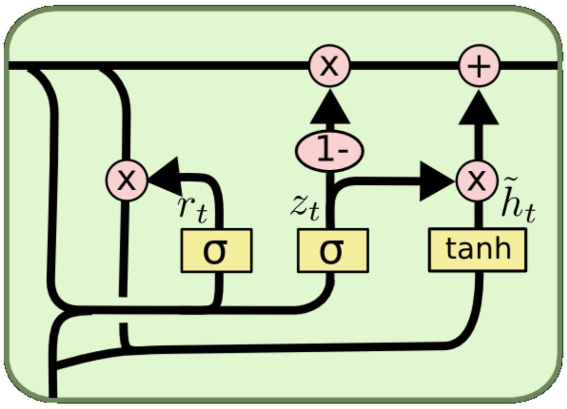


Figure 2.5: Schematic representation of a GRU cell. Taken from (OLAH, 2015).

controlled by the output gate. On the other hand the GRU exposes its full content without any control (CHUNG et al., 2014)

Another difference is in the location of the input gate, or the corresponding reset gate. The LSTM unit computes the new memory content without any separate control of the amount of information flowing from the previous time step. Rather, the LSTM unit controls the amount of the new memory content being added to the memory cell independently from the forget gate. On another hand, the GRU controls the information flow from the previous activation when computing the new, candidate activation, but does not independently control the amount of the candidate activation being added (the control is tied via the update gate) (CHUNG et al., 2014).

The gated architectures of the LSTM and the GRU help in alleviating the vanishing gradients problem of the *"vanilla" RNN*, and allow these RNNs to capture dependencies that spam long time ranges. Jozefowicz, Zaremba & Sutskever (2015) did an extensive comparison among LSTM, GRU and some mutations of them in various tasks (language modelling, number sequencing, music next note prediction) and found that is not clear which one is the best. GRU outperformed LSTM on various tasks with exception of language modeling. On another hand, LSTM with a large forget bias outperformed both LSTM and GRU an almost all tasks (JOZEFOWICZ; ZAREMBA; SUTSKEVER, 2015).

2.3.3 Convolutional Neural Networks

Convolution Neural Networks architectures (LECUN; BENGIO, 1998) evolved in the neural networks vision community, where they showed great success as object detectors regardless of its position in the image (KRIZHEVSKY; SUTSKEVER; HINTON, 2012).

When applied to text, NLP we are mainly concerned with 1-d (sequence) convolutions. *Convolutional Neural Networks* were introduced to the NLP community in the pioneering work of Collobert et al. (2011) who used them for semantic-role labeling, and Kim (2014) who used them for sentiment and question-type classification.

The main idea behind a convolution and pooling architecture for language tasks is to apply a non-linear (learned) function over each representation of a k-word sliding window over the sentence. This function (also called "filter") transforms a window of k words into a d dimensional vector that captures important properties of the words in the window (each dimension is sometimes referred to in the literature as a "channel"). Then, a "pooling" operation is used to combine the vectors resulting from the different windows into a single d-dimensional vector, by taking the maximum or the average value observed in each of the d channels over the different windows (GOLDBERG, 2015).

The intention is to focus on the most important "features" in the sentence, regardless of their location. The d-dimensional vector is then fed further into a network that is used for prediction. The gradients that are propagated back from the network's loss during the training process are used to tune the parameters of the filter function to highlight the aspects of the data that are important for the task the network is trained for (GOLDBERG, 2015).

In layman terms, we can think that a convolutional compare the region of its filters with the sequence of words that we presented one by line. When it finds a specific combination of words in the region where the filter is looking, it activates a signal and the pooling layer pass that information to the next layer. The next layer has combinations of combinations as input, and if a specific group of them is found, it activates a signal and the pooling layer pass that information to the next layer. The process is repeated for all the layers in the architecture of the network. In the final layer, a decider take all signals coming from various available combinations into account and make a probabilistic inference judging if that information is relevant to better recognize a class. The decider does not know where the combination of combinations was found, just that it was found.

The accuracy of the strategy greatly depends on how a sequence of layers is structured. As a rule of thumb, the deeper is the architecture, the more powerful it is, because the decider have information of many possible combinations and could make a better judgment. The problem is that the deeper is an architecture, more difficult is propagates its errors to the first layers and adjust the filters weights. Another problem is that too deep architectures could memorize our training dataset if it does not have enough variability, being a poor predictor of new data. So, to train deep architectures we need a massive amount of samples in various

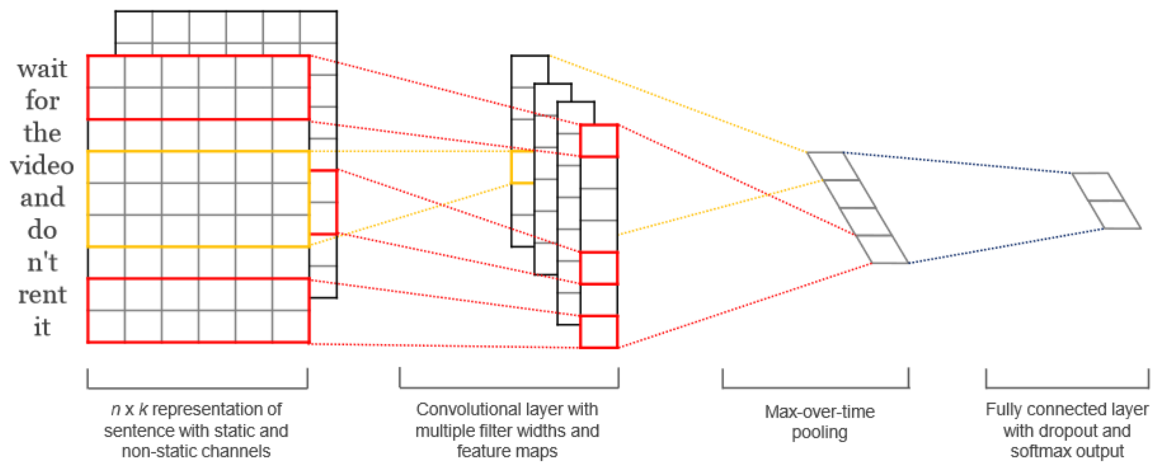

wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 2.6: Schematic representation of a wide convolutional network. Taken from (MIKOLOV et al., 2013).

contexts of interest.

Until now, there is not a theory on how to design good architectures, but some of them became famous by solving hard problems and for being robust in many problems. Many of them by achieving success in *ImageNet* (DENG et al., 2009) contest. *AlexNet* (KRIZHEVSKY; SUTSKEVER; HINTON, 2012), *VGG* (SIMONYAN; ZISSERMAN, 2014), *Inception* (SZEGEDY et al., 2015), *ResNet* (HE et al., 2015) are examples of popular very deep architectures.

Another option is instead of making a deeper architecture, make a wider one. Several convolutional layers may be applied in parallel each one with a different window size in the range 2–5, capturing n-gram sequences of varying lengths. The result of each convolutional layer will then be pooled, and the resulting vectors concatenated and fed to further processing (GOLDBERG, 2015). Kim (2014) applied a convolution of this type to classify text with very good results.

A great advantage of *Convolutional Neural Networks* over *Recurrent Neural Networks* is that it allows computing many operations in parallel, being generally faster to train. Another advantage is the possibility of very deep architectures, where initial layers look at local combinations and last ones being able to construct higher-level abstractions of these earlier combinations.

An indirect advantage of using *Convolutional Neural Networks* for NLP is that since the success of Deng et al. (2009) in identifying objects in images, practitioners of deep learning for images and video use them a lot to generate their artificial intelligence models. Many of architectures generated by them could be reused or re-adapted to text models. Not to mention the fact that their market of applications is bigger and problems are much more computationally intense, so there is a lot of interest in building efficient solutions, frameworks, libraries and hardware able to compute it.

Chapter 3

Proposal

3.1 Compressed Encoding for Text Representation

So far, character-level approaches fall short when compared with the state of the art word-level approaches (BLUNSOM et al., 2017) in cases where token structures can easily be exploited (e.g., in well-edited newspaper text). However, in modern Internet sources and social networks, there is a diversity of producers of texts. Reviewers on websites, social networks users, blogs, twitter, etc. produces a myriad of not curated texts constructions that need the power of a robust method.

Human reading is robust in the sense that small perturbations on the input do not affect processing negatively. Such perturbations include letter insertions, deletions, substitutions and transpositions and the insertion of spaces and the deletion of spaces. Such perturbations can cause complete failure of token-based processing (BLUNSOM et al., 2017).

Orthographic blends and modifications of existing words (e.g. "staycation", "Obamacare"), character repetition in tweets (e. g. "cooooooooooool"), the pseudo-derivational suffix "-gate" signifying "scandal" (e. g. "Watergate, "Irangate, "Dieselgate"), number variation ("4.12 million" may become "4 million") are examples where word tokenization is useless.

It can be asserted that probably at some level there exists a morpheme and then an inflection generation on the morpheme that leads to the creation of new words (BLUNSOM et al., 2017). A common agreement is that derivational morphology is often ignored in modeling, even though around 50% of words in English are constructed through derivational processes.

Natural Language Processing must sometimes consider the internal structure of words in order to understand or generate an unfamiliar word. Unfamiliar words are systematically related to familiar ones due to linguistic processes such as morphology, phonology, abbreviation, copying error, and historical change (BLUNSOM et al., 2017).

A solution to these questions is character-level models, but they are computationally more expensive than word-level models because detecting syntactic and semantic relationships at the character-level is more complicated (even though it is potentially more robust) than at the word-level (BLUNSOM et al., 2017). This problem could be alleviated if we delivery to the model text already parsed into words because the model does not have to analyze the intersection of last letters of an anterior word and the beginning of the next. Furthermore, if we could represent a word by its characters in a single vector, we could reduce the size of the input matrix, because there are much fewer words in a text than characters, a possible key to have a better performance in training.

By searching for a way to retaining the flexibility and power of character-level convolutional to classify text while reducing its training time, we found a way to better encode texts. Our approach, with competitive accuracy, achieve a significant reduction in time execution from hours to minutes and from days to hours, by epoch compared to Zhang & LeCun (2015) and Conneau et al. (2016).

Our approach is based on the *Tagged Huffman encoding* (MOURA et al., 2000; BRISABOA et al., 2003), where a certain number of pairs of '0' digits is the signal and the digit '1' is the tags the beginning and the end of code, the only difference is that for our approach, we need a shorter version to reduce the size of the input matrix, so we choose to use only one digit 0 instead of two for each character, marking the beginning and the end of each char code with a digit 1, the same way.

As in the approach by Moura et al. (2000), the coding we employ has the following advantages:

1. No character code is a prefix of another, that is, the match is one-to-one.
2. The coding procedure is not restricted to any size of vocabulary.
3. The representation is stable in a given language, not depending on which set of documents we are dealing.
4. We could concatenate codes for representing each word, so the ones with the same prefix were represented by vectors near each other, responding especially well to morphological derivations.
5. It allows a direct search, that is, to search for a word in an encoded document, just

encode the word and use traditional methods of comparing strings with the encoded document.

6. This encoding approach is a binary compression technique, requiring less storage space, less use of RAM and smaller data transfer latencies.
7. There is no need to decompression to perform a classification.
8. There is no need to train a representation, so differences of performance are directly related to the architecture used, facilitating comparison among them.
9. This approach could be coupled with any other word embedding method, if available.

All these properties allow saving already encoded text documents permanently, thus becoming a useful solution especially if the goal is to extract knowledge about files in a repository to perform various classifications on different dimensions with the same files in distinct moments.

Another advantage is that it is able to respond to unseen words on training data since the network will at least have some prefix to guess the meaning of the word, the same way we humans do. This is especially interesting in languages where there are a lot of declensions like Portuguese, Spanish, Italian, French, Russian, German, Arabic and Turkish, for example.

Our main contribution is to demonstrate that such an approach reduces the dimensionality of the encoded matrix and that consequently substantially reduces training time and allows the use of devices with lower computational power, with no harm to accuracy.

3.2 Word101-Encoding Text Representation Procedure

In our approach, we used the following procedure to encode text:

- *Obtaining a character frequency rank*: We read the text database and count the frequency of each character, generating a list sorted by frequency of occurrence. Then we create a rank with only the relative positions of the characters. For a given language, this rank is quite stable since only the order of the rank is used. This means that if all documents are in the same idiom, this procedure can be replaced by a list with the characters rank of frequency for that language.

Table 3.1: Example of coding using English language ranking of characters. Characters shown are the ones used in the subsequent examples.

CHARACTER	FREQ. RANK	COMPRESSED ENCODING
␣	0	11
e	1	101
a	2	1001
t	3	10001
i	4	100001
s	5	1000001
n	7	100000001
r	8	1000000001
d	10	100000000001
h	11	1000000000001
c	12	10000000000001
g	17	1000000000000000001
⋮	⋮	⋮
	n	$'1' + n \times '0' + '1'$

Table 3.2: Example of coding using English language ranking of characters. Prefix common to more than a word is underscored

TEXT	ENCODED TEXT
science	100000110000000000000110000110110000000110000000000001101
scientist	100000110000000000000110000110110000000110001100001100000110001
art	1001100000000110001
artist	1001100000000110001100001100000110001
tl;dr	10001100000000011000000000000000000000000000001110000000000110000000001
u2	100000000000000110000000000000000000000000000000001
e^a	1011000000000000000000000000000000000000000000000000000000000011001

- *Creating a mapping from character to compress code*: To encode each character, we insert the digit 0 in the same amount of the rank position of the character, between a 1 digit to signal the begin and another 1 digit to signal the end of the code.

Intuitively, the larger is the frequency, the smaller is the code, as we could see in Table 3.1. Based on this strategy, we named this encoding Word101. Table 3.1 has some examples of encoded characters. To encode each word, we just concatenate the codes of each character. As previously commented, Figure 3.1 provides an example of the result of encoding a phrase.

To encode each word, we just concatenate the codes of each character. As an example, we provide in Table 3.2 some examples of plain text words and their corresponding encoding.

Given a document, we consider that it is composed of words, being those any set of

```
10000000011011000001101100110000000011000000000000110000000000010
10000110000010000000000000000000000000000000000000000000000000000
10011000000010000000000000000000000000000000000000000000000000000
10011000000001100010000000000000000000000000000000000000000000000
10011000000011000000000010000000000000000000000000000000000000000
10010000000000000000000000000000000000000000000000000000000000000
10000011000000000000110000110110000000110000000000001101000000000
00000000000000000000000000000000000000000000000000000000000000000
```

Figure 3.1: Matrix encoding of sentence ‘Research is an art and a science’. The text is encoded one word per row. Each word is underscored for easy identification.

characters limited by the space character. This means ‘word’ could be math equations, web addresses, LaTeX code, computer programming commands, etc. In Table 3.2 we can see that this encoding could represent words with the same prefix in vectors that have same initial coordinates. Slangs like *tl;dr : too long, did not read* and *u2 : you too* as well mathematical expressions like e^a could be manipulated.

In a matrix of *number of words* $\times$ *code size* representing the document, each row represents a properly encoded word, where the code is embedded with its first symbol in the first column. Columns that were not occupied are padded with 0, larger codes are represented up to the chosen limit. Unoccupied lines are filled with 0 and larger documents are represented only up to a maximum number of words, ignoring the last remaining ones. As an example, we represent a document in an 8×65 matrix in Figure 3.1.

In this example, we used a 8×65 matrix (520 elements) to encode the text with a certain amount of slack. At the very least, we would need 7×64 (448 elements). In contrast, the approach of Zhang, Zhao & LeCun (2015) would employ at least 32×69 (2208) elements to represent the same sentence.

In the datasets studied in this work, 256 rows were enough to represent 99.5% of the words. We choose 128 as a limit of words to represent a document, encoding each text in a 128×256 matrix.

We can see that *Word101* represents a character present in a vocabulary sorted by frequency with a correspondent number of zeros, and tags the begin and end of signal with a digit 1. It is also possible to generate a similar encode *Word01*, where only the end of signal is tagged. That last strategy is similar to concatenate one-hot encoding, discarding information after the correspondent one is found. The main problem with this encoding is that some codes are a subset of others. Maybe that is the reason why some experiments that we did with this encoding have similar results, but *Word101* have a better accuracy.

3.3 Deep Pyramid Convolutional Neural Networks

To show the possibilities of this encoding to text classification, we choose a deep and computational efficient network architecture: *Deep Pyramid Convolutional Neural Networks* (JOHNSON; ZHANG, 2017).

He et al. (2015) developed an architecture named *Deep Residual Networks-RESNET*, where a series of blocks composed of convolutions and a parallel shortcut allows to produce a very deep architecture, with a depth of up to 152 layers. Using that architecture they achieved state of the art results in *IMAGENET* - image classification contest (DENG et al., 2009). In the domain of text classification, Conneau et al. (2016) applied a similar network architecture using a *one-hot encoding* suggested by Zhang & LeCun (2015) to represent text at character level, achieving very good results.

Recently, Johnson & Zhang (2017) created a new interpretation of RESNET architecture named *Deep Pyramid Convolutional Neural Networks* that they applied in text classification with a region embedding using one-hot encoding in the word level and using the 30K more frequent words to compound the text matrix representation. They achieved state of the art results in datasets cited in Zhang & LeCun (2015). More importantly, the architecture created by Johnson & Zhang (2017) has very interesting features, many of them derived by using a pooling of 3 with a stride of 2 at the beginning of the recursive block:

- Every block reduces on half the number of parameters to be trained, forming a geometric progression with a ratio 0.5, so the number of parameters is limited to the double of the first iteration(JOHNSON; ZHANG, 2017), allowing to be deep but with fast training times.
- A pool of 3 with stride 2 allow each layer double the effective coverage of each of the convolution kernel (JOHNSON; ZHANG, 2017). In our implementation, each instance of convolution kernel on the last layer has access to all information on the document.

The architecture used in this dissertation is composed of a sequence of pyramid units that progressively reduce the dimension of the data, as shown in Figure 3.2. The network architecture is presented in Figure 3.3. It is similar to that one introduced by Johnson & Zhang (2017), with minors differences in selected parameters.

The main difference is that instead of embedding each word using some kind of co-occurrence statistics, we applied word101 to encode each word and form the matrix representation of texts.

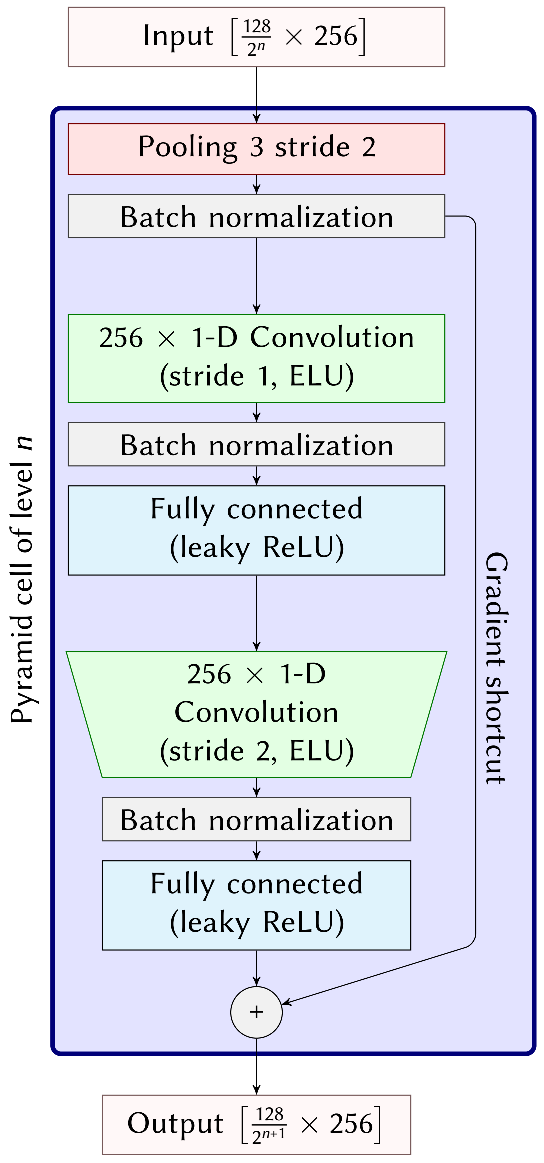

Input $\left[\frac{128}{2^n} \times 256\right]$
Pooling 3 stride 2
Batch normalization
256 × 1-D Convolution (stride 1, ELU)
Batch normalization
Fully connected (leaky ReLU)
256 × 1-D Convolution (stride 2, ELU)
Batch normalization
Fully connected (leaky ReLU)
+
Pyramid cell of level n
Gradient shortcut
Output $\left[\frac{128}{2^{n+1}} \times 256\right]$

Figure 3.2: A cell of level *n*.

As in Zhang, Zhao & LeCun (2015), *Word101* uses the character as inputs, but that information is used to compose words. Indirectly, we achieved a way to take the morphological structure of word formation into account, with all advantages that the approach of Zhang, Zhao & LeCun (2015) suggests, but with faster training times.

To make a comparison, performing the training of the architecture suggested with an output of 4 classes make necessary to optimize 1,135,876 parameters, even with its 17 layers $(3 + 2 \times 7)$. As a comparison, in the architecture suggested by Zhang, Zhao & LeCun (2015) it is necessary to optimize 11,337,988 parameters.

Besides being faster to optimize at each epoch, once we are dealing with word representation composed of codified characters, best accuracy is achieved sooner. We analyzed 10 epochs in our implementation, but 5 to 7 epochs its enough to converge. Zhang, Zhao & LeCun (2015) converges in at least 30 epochs and Conneau et al. (2016) in 10 to 15 epochs.

3.4 Earlier Attempts

In the Appendix A is available a paper with results of an earlier attempt using *Word101* strategy with 3 others neural network architectures.

The results were par with the work of Zhang, Zhao & LeCun (2015). They are good results, but using a *Deep Pyramid Convolutional Neural Network* architecture is definitely better. As the input representation is identical in all these experiments, we could attribute the improvement of the results completely to the change of architecture. Having better results in a deep architecture as Deep Pyramidal is a good indication that improvements in accuracy classification could be achieved using others deep architectures in the future.

But the main point to cite that earlier attempt is to demonstrate that our representation is architecture independent and have good results as in CNN as in RNN applications. This opens the door to more complex strategies already applied to word embeddings, like *Text Summarization* (GAMBHIR; GUPTA, 2017), *Answer Questions using Attention* (HERMANN et al., 2015) and *Generative Auto-encoder Networks* for text (ZHANG et al., 2017).

3.4.1 CNN1 topology

At first, we choose a network architecture that we usually use to classify text using an embedding created by *Word2vec* (MIKOLOV et al., 2013), the only difference is that instead of 300 features, we reduce the input size to 256. This architecture we named CNN1. It is based on concatenation of convolutions in a shallow way, inspired by the work of Kim (2014), who achieve state of the art results for some databases.

We trained this model for 5 epochs. The neural network architecture CNN1 is summarized in Figure 3.4a as a diagram.

3.4.2 CNN2 topology

Prompted by the positive outcome of CNN1, we decided to investigate others possible architectures. We created another shallow but wide convolution architecture following the recommendations of Zhang & Wallace (2015) for choosing parameters, executing training on dataset ag_news, for being the smallest of our test datasets. This architecture is composed by:

- *Convolution width filter*: a combination of region sizes near the optimal single best region size outperforms using multiple region sizes far from the optimal single region size (ZHANG; WALLACE, 2015). We scan the width from 1 to 7 comparing accuracy performance. In these evaluations, the convolution of width 1 was a better option.
- *Pooling size*: max pooling consistently performs better than alternative strategies for the task of sentence classification (ZHANG; WALLACE, 2015).

The neural network architecture CNN2 is summarized in Figure 3.4b as a diagram

3.4.3 LSTM topology

To illustrate the possibilities of applying the proposed encoding, we did an experiment using LSTMs (HOCHREITER; SCHMIDHUBER, 1997), similar to LSTM model described by Zhang & LeCun (2015), the difference is that instead of using *Word2vec* embedding (MIKOLOV et al., 2013), we used our representation. The architecture is very simple: an input layer of 128×256, followed by a LSTM layer of 300, a Dropout layer of 0.10 (SRIVASTAVA et al., 2014), a fully connected layer of 128 units and a softmax layer.

We trained this model for 5 epochs. This architecture is in general, twice slower than *Word101*. The neural network architecture LSTM is summarized in Figure 3.4c as a diagram.

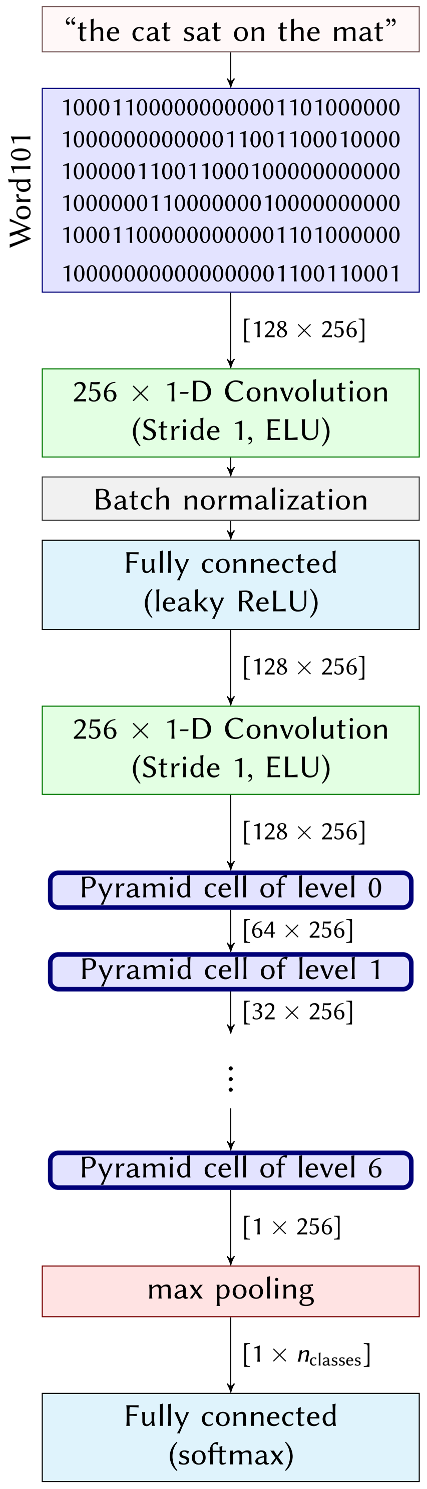
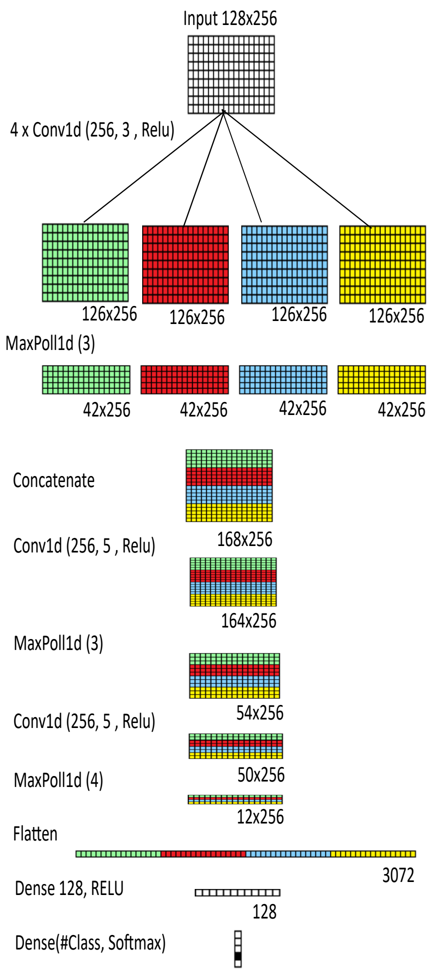
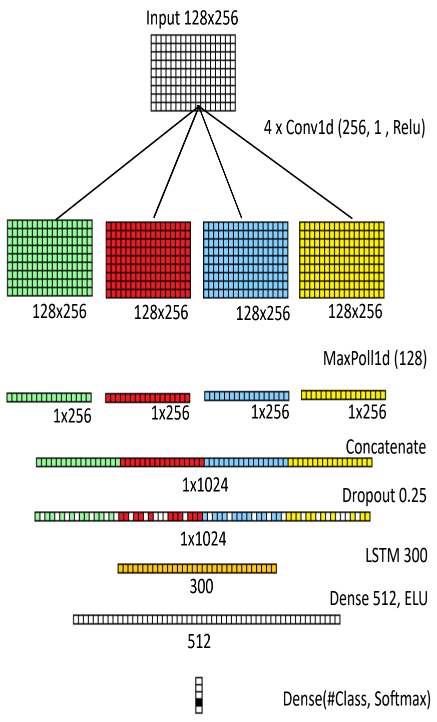


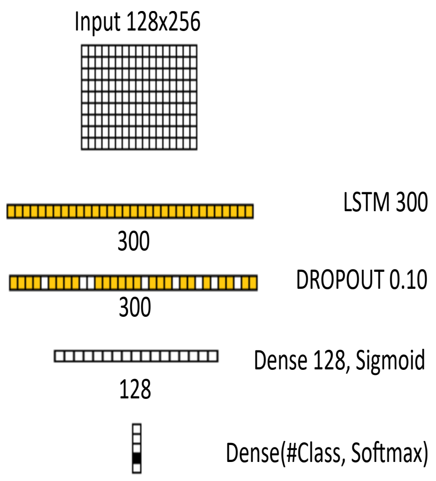
Figure 3.3: Deep pyramidal architecture.



(a) CNN1



(b) CNN2



(c) LSTM

Figure 3.4: Earlier attempt Network architectures used with Word101.

Chapter 4

Experimental Study

4.1 Applying Word101 to massive texts datasets

An essential part of this work is to contrast our proposal both within the context of character-level text classification and with other state-of-the-art approaches.

The datasets used were the same as those cited in an article by Zhang, Zhao & LeCun (2015), where there is an extensive description of them.[1] A detailed analysis of these datasets is out of the scope of this paper, instead, we will only summarize the main characteristics:

- **AG's news**: categorized news articles from more than 2000 news sources. Four classes (World, Sports, Business, SciTech).The dataset contains 120k train samples and 7.6k test samples equally distributed (ZHANG; ZHAO; LECUN, 2015).
- **Sogou news**: categorized news articles originally in Chinese. Zhang, Zhao & LeCun (2015) applied the *pypinyin* package combined with *jieba* Chinese segmentation system to produce *Pinyin*, a phonetic romanization of Chinese. Five classes (sports, finance, entertainment, automobile and technology). The dataset contains 450k train samples and 60k test samples equally distributed (ZHANG; ZHAO; LECUN, 2015).
- **DBpedia**: title and abstract from Wikipedia articles available in DBpedia crowd-sourced community (LEHMANN et al., 2015). Fourteen non-overlapping classes from DBpedia 2014. The dataset contains 560k train samples and 70k test samples equally distributed (ZHANG; ZHAO; LECUN, 2015).
- **Yelp full**: sentiment analysis from the Yelp Dataset Challenge in 2015[2]. Five classes

[1]For the sake of replicability we have made all the datasets available via <https://drive.google.com/open?id=1o5CNT0UHuFfHBxC-Mz4lmFpN2-Lcllmx>.

[2]https://www.yelp.com/dataset/challenge

representing the number of stars a user has given.The dataset contains 560k train samples and 38k test samples equally distributed (ZHANG; ZHAO; LECUN, 2015).

- **Yelp polarity**: sentiment analysis from the Yelp Dataset Challenge in 2015[2]. The original data is transformed into a polarity problem. Rating of 1 and 2 stars are represented as Bad and 4 and 5 as Good. The dataset contains 560k train samples and 50k test samples equally distributed. (ZHANG; ZHAO; LECUN, 2015).

- **Yahoo! answers**: questions and their answers from Yahoo! Answers. Ten classes (Society & Culture, Science & Mathematics, Health, Education & Reference, Computers & Internet, Sports, Business & Finance, Entertainment & Music, Family & Relationships, Politics & Government). Each sample contains question title, question content and best answer.The dataset contains 1,400k train samples and 60k test samples (ZHANG; ZHAO; LECUN, 2015).

- **Amazon full**: sentiment analysis from Amazon reviews dataset from the Stanford Network Analysis Project (SNAP) (MCAULEY; LESKOVEC, 2013). Five classes representing the number of stars a user has given. The dataset contains 3,000k train samples and 650k test samples (ZHANG; ZHAO; LECUN, 2015).

- **Amazon polarity**: sentiment analysis from Amazon reviews dataset from the Stanford Network Analysis Project (SNAP) (MCAULEY; LESKOVEC, 2013). Two classes, rating of 1 and 2 stars are represented as Bad and 4 and 5 as Good. The dataset contains 3,600k train samples and 400k test samples equally distributed (ZHANG; ZHAO; LECUN, 2015).

The baseline comparison models are the same as Zhang, Zhao & LeCun (2015). We just reproduce their results, the only difference is that they report loss error and for better comprehension, we translated it to accuracy. In Zhang, Zhao & LeCun (2015) there is an extensive description of them. In this paper, we just summarize the main information:

- **Bag of Words (BoW) and its term-frequency inverse-document-frequency (BoW TFIDF)**: For each dataset, they selected 50,000 most frequent words from the training subset. For the normal bag-of-words, they used the counts of each word as the features and for the TF-IDF they used the counts as the term-frequency (ZHANG; ZHAO; LECUN, 2015).

- **Bag-of-ngrams (Ngrams) and its TFIDF (Ngrams TFIDF)**: The bag-of-ngrams models were constructed by selecting the 500,000 most frequent n-grams (up to 5-grams) from the training subset for each dataset. The feature values were computed the same way as in the bag-of-words model (ZHANG; ZHAO; LECUN, 2015).

- **Long Short Term Memory (LSTM)**: The LSTM (HOCHREITER; SCHMIDHUBER, 1997) model used is word-based, using pretrained word2vec (MIKOLOV et al., 2013) embedding of size 300. The model is formed by taking a mean of the outputs of all LSTM cells to form a feature vector, and then using multinomial logistic regression on this feature vector. The output dimension is 512 (ZHANG; ZHAO; LECUN, 2015).

For all the experiments, we used the environment and parameters settings listed in Table 4.1. Besides the encoding procedure, we do not use any preprocessing strategy except the use of lowercase letters. We treated as a word any set of characters delimited with blank spaces and treat each punctuation (! "#$%&\ ' ( ) * + , - . / : ; <=>?@[\ ] ^_ ` { | } ~) as a separated word to compose text codified matrix. No data enhancement technique was employed.

As one of our concerns was to make our proposal as usable as possible on commodity hardware, we focused our studies in that hardware configuration. The major bottleneck for analyzing this large amount of matrix-encoded text is the need for intensive use of RAM. Our approach generates a 128×256 matrix, smaller than the 1014×69 generated by Zhang and Lecun Zhang, Zhao & LeCun (2015). In spite of that, a large set of them quickly occupies the available RAM on a 'regular' personal computer. On the machine we used, there were 16 GB available, which is not uncommon in modern personal computers. Therefore, the use of generators to control the number of matrices generated and sent to GPU is an important detail in the implementation of this optimization algorithm. If your computer has only 8 GB of RAM or less, maybe it will operate at the limit of necessary memory.

A comparison of accuracy with traditional models and the approaches of Zhang, Zhao & LeCun (2015), Xiao & Cho (2016), Conneau et al. (2016), Joulin et al. (2016), Johnson & Zhang (2017) and Johnson & Zhang (2016) is shown in Table 4.2. A time per epoch comparison among *Word101* and the works of Zhang, Zhao & LeCun (2015) and Conneau et al. (2016) is available in Table 4.3, which are models that also use character informations as inputs. Since time were evaluated in different machines, we could use this information only as a reference. The main point that we could highlight is that our implementation could be as fast as any implementation of word embeddings, since we are using the same strategy of one word by row.

Results show that accuracy of *Word101* is very competitive with character-based methods, but is at least one order of magnitude faster to compute.

4.1.1 Mixing Morphological structure and Semantic information

The main power of character level approaches is being able to have better responses in sets of texts in languages where there is not available a word embedding or this embedding do not address typos errors, slangs, neologisms, equations, etc. Since a word is a sequence of characters, any one of them will have a *Word101* representation, even if unseen on training data. The main premise is that the network is using the morphological construction of the word to guess the meaning of that word, or at least some subset of it, in case of declensions or typos errors, to compare to a known word present in training data.

But if we know in which language is the set of data of interest and have an embedding available, this could allow taking some semantic information into account. Even in degraded text constructions, many words could be correct and use an embedding pre-trained in a massive amount of texts could help in identify meaning. Since both information are more or less orthogonal, we could expect a better accuracy than just using one of them isolatedly.

As the majority of datasets studied in this paper use the English language, to illustrate this idea in a second experiment, we concatenated to the 256 code-word generated with *Word101* the pre-trained vector embedding of Glove (PENNINGTON; SOCHER; MANNING, 2014) using 100 dimensions, if available. In case that the correspondent word is not represented in Glove100, we just pad the last 100 dimensions of the vector with zeros.

The results in Table 4.2, *Word101+Glove100* show that there is a significant improvement in all datasets, except Sogou news. It is expected since this dataset is a romanization of Chinese Ideograms and there is not a representation available for its "words".

Once the input matrix became 128x356 instead of 128x256, the training time is in average 1.5 times slower.

Table 4.1: Training environment and parameters

DESCRIPTION	PARAMETERS	OBSERVATIONS
Neural Net Lib.	Keras 2.0.5	
Tensor Backend	Theano 1.0.1	
GPU Interface	Cuda 8	with cuBLAS Patch Update
CNN optimizer	Cudnn 5.1	
Program. Lang.	Python 3.6	using Anaconda 4.4.0
Minibatch	128	Batch to update the network weights
Optimizer	ADAM (ZEILER, 2012)	$lr = 10^{-3}$, $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$
Epochs	10	
Op. System	Ubuntu 16.04 LTS	
GPU	GeForce 1080ti	
RAM Memory	16 GB	

Table 4.2: Test set accuracy comparison among traditional models of Zhang, Zhao & LeCun (2015) models 'large' and 'small' and our approaches when applied to the AG's news (AG), Sogou news (SOGOU), DBpedia (DBP), Yelp polarity (YLP-P), Yelp full (YLP), Yahoo! answers (YAH), Amazon full (AMZ) and Amazon polarity (AMZ-P) datasets.

MODEL		AG	SOGOU	DBP	YLP-P	YLP	YAH	AMZ	AMZ-P
Tradicional Methods (ZHANG; ZHAO; LECUN, 2015)									
BoW		88.81	92.85	96.61	92.24	57.99	68.89	54.64	90.40
BoW TFIDF		89.64	93.45	97.37	93.66	59.86	71.04	55.26	91.00
Ngrams		92.04	97.08	98.63	95.64	56.26	68.47	54.27	92.02
Ngrams TFIDF		92.36	97.19	98.69	95.44	54.80	68.51	52.44	91.54
LSTM		86.06	95.18	98.55	94.74	58.17	70.84	59.43	93.90
Word-based Methods									
Johnson & Zhang (2017)		93.13	98.16	99.12	97.36	69.42	76.10	65.19	96.68
Johnson & Zhang (2016)		93.43	98.11	99.16	97.10	67.61	75.15	63.76	96.21
Joulin et al. (2016)		92.50	96.80	98.60	95.70	63.90	72.30	60.20	94.60
Character-based Methods									
Conneau et al. (2016)	17 depth	91.12	96.46	98.60	95.50	63.93	72.49	62.61	95.59
	29 depth	91.27	96.64	98.71	95.72	64.26	73.43	63.00	95.69
Xiao & Cho (2016)		91.36	95.17	98.57	94.49	61.82	71.74	59.23	94.13
Zhang & LeCun (2015)	Large	87.18	95.12	98.27	94.11	60.38	70.45	58.69	94.49
	Small	84.35	91.35	98.02	93.47	59.16	70.16	59.47	94.50
Our Proposal									
Word101		91.34	95.72	98.68	93.56	61.14	73.58	61.61	95.21
Word101 + Glove100d		93.41	95.76	98.97	94.13	62.13	75.15	62.50	95.66
Earlier attempts using Our Proposal									
Word101 CNN1		87.67	95.16	97.93	92.04	58.00	68.10	58.09	93.69
Word101 CNN2		91.43	93.96	98.03	91.53	57.03	70.24	55.72	91.23
Word101 LSTM		88.38	94.52	98.34	93.18	59.71	70.27	59.79	94.35

Table 4.3: Time per epoch as reported by Zhang, Zhao & LeCun (2015) and Conneau et al. (2016) models implemented in Torch 7 on a single NVidia K40 GPU and Word101 on an NVidia GeForce 1080ti GPU using the setup on Table 4.1. AG's news (AG), Sogou news (SOGOU), DBpedia (DBP), Yelp polarity (YLP-P), Yelp full (YLP), Yahoo! answers (YAH), Amazon full (AMZ) and Amazon polarity (AMZ-P) datasets.

DATASET	**TRAIN/TEST**	**Zhang, Zhao & LeCun (2015)**		**Conneau et al. (2016)**		**Word101**
	SIZE	**Small**	**Large**	**depth 17**	**depth 29**	**depth 17**
AG	120 k / 7.6 k	1 h	3 h	37 min	51 min	4 min
SOGOU	450 k / 60 k	N/A	N/A	41 min	56 min	26 min
DBP	560 k / 70 k	2 h	5 h	44 min	1 h	19 min
YLP-P	560 k / 38 k	N/A	N/A	43 min	1 h 09 min	23 min
YLP	650 k / 50 k	N/A	N/A	45 min	1 h 12 min	28 min
YAH	1,400 k / 60 k	8 h	1 days	1 h 33 min	2h	50 min
AMZ	3,000 k / 650 k	2 days	5 days	4 h 20 min	7 h	2 h 12 min
AMZ-P	3,600 k / 400 k	2 days	5 days	4h 25 min	7 h	2 h 13 min

Chapter 5

Conclusions

5.1 Final Remarks

In this dissertation is proposed an efficient encoding to represent text. That encoding is friendly to tensorial word representations using its character-level composition, especially in *Deep Learning* applications for tasks of *Natural Language Processing*. Since it makes use of a digit 1 to mark the begin and the end of a signal represented by some amount of 0 digits, we named it *Word101*.

This encoding is derived from the *Tagged Huffman* (MOURA et al., 2000) information compression method and inherited many of its advantages for applications in repositories of texts. The main idea behind this encoding is representing very frequent characters with shorter codes and leave longer codes to rare characters. It associates each character of a text to a binary representation, there is no information loss, but more importantly, no code representation is a subset of another, so concatenated codes could represent uniquely each word or any amount of text. Specifically, it allows a direct search or edition of an arbitrary section in a compressed document without the need of decompression. Since it only uses rank frequency of characters, *Word101* is quite stable in a given language and is not restricted to any vocabulary or alphabet size.

We have shown that concatenating codes using the proposed encoding is a convenient possibility to represent words, especially due to three great advantages:

First, it solves the "out-of-vocabulary problem", assigning one representation for any word occurring in our dataset of interest. This is particularly important as encoding text using characters can be relevant when dealing with less curated texts datasets, as it is robust to spelling errors, typos, slang, language variations, and others usual characteristics of Internet texts. Equations, chemical formulas, emoticons, ideograms and etc, could be represented and evaluated by this technique.

Second, similar characters sequences are represented by the same numerical sequence, so similar words are represented by vectors near each other. We argue that this procedure could take advantage of morphological constructions of words the same way we humans do. *Natural Language Processing* (NLP) is robust in the sense that small perturbations of the input do not affect processing negatively. Such perturbations include letter insertions, deletions, substitutions and transpositions and the insertion of spaces and the deletion of spaces. Unfamiliar words are systematically related to familiar ones due to linguistic processes such as morphology, phonology, abbreviation, copying error, and historical change (BLUNSOM et al., 2017). Furthermore, this morphological information could be concatenated with others word embeddings, like *Glove* (PENNINGTON; SOCHER; MANNING, 2014), allowing the model to take into account pre-trained semantic information. Since both information are more or less orthogonal, we could expect a better accuracy than just using one of them isolatedly.

Third, using a binary representation is possible to reduce the amount of storage space and RAM needed for applications reducing data transfer latencies and enabling use in commodity hardware. It allows to permanently store codified texts in a repository and accomplish classifications or other NLP tasks without the need of decompression. Furthermore, this encoding is useful for reducing the dimensions of tensorial representations and being these the main responsible for the number of operations needed in *Deep Learning* optimization algorithms, it allows an expressive reduction in training time.

In order to study the impact of this encoding, we coupled it with a deep architecture developed by Johnson & Zhang (2017) for classifying texts using word representations, a *Deep Pyramid Convolutional Neural Networks* architecture. These experiments showed that we managed to achieve a performance superior to base traditional methods and the one achieved by Zhang, Zhao & LeCun (2015) and Xiao & Cho (2016) and competitive results with Conneau et al. (2016), which uses only character inputs. By concatenating pre-trained word representations available in Glove (PENNINGTON; SOCHER; MANNING, 2014), we could achieve competitive results even with models that use word inputs as Joulin et al. (2016) and in some datasets, similar even with Johnson & Zhang (2016, 2017) that are the state of art today.

Furthermore, we show that this strategy is more computational efficient, comparing with Zhang & LeCun (2015) and Conneau et al. (2016), that uses characters as inputs. Using a simpler setup, we could do a classification on the same repository of texts in at least half of training time.

5.2 Future Directions

The word-based approach of Johnson & Zhang (2017) consistently have better results that our character-based approach. They use a region embedding to represent words as vectors using a one-hot encoding strategy in an unsupervised way. In the near future, we will focus on devising a similar strategy using *Word101* to generate region embeddings that may further improve the results.

It must be outlined the fact that this compact numeric representation of text is not limited to the domain of CNN or neural networks, for that matter. Earlier attempts using more shallow architectures demonstrate that our representation is architecture independent and have good results as in CNN as in RNN applications. This opens the door to other information-theoretic-based methods for text representation and uses in more complex strategies already applied to word embeddings, like *Text Summarization* (GAMBHIR; GUPTA, 2017), *Answer Questions using Attention* (HERMANN et al., 2015) and *Text Generative Auto-encoder Networks* (ZHANG et al., 2017).

It could be interesting to assess its impact on text representation for others natural languages tasks.

References

ABADI, M. et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems.* 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

AGGARWAL, C. C.; ZHAI, C. X. A survey of text classification algorithms. In: AGGARWAL, C. C.; ZHAI, C. X. (Ed.). *Mining Text Data*. [S.l.]: Springer US, 2012. p. 163–222. ISBN 978-1-4614-3222-7.

BAKER, F. B. *The Basics of Item Response Theory*. [S.l.]: Heinemann, 1985. ISBN 0435080040.

BENGIO, Y. et al. A neural probabilistic language model. *Journal of Machine Learning Reseach*, JMLR.org, v. 3, p. 1137–1155, 3 2003. ISSN 1532-4435. Disponível em: <http://dl.acm.org/citation.cfm?id=944919.944966>.

BENGIO, Y.; SIMARD, P.; FRASCONI, P. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, IEEE Press, Piscataway, NJ, USA, v. 5, n. 2, p. 157–166, 3 1994. ISSN 1045-9227. Disponível em: <http://dx.doi.org/10.1109/72.279181>.

BERGSTRA, J. et al. Theano: A CPU and GPU math expression compiler. In: *Proceedings of the Python for Scientific Computing Conference (SciPy)*. [S.l.: s.n.], 2010.

BLOOM, B. S. The 2 sigma problem: The search for methods of group instruction as effective as one-to-one tutoring. *Educational Researcher*, v. 13, n. 6, p. 4–16, 1984. Disponível em: <https://doi.org/10.3102/0013189X013006004>.

BLUNSOM, P. et al. From characters to understanding natural language (C2NLU): Robust end-to-end deep learning for NLP (Dagstuhl Seminar 17042). *Dagstuhl Reports*, Schloss Dagstuhl – Leibniz-Zentrum fuer Informatik, Dagstuhl, Germany, v. 7, n. 1, p. 129–157, 2017. ISSN 2192-5283. Disponível em: <http://drops.dagstuhl.de/opus/volltexte/2017/7248>.

BRISABOA, N. et al. An efficient compression code for text databases. *Advances in Information Retrieval*, Springer, p. 78–78, 2003.

CHO, K. et al. Learning phrase representations using rnn encoder–decoder for statistical machine translation. In: *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. Doha, Qatar: Association for Computational Linguistics, 2014. p. 1724–1734. Disponível em: <http://www.aclweb.org/anthology/D14-1179>.

CHOLLET, F. et al. *Keras*. [S.l.]: GitHub, 2015. Https://github.com/fchollet/keras.

CHUNG, J. et al. Empirical evaluation of gated recurrent neural networks on sequence modeling. *CoRR*, abs/1412.3555, 2014. Disponível em: <http://arxiv.org/abs/1412.3555>.

COLLOBERT, R. et al. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, v. 12, p. 2493–2537, 2011.

CONNEAU, A. et al. Very deep convolutional networks for natural language processing. *CoRR*, abs/1606.01781, 2016. Disponível em: <http://arxiv.org/abs/1606.01781>.

CORBETT, A. T.; ANDERSON, J. R. Knowledge tracing: Modelling the acquisition of procedural knowledge. *User Model. User-Adapt. Interact.*, v. 4, n. 4, p. 253–278, 1995. Disponível em: <http://dblp.uni-trier.de/db/journals/umuai/umuai4.html\#CorbettA95>.

DENG, J. et al. ImageNet: A large-scale hierarchical image database. In: *CVPR09*. [S.l.: s.n.], 2009.

FREEDMAN, R.; ALI, S. S.; MCROY, S. Links: What is an intelligent tutoring system? *Intelligence*, ACM, New York, NY, USA, v. 11, n. 3, p. 15–16, set. 2000. ISSN 1523-8822. Disponível em: <http://doi.acm.org/10.1145/350752.350756>.

GAMBHIR, M.; GUPTA, V. Recent automatic text summarization techniques: A survey. *Artif. Intell. Rev.*, Kluwer Academic Publishers, Norwell, MA, USA, v. 47, n. 1, p. 1–66, jan. 2017. ISSN 0269-2821. Disponível em: <https://doi.org/10.1007/s10462-016-9475-9>.

GOLDBERG, Y. A primer on neural network models for natural language processing. *CoRR*, abs/1510.00726, 2015. Disponível em: <http://arxiv.org/abs/1510.00726>.

GOODFELLOW, I.; BENGIO, Y.; COURVILLE, A. *Deep Learning*. Cambridge, MA: MIT Press, 2016. Http://www.deeplearningbook.org.

HARRIS, D.; HARRIS, S. *Digital design and computer architecture*. 2nd. ed. San Francisco, CA, USA: Morgan Kaufmann, 2012.

HE, K. et al. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015. Disponível em: <http://arxiv.org/abs/1512.03385>.

HERMANN, K. M. et al. Teaching machines to read and comprehend. *CoRR*, abs/1506.03340, 2015. Disponível em: <http://arxiv.org/abs/1506.03340>.

HOCHREITER, S.; SCHMIDHUBER, J. Long short-term memory. *Neural computation*, MIT Press, v. 9, n. 8, p. 1735–1780, 1997.

HOTHO, A.; NRNBERGER, A.; PAA, G. A brief survey of text mining. *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, v. 20, n. 1, p. 19–62, maio 2005. ISSN 0175-1336. Disponível em: <http://www.kde.cs.uni-kassel.de/hotho/pub/2005/hotho05TextMining.pdf>.

HUFFMAN, D. A. A method for the construction of minimum-redundancy codes. *Proceedings of the Institute of Radio Engineers*, v. 40, n. 9, p. 1098–1101, September 1952.

JOHNSON, R.; ZHANG, T. Convolutional neural networks for text categorization: Shallow word-level vs. deep character-level. *CoRR*, abs/1609.00718, 2016. Disponível em: <http://arxiv.org/abs/1609.00718>.

JOHNSON, R.; ZHANG, T. Deep pyramid convolutional neural networks for text categorization. In: BARZILAY, R.; KAN, M. (Ed.). *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics, ACL 2017, Vancouver, Canada, July 30 - August 4, Volume 1: Long Papers*. Association for Computational Linguistics, 2017. p. 562–570. Disponível em: <https://doi.org/10.18653/v1/P17-1052>.

JONES, K. S. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, v. 28, p. 11–21, 1972.

JOULIN, A. et al. Bag of tricks for efficient text classification. *CoRR*, abs/1607.01759, 2016. Disponível em: <http://arxiv.org/abs/1607.01759>.

JOZEFOWICZ, R.; ZAREMBA, W.; SUTSKEVER, I. An empirical exploration of recurrent network architectures. In: *Proceedings of the 32Nd International Conference on International Conference on Machine Learning - Volume 37*. JMLR.org, 2015. (ICML'15), p. 2342–2350. Disponível em: <http://dl.acm.org/citation.cfm?id=3045118.3045367>.

JURAFSKY, D.; MARTIN, J. H. *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition*. 1st. ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2000. ISBN 0130950696.

JURAFSKY, D.; MARTIN, J. H. *Speech and language processing*. London: Pearson, 2014. v. 3.

KHAJAH, M.; LINDSEY, R. V.; MOZER, M. C. How deep is knowledge tracing? *CoRR*, abs/1604.02416, 2016. Disponível em: <http://arxiv.org/abs/1604.02416>.

KIM, Y. Convolutional neural networks for sentence classification. *CoRR*, abs/1408.5882, 2014. Disponível em: <http://arxiv.org/abs/1408.5882>.

KOSALA, R.; BLOCKEEL, H. Web mining research: a survey. *SIGKDD Explor. Newsl.*, ACM Press, New York, NY, USA, v. 2, n. 1, p. 1–15, June 2000. Disponível em: <http://dx.doi.org/10.1145/360402.360406>.

KRIZHEVSKY, A.; SUTSKEVER, I.; HINTON, G. E. Imagenet classification with deep convolutional neural networks. In: *Advances in Neural Information Processing Systems (NIPS)*. [S.l.: s.n.], 2012. p. 1097–1105.

LECUN, Y.; BENGIO, Y. The handbook of brain theory and neural networks. In: ARBIB, M. A. (Ed.). Cambridge, MA, USA: MIT Press, 1998. cap. Convolutional Networks for Images, Speech, and Time Series, p. 255–258. ISBN 0-262-51102-9. Disponível em: <http://dl.acm.org/citation.cfm?id=303568.303704>.

LeCun, Y. et al. Gradient-based learning applied to document recognition. In: *Proceedings of the IEEE*. [S.l.: s.n.], 1998. p. 2278–2324.

LEHMANN, J. et al. DBpedia - A large-scale, multilingual knowledge base extracted from wikipedia. *Semantic Web Journal*, v. 6, n. 2, p. 167–195, 2015.

MANNING, C. D.; SCHÜTZE, H. *Foundations of statistical natural language processing*. Boston, MA: MIT press, 1999.

MARINHO, W.; MARTÍ, L. Review of student proficiency modeling techniques for use in intelligent tutoring systems. In: *Workshop de Pesquisa e Desenvolvimento em Inteligêcia Artificial, Inteligêcia Collectiva e Ciêcia de Dados*. [s.n.], 2016. Disponível em: <http://www.addlabs.uff.br/workpedia2016/anais-do-workpedia-2016/>.

MCAULEY, J.; LESKOVEC, J. Hidden factors and hidden topics: Understanding rating dimensions with review text. In: *Proceedings of the 7th ACM Conference on Recommender Systems*. New York, NY, USA: ACM, 2013. (RecSys '13), p. 165–172. ISBN 978-1-4503-2409-0. Disponível em: <http://doi.acm.org/10.1145/2507157.2507163>.

MIKOLOV, T. et al. Distributed representations of words and phrases and their compositionality. In: *Advances in Neural Information Processing Systems (NIPS)*. [S.l.: s.n.], 2013. p. 3111–3119.

MITTAL, S.; VETTER, J. S. A survey of CPU-GPU heterogeneous computing techniques. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 47, n. 4, p. 69:1–69:35, jul. 2015. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/2788396>.

MOURA, E. Silva de et al. Fast and flexible word searching on compressed text. *ACM Trans. Inf. Syst.*, ACM, New York, NY, USA, v. 18, n. 2, p. 113–139, abr. 2000. ISSN 1046-8188. Disponível em: <http://doi.acm.org/10.1145/348751.348754>.

NORVIG, P. Inference in text understanding. In: *Proceedings of the Sixth National Conference on Artificial Intelligence - Volume 2*. AAAI Press, 1987. (AAAI'87), p. 561–565. ISBN 0-934613-42-7. Disponível em: <http://dl.acm.org/citation.cfm?id=1863766.1863797>.

OLAH, C. *Understanding LSTM Networks*. 2015. [Online; posted 27-August-2015]. Disponível em: <http://colah.github.io/posts/2015-08-Understanding-LSTMs/>.

OXFORD English Dictionary. [S.l.]: Oxford University Press, 2012.

PANE, J. F. et al. Effectiveness of cognitive tutor algebra i at scale. *Educational Evaluation and Policy Analysis*, v. 36, n. 2, p. 127–144, 2014. Disponível em: <https://doi.org/10.3102/0162373713507480>.

PAVLIK, P. I.; CEN, H.; KOEDINGER, K. R. Performance factors analysis –a new alternative to knowledge tracing. In: *Proceedings of the 2009 Conference on Artificial Intelligence in Education: Building Learning Systems That Care: From Knowledge Representation to Affective Modelling*. Amsterdam, The Netherlands, The Netherlands: IOS Press, 2009. p. 531–538. ISBN 978-1-60750-028-5. Disponível em: <http://dl.acm.org/citation.cfm?id=1659450.1659529>.

PENNINGTON, J.; SOCHER, R.; MANNING, C. D. Glove: Global vectors for word representation. In: *2014 Empirical Methods in Natural Language Processing Conference (EMNLP 2014)*. [s.n.], 2014. p. 1532–1543. Disponível em: <http://www.aclweb.org/anthology/D14-1162>.

PIECH, C. et al. Deep knowledge tracing. In: CORTES, C. et al. (Ed.). *Advances in Neural Information Processing Systems 28*. Curran Associates, Inc., 2015. p. 505–513. Disponível em: <http://papers.nips.cc/paper/5654-deep-knowledge-tracing.pdf>.

SAHAMI, M. et al. *A Bayesian Approach to Filtering Junk E-Mail*. 1998.

SALTON, G.; MCGILL, M. *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, 1983.

SANTOS, C. D.; ZADROZNY, B. Learning character-level representations for part-of-speech tagging. In: XING, E. P.; JEBARA, T. (Ed.). *Proceedings of the 31st International Conference on Machine Learning*. Bejing, China: PMLR, 2014. (Proceedings of Machine Learning Research, v. 32), p. 1818–1826. Disponível em: <http://proceedings.mlr.press/v32/santos14.html>.

SEBASTIANI, F. Machine learning in automated text categorization. *ACM Computing Surveys*, ACM, New York, NY, USA, v. 34, n. 1, p. 1–47, 3 2002. ISSN 0360-0300. Disponível em: <http://doi.acm.org/10.1145/505282.505283>.

SIMONYAN, K.; ZISSERMAN, A. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014. Disponível em: <http://arxiv.org/abs/1409.1556>.

SRIVASTAVA, N. et al. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, v. 15, p. 1929–1958, 2014. Disponível em: <http://jmlr.org/papers/v15/srivastava14a.html>.

SU, Y. et al. Exercise-enhanced sequential modeling for student performance prediction. In: *AAAI*. [S.l.]: AAAI Press, 2018.

SZEGEDY, C. et al. Going deeper with convolutions. In: *Computer Vision and Pattern Recognition (CVPR)*. [s.n.], 2015. Disponível em: <http://arxiv.org/abs/1409.4842>.

Wikipedia contributors. *Wikipedia, The Free Encyclopedia*. 2004. Disponível em: <https://en.wikipedia.org>.

WILSON, K. H. et al. Back to the basics: Bayesian extensions of IRT outperform neural networks for proficiency estimation. *CoRR*, abs/1604.02336, 2016. Disponível em: <http://arxiv.org/abs/1604.02336>.

XIAO, Y.; CHO, K. Efficient character-level document classification by combining convolution and recurrent layers. *CoRR*, abs/1602.00367, 2016. Disponível em: <http://arxiv.org/abs/1602.00367>.

ZEILER, M. D. ADADELTA: An adaptive learning rate method. *CoRR*, abs/1212.5701, 2012.

ZHANG, X.; LECUN, Y. Text understanding from scratch. *arXiv preprint arXiv:1502.01710*, 2015.

ZHANG, X.; ZHAO, J.; LECUN, Y. Character-level convolutional networks for text classification. In: *Advances in neural information processing systems*. [S.l.: s.n.], 2015. p. 649–657.

ZHANG, Y. et al. Adversarial feature matching for text generation. In: PRECUP, D.; TEH, Y. W. (Ed.). *Proceedings of the 34th International Conference on Machine Learning*. International Convention Centre, Sydney, Australia: PMLR, 2017. (Proceedings of Machine Learning Research, v. 70), p. 4006–4015. Disponível em: <http://proceedings.mlr.press/v70/zhang17b.html>.

ZHANG, Y.; WALLACE, B. C. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *CoRR*, abs/1510.03820, 2015. Disponível em: <http://arxiv.org/abs/1510.03820>.

APPENDIX A – Publications

Following a list of publications in the Machine Learning area that have been accepted.

MARINHO, W., MARTÍ, L. and SANCHEZ-PI N. (2018, July). A Compact Encoding for Efficient Character-level Deep Text Classification. In A. Editor, B. Editor, & C. Editor. Title of Published Proceedings. Paper presented at International Joint Conference on Neural Networks (IJCNN), Rio de Janeiro. New Jersey: IEEE Press. **Qualis A1**

MARINHO, W.; MARTÍ, L. Review of Student Proficiency Modeling Techniques for use in Intelligent Tutoring Systems. In: Workshop de Pesquisa e Desenvolvimento em Inteligência Artificial, Inteligência Coletiva e Ciência de Dados. [s.n.], 2016. Disponível em: <http://www.addlabs.uff.br/workpedia2016/anais-do-workpedia-2016/>.

A Compact Encoding for Efficient Character-level Deep Text Classification

Wemerson Marinho
Institute of Computing
Universidade Federal Fluminense
Niteroi, Brazil
wmarinho@ic.uff.br

Luis Martí
Institute of Computing
Universidade Federal Fluminense
Niteroi, Brazil
lmarti@ic.uff.br

Nayat Sanchez-Pi
Institute of Mathematics and Statistics
Universidade do Estado do Rio de Janeiro
Rio de Janeiro, Brazil
nayat@ime.uerj.br

Abstract—**This paper puts forward a new text to tensor representation that relies on information compression techniques to assign shorter codes to the most frequently used characters. This representation is language-independent with no need of pretraining and produces an encoding with no information loss. It provides an adequate description of the morphology of text, as it is able to represent prefixes, declensions, and inflections with similar vectors and are able to represent even unseen words on the training dataset. Similarly, as it is compact yet sparse, is ideal for speed up training times using tensor processing libraries. As part of this paper, we show that this technique is especially effective when coupled with convolutional neural networks (CNNs) for text classification at character-level. We apply two variants of CNN coupled with it. Experimental results show that it drastically reduces the number of parameters to be optimized, resulting in competitive classification accuracy values in only a fraction of the time spent by one-hot encoding representations, thus enabling training in commodity hardware.**

Index Terms—**text classification, character level convolutional neural networks, encoding of words**

I. Introduction

Document classification is one of the principal tasks addressed in the context of natural language processing [1]. It implies associating a document —or any text fragment, for that matter— with a category or label relying on their content. The increasing availability of texts in digital form, especially through the Internet, has called for the development of statistical and artificial intelligence tools for automating this process. Spam detectors, sentiment analysis, news archiving, among many others, demand high-quality text classifiers.

There is a broad range of approaches to document classification (see [1]–[4]). An important portion of them relies on a representation that handles words as the atomic element of text. Consequently, those methods carry out their analysis through statistics of words occurrence [5]. However, the variability of words and structures belonging to a language hinders the viability of this method. Because of this, these models have a superior performance in specific domains and applications, where the vocabulary is or can be restricted to a relatively small number of words, possibly chosen by a specialist. Furthermore, such modeling becomes specific to a language, causing the replication process in another language to be carried out from scratch [5].

In recent years, we have experienced a revolution in the machine learning with the advent of deep learning methods [6]. The development of convolutional neural networks (CNNs) [7] coupled with the popularization of parallel computing libraries (e. g. Theano [8], Tensorflow [9], Keras [10], etc.) that simplify general-purpose computing on graphics processing units (GPGPU) [11] has been successful in tackling image classification problem [12] quickly becoming the state of the art of the field.

As it could be expected, the success of deep learning and CNNs in the image classification domain has prompted the interest to extend the deep learning principles to the document classification domain. Some existing methods have been updated but the clear majority are still based on the tokenization of words and the inference of their statistics. Bag of Words (BoW) [13] and Word2vec [14] are some of the most popular strategies.

It can be argued that the replication of image classification success in the documents domain faces as main challenge the difficulty of representing text as numerical tensors.

To address this issue, [5] suggested a groundbreaking approach that considers the characters as the atomic elements of a text. In particular, they represented the text as a sequence of one-hot encoded characters. This encoding provides a robust, language-independent representation of texts as matrices, that are then used as inputs of different CNNs. Their experimental results showed that this approach was able to attain and, in some cases, improve the state of the art results in complex text classification problems. More recently, reference [15] improved those results by combining CNNs with Long Short-Term Memories (LSTMs) [16]. In spite of that, the impact of this idea is hampered by the large computational demands of the approach, since its training can take days per epoch in relatively complex problems.

Character-level representations have the potential of being more robust than word-level ones. On the other hand, they are computationally more expensive because detecting syntactic and semantic relationships at the character-level is more expensive [17]. One possible solution could be a word representation that incorporates the character-level information.

In this paper, we propose an efficient character-level encoding of word to represent texts derived from the Tagged Huffman [18] information compression technique. This encoding takes into account the character appearance frequency in the texts in order to assign shorter codes to the most frequently used ones. This novel text encoding makes the idea put forward by [5] more computationally accessible by reducing its training requirements in terms of time and memory.

The proposed encoding makes possible to represent larger portions of texts in a less sparse form, without any loss of information, while preserving the ability to encode any word, even those not present in the training dataset ones. In order to study the impact of this encoding, we coupled it with two CNN architectures. The experimental studies performed showed that we managed to achieve a performance similar or in some cases better than the state of the art at a fraction of the training time even if we employed a simpler hardware setup.

Our main contribution is to show that this novel character-level text encoding produces a reduced input matrix, leading to a substantial reduction in training times while producing comparable or better results in terms of accuracy than the original approach by [5]. This opens the door to more complex applications, the use of devices with lower computational power and the exploration of other approaches that can be coupled with input representation.

The rest of the paper is structured as follows. In the next section, we deal with the theoretical foundations and motivation that are required for the ensuing discussions. There we also analyze the alternatives to character-level text compression that were taken into account for producing our proposal. After that, in Section III, we describe the encoding procedure and the neural network architectures that will take part of the experiments. Subsequently, in Section IV, we replicate the experiments of [5] in order to contrast our proposal with theirs under comparable conditions. Finally, in Section V, we provide some final remarks, conclusive comments and outline our future work directions.

The algorithms and test problems implementations supporting the findings of this paper are available on https://github.com/rio-group/compact-character-level-rep.

II. Preliminaries

The success in using Convolutional Neural Networks (CNNs) [7] in image classification [12] flourish with the development of many libraries [8], [10], [19], techniques and hardware. The effort to use CNNs for text classification tasks is justified by the possibility of appropriating these tools for obtaining better results and more robust algorithms, facilitating the use of the same approach for several applications.

There are two usual approaches to use CNNs for handling textual information: the (i) bag of words (BoW) [13] and (ii) Word2vec [14] approaches.

In the case of BoW and some of its variants, for representing each word in a vocabulary of size N, a digit 1 is placed in the correspondent position of that word in a $1 \times N$ vector, all others positions remaining with digit 0. Since natural languages usually have a large vocabulary, a limited subset of the vocabulary must be used in order to make viable to perform the necessary computations in terms of memory requirements. The chosen subset of the vocabulary must be representative of the texts. Therefore, in practical problems, a great deal of attention is devoted to this matter. In particular, it is common to involve an application domain specialist or use some kind of word frequency statistic or relevance metric, where the most frequent and rare words are excluded.

In the word2vec approach, each word is projected via a metric embedding of fixed size, representing its co-occurrence in a large corpus of text in the same language of the texts of interest. It is possible to use pretrained vectors or readjust the representation with new words. The main problem with both strategies is that it does not allows to represent words that are not in the training dataset. Typos, spelling errors, mixed up letters and text written in languages with a complex structure (declensions, conjugations, etc.) are completely ignored.

Establishing the character as the basic unit of text formation provides a better opportunity to be robust to typos, acceptance of neologisms, and other textual forms as equations and chemical formulas, abbreviations and idiosyncrasies of written language on the internet, such as emoticons, slang and dialects of the online world, etc. Assuming the word as a base item, much of this ability is lost, especially when models assume that text producers use a formal language.

Reference [5] put forward an important innovation in this regard. They represent text as a sequence of characters, not words. Consequently, they are capable of reducing the vocabulary of symbols to the size of the alphabet (69 in the case of the paper) and thus allowing the use of one-hot encoding [20]. In this paper, they represented a text as a matrix of size 1014×69 where each row corresponds to a position in the text sequence and columns with the presence or not of a given character. Therefore, a row with a 1 in a given column indicates that the presence of the corresponding character in that point of the text. With this representation on hand, they applied a CNN and obtained results competitive with other techniques, and in some cases improving the state of the art at their release. However, the main drawback is a large computational requirement, that in some cases called for days per training epoch.

The results obtained by them suggest that language, and therefore, text, can be treated as a sequence of signals, like any other [5]. However, the training times and the dimension of the matrices to be computed are still obstacles to the most effective use of the method. That is why a better encoding of text could be a right path towards a substantial improvement of this issue.

III. Compressed Encoding for CNN-based Text Classification

Searching for a way to reduce these training time while retaining the flexibility and power of character-level convolutional to classify text, we found a way to better encode texts. Our approach, with competitive accuracy, achieve a significant reduction in time execution from hours to minutes and from days to hours, by epoch.

To achieve better performance in execution, at first we though of using some form of encoding each character, and use the same approach of [5] but we realize that using a variable length encoding for each word could be more efficient. To do this we need a way to encode each char, generating distinct concatenated code for representing each word and that words with the same prefix were near each other, especially to respond well to declensions.

A. Compressed Representations

Although the Huffman encoding [21] yields shortest encoding possible, it does not generate unique representations once we concatenate encoded characters to form a word. We investigated encoding of [22] and found promising alternatives.

Our approach is based on the Tagged Huffman encoding [18], where a pair of '0' digits is the signal and the digit '1' is the tag of beginning and end code, the only difference is that for our approach, we need a shorter version to reduce the size of input matrix, so we choose to use only one digit 0 instead of two for each char, marking the beginning and the end of each char code with a digit 1, the same way.

As in the approach by [18], the coding we employ has the following advantages [22]:

1) No character code is a prefix of another, that is, the match is one-to-one.
2) It allows a direct search, that is, to search for a word in an encoded document, just encode the word and use traditional methods of comparing strings with the encoded document.
3) This encoding approach is a compression technique, so it also allows saving already encoded text documents permanently using a binary system, requiring less storage space.

These advantages become attractive especially if the goal is to extract knowledge about files in a repository to perform various classifications on different dimensions with the same files.

A possible advantage of this encoding over others strategies that use a word as an atomic representation of text is better respond to unseen words on training data, once that the network at least have some prefix to guess the meaning of the word, the same way we humans do. This is especially interesting in languages where there are a lot of declensions like Portuguese, Spanish, Italian, French, Russian, German, Arabic and Turkish, for example.

The coding procedure is not restricted to any size of vocabulary, the only problem is that less frequent characters will generate bigger codes, consequently, bigger encoding matrix. If your database has a lot of them, you could use a higher word code size.

B. Encoding Procedure

In all of our experiments, we used the following procedure to encode words:

- *Obtaining a character frequency rank*: We read the text database and count the frequency of each character, generating a list sorted by frequency of occurrence. Then we create a rank with only the relative positions of the characters. For a given language, this rank is quite stable since only the order of the rank is used. This means that if all documents are in the same idiom, this procedure can be replaced by a list with the characters rank of frequency for that language.
- *Creating a mapping from character to compress code*: To encode each character, we insert the digit 0 in the same amount of the rank position of the character, between a 1 digit to signal the begin and another 1 digit to signal the end of the code. Table I have some examples of encoded characters.

TABLE I: Example of coding using English language ranking of characters. Characters shown are the ones used in the subsequent examples.

CHARACTER	FREQ. RANK	COMPRESSED ENCODING
␣	0	11
e	1	101
a	2	1001
t	3	10001
i	4	100001
s	5	1000001
n	7	100000001
r	8	1000000001
d	10	100000000001
h	11	1000000000001
c	12	10000000000001
g	17	1000000000000000001
⋮	⋮	⋮
	n	$'1' + n \times '0' + '1'$

TABLE II: Example of coding using English language ranking of characters. Prefix common to more than one word are underscored

TEXT	ENCODED TEXT
<u>scien</u>ce	<u>1000001100000000000011000011011000000001</u>10000000000001101
<u>scien</u>tist	<u>1000001100000000000011000011011000000001</u>10001100001100000110001
<u>art</u>	<u>1001100000000110001</u>
<u>art</u>ist	<u>1001100000000110001</u>100001100000110001
u2	100000000000000011000000000000000000000000000000001

To encode each word, we just concatenate the codes of each character. As an example, we provide in Table II some examples of plain text words and their corresponding encoding.

Given a document, we consider that it is composed of words, being those any set of characters limited by the space character. This means 'word' could be math equations, web addresses, LaTeX code, computer programming commands, etc. In Table II we can see that this encoding could represent words with the same prefix in vectors that have same initial coordinates. Slangs like *tl;dr : too long, did not read* and *u2 : you too* as well mathematical expressions like e^a could be manipulated.

In a matrix of $number\ of\ words \times code\ size$ representing the document, each row represents a properly encoded word, where the code is embedded with its first symbol in the first column. Columns that were not occupied are padded with 0, larger codes are represented up to the chosen limit.

```
10000000011011000001101100110000000011000000000000110000000000010
10000110000010000000000000000000000000000000000000000000000000000
10011000000010000000000000000000000000000000000000000000000000000
10011000000001100010000000000000000000000000000000000000000000000
10011000000001100000000010000000000000000000000000000000000000000
10010000000000000000000000000000000000000000000000000000000000000
10000011000000000000011000011011000000001100000000001101000000000
00000000000000000000000000000000000000000000000000000000000000000
```

Fig. 1: Matrix encoding of sentence 'Research is an art and a science'. The text is encoded one word per row. Each word is underscored for easy identification.

TABLE III: Architectures of the 'large' and 'small' CNNs used by [5].

CONVOLUTIONS				
Layer	**Large Feature**	**Small Feature**	**Kernel**	**Poll**
1	1024	256	7	3
2	1024	256	7	3
3	1024	256	3	—
4	1024	256	3	—
5	1024	256	3	—
6	1024	256	3	3

FULLY CONNECTED			
Layer	**Large Feature Out**	**Small Feature Out**	**Dropout**
7	2048	1014	0.5
8	2048	1014	0.5
9	Depends on the problem		

Unoccupied lines are filled with 0 and larger documents are represented only up to the chosen maximum number of words, ignoring the last remaining ones. As an example, we represent a document in an 8×65 matrix in Figure 1.

In the example in Figure 1 we used a 8×65 matrix (520 elements) to encode the text with a certain amount of slack. At the very least, we would need 7×64 (448 elements). In contrast, the approach of [5] would employ at least 32×69 (2208) elements to represent the same sentence.

In our experiments, 256 coordinates were enough to represent 99.5% of the words from one of the databases studied. In all datasets studied in this paper, we choose 128 as a limit of words to represent a document, encoding each text in a 128×256 matrix.

1) Convolutional Network Model: As mentioned before, this work was prompted by the results of [5]. In their original approach, they encode each character using one-hot encoding in a vocabulary of 69 elements. The non-space characters are letters, numbers and punctuation. The model is composed of 9 layers, 6 of convolutions and 3 fully connected. Their architecture is described in Table III.

They used stochastic gradient descent (SGD) with a mini-batch of size 128, using momentum 0.9 and initial step size 0.01 which is halved every 3 epochs for 10 times. So, their results were obtained in at least 30 epochs.

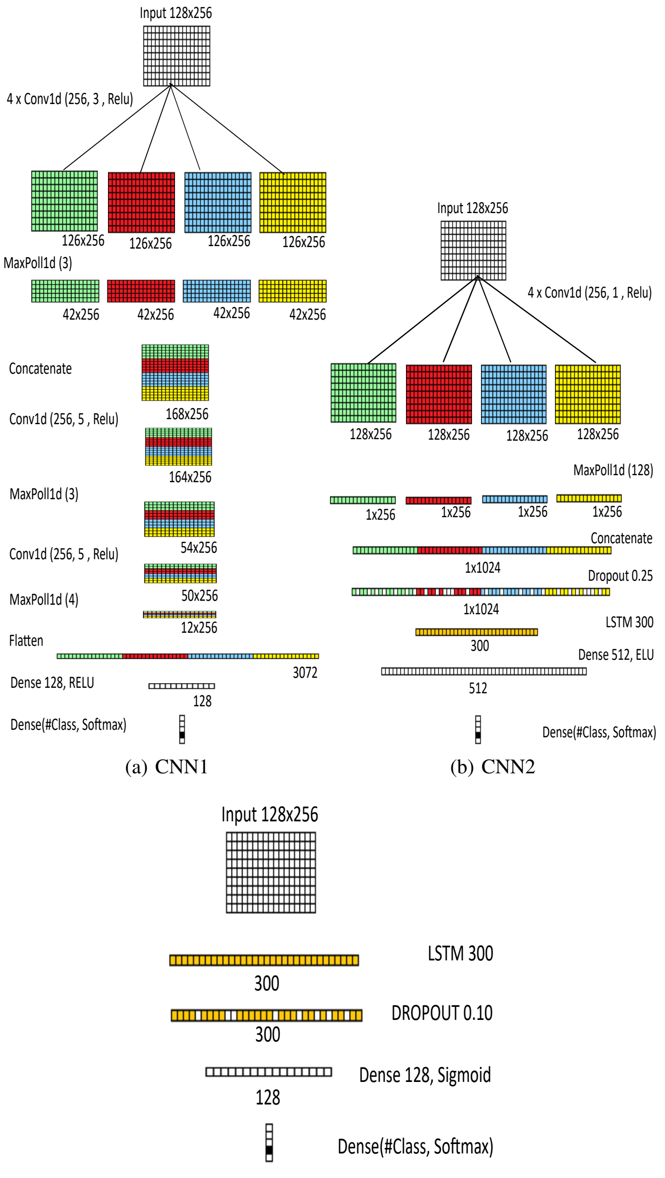


Fig. 2: Network architectures used coupled with the compressed encoding.

C. Neural Network Architectures

To verify the efficiency of the encoding procedure, we realized 3 experiments, named CNN1 , CNN2 and LSTM:

D. CNN1 topology

At first, we choose a network architecture that we usually use to classify text using an embedding created by word2vec [14], the only difference is that instead of 300 features, we reduce the input size to 256. This architecture we named CNN1. It is based on concatenation of convolutions in a shallow way, inspired by work of [23], who achieve state of the art results for some databases.

We trained this model for 5 epochs. The neural network architecture CNN1 is summarized in Figure 2a as a diagram.

E. CNN2 topology

Prompted by the positive outcome of CNN1, we decided to investigate others possible architectures. We created another shallow but wide convolution architecture following the recommendations of [24] for choosing parameters, executing training on dataset ag_news, for being the smallest of our test datasets. This architecture is composed by:

- *Convolution width filter*: a combination of region sizes near the optimal single best region size outperforms using multiple region sizes far from the optimal single region size [24]. We scan the width from 1 to 7 comparing accuracy performance. In these evaluations, the convolution of width 1 was a better option.
- *Pooling size*: max pooling consistently performs better than alternative strategies for the task of sentence classification [24].

The neural network architecture CNN2 is summarized in Figure 2b as a diagram

F. LSTM topology

To illustrate the possibilities of applying the proposed encoding, we did an experiment using LSTMs [16], similar to LSTM model described by [25], the difference is that instead of using word2vec embedding [14], we used our representation. The architecture is very simple: an input layer of 128×256, followed by a LSTM layer of 300, a Dropout layer of .10 [26], a fully connected layer of 128 units and a softmax layer.

We trained this model for 5 epochs. This architecture is in general, twice slower than CNN1. The neural network architecture LSTM is summarized in Figure 2c as a diagram.

IV. Experimental Study

An essential part of this work is to contrast our proposal both within the context of character-level text classification and with other state-of-the-art approaches.

The databases used were the same as those cited in an article by [5], where there is an extensive description of them.[1] A detailed analysis of these datasets is out of the scope of this paper, instead, we will only summarize the main characteristics:

- **AG's news**: categorized news articles from more than 2000 news sources. Four classes (World, Sports, Business, SciTech).The dataset contains 120k train samples and 7.6k test samples equally distributed [5].
- **Sogou news**: categorized news articles originally in Chinese. [5] applied the *pypinyin* package combined with *jieba* Chinese segmentation system to produce Pinyin , a phonetic romanization of Chinese. Five classes (sports, finance, entertainment, automobile and technology). The dataset contains 450k train samples and 60k test samples equally distributed [5].
- **DBpedia**: title and abstract from Wikipedia articles available in DBpedia crowd-sourced community [27]. Fourteen non-overlapping classes from DBpedia 2014. The dataset contains 560k train samples and 70k test samples equally distributed [5].
- **Yelp full**: sentiment analysis from the Yelp Dataset Challenge in 2015[2]. Five classes representing the number of stars a user has given. The dataset contains 560k train samples and 38k test samples equally distributed. [5].
- **Yelp polarity**: sentiment analysis from the Yelp Dataset Challenge in 2015[2]. The original data is transformed into a polarity problem. Rating of 1 and 2 stars are represented as Bad and 4 and 5 as Good. The dataset contains 560k train samples and 50k test samples equally distributed. [5].
- **Yahoo! answers**: questions and their answers from Yahoo! Answers. Ten classes (Society & Culture, Science & Mathematics, Health, Education & Reference, Computers & Internet, Sports, Business & Finance, Entertainment & Music, Family & Relationships, Politics & Government). Each sample contains question title, question content and best answer. The dataset contains 1,400k train samples and 60k test samples [5].
- **Amazon full**: sentiment analysis from Amazon reviews dataset from the Stanford Network Analysis Project (SNAP) [28]. Five classes representing the number of stars a user has given. The dataset contains 3,000k train samples and 650k test samples. [5].
- **Amazon polarity**: sentiment analysis from Amazon reviews dataset from the Stanford Network Analysis Project (SNAP) [28]. Two classes, rating of 1 and 2 stars are represented as Bad and 4 and 5 as Good. The dataset contains 3,600k train samples and 400k test samples equally distributed. [5].

The baseline comparison models are the same of [5] where there is an extensive description of them, we just reproduce their results, the only difference is that they report loss error and for better comprehension, we translated it to accuracy. In [5] there is an extensive description of them. In this paper, we just summarize the main information:

- **Bag of Words (BoW) and its term-frequency inverse-document-frequency (BoW TFIDF)**: For each dataset, they selected 50,000 most frequent words from the training subset. For the normal bag-of-words, they used the counts of each word as the features and for the TFIDF they used the counts as the term-frequency [5].
- **Bag-of-ngrams (Ngrams) and its TFIDF (Ngrams TFIDF)**: The bag-of-ngrams models were constructed by selecting the 500,000 most frequent n-grams (up to 5-grams) from the training subset for each dataset. The feature values were computed the same way as in the bag-of-words model [5].

[1]For the sake of replicability we have made all the datasets available via https://drive.google.com/open?id=1o5CNT0UHuFfHBxC-Mz4ImFpN2-Lcllmx.

[2]https://www.yelp.com/dataset/challenge

TABLE IV: Training environment and parameters

DESCRIPTION	PARAMETERS	OBSERVATIONS
Neural Net Lib.	Keras 2.0	
Tensor Backend	Theano 0.9	
GPU Interface	Cuda 8	with cuBLAS Patch Update
CNN optimizer	Cudnn 5.1	
Program. Lang.	Python 3.6	using Anaconda 4.4.0
Superbatch	10000	Number of matrixes sent to gpu each time
Minibatch	32	Batch to update the network weights
Optimizer	ADAM [29]	$lr = 10^{-3},\ \beta_1 = 0.9,\ \beta_2 = 0.999, \epsilon = 10^{-8}$
Epochs	5 , 12	
Op. System	Windows 10	
GPU	GeForce 1080ti	
RAM Memory	16 GB	

- **Bag-of-means on word embedding**: an experimental model that uses k-means on word2vec [14] learned from the training subset of each dataset, and then used these learned means as representatives of the clustered words. Took into consideration all the words that appeared more than 5 times in the training subset. The dimension of the embedding is 300. The bag-of-means features are computed the same way as in the bag-of-words model. The number of means is 5000 [5].
- **Long Short Term Memory (LSTM)**: The LSTM [16] model used is word-based, using pretrained word2vec [14] embedding of size 300. The model is formed by taking a mean of the outputs of all LSTM cells to form a feature vector, and then using multinominal logistic regression on this feature vector. The output dimension is 512 [5].

For all the experiments, we used the environment and parameters settings listed in Table IV. Besides the encoding procedure, we do not use any preprocessing strategy except the use of lowercase letters. No data enhancement technique was employed.

All the results and comparison with traditional models and the approaches of [5] is shown in Table V.

As tabular data are hard to grasp, we have decided to go for a graphical representation of the results. In particular, from the previous results, we have selected the large and small architectures of [5] and our CNN1, CNN2 and LSTM. Those values of accuracy were then scaled to the interval $[0, 1]$ for every dataset, being 0 the worst and 1 the best performance among all models, including traditional models . The outcome of this process is represented on Figure 3. On Table VI, we did a running time comparison, based on reports available by [5].

The main objective of this research was to evaluate the possibility of using a coding approach that contemplated the construction of words using characters as basic tokens. Our main contribution is demonstrate that such approach allows reducing the dimensionality of the encoding matrix, thus allowing substantially shorter optimization times and the use of devices with lower computational power. Some datasets of text have peculiarities that were not addressed by word frequency methods (i.e. BoW and word2vec), like declensions and new vocabulary. The article of [5] was a great innovation in this regard. However, the training times are still obstacles to the most effective use of the technique. We thought of a better representation should be a solution.

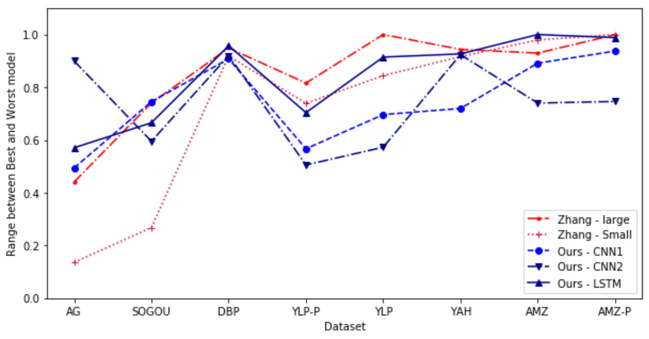


Fig. 3: Performance re-scaled in the range between best and worst model comparing [5] and our approaches when applied to the AG's news (AG), Sogou news (SOGOU), DBpedia (DBP), Yelp polarity (YLP-P), Yelp full (YLP), Yahoo! answers (YAH), Amazon full (AMZ) and Amazon polarity (AMZ-P) datasets.

To make a comparison, perform the training of architecture CNN1 with an output of 4 classes make necessary to optimize 1,837,188 parameters. As a comparison, in the architecture suggested by [5] it is necessary to optimize 11,337,988 parameters [5].

As one of our concerns was to make our proposal as usable as possible on commodity hardware, we focused our studies in that hardware configuration. The major bottleneck for analyzing this large amount of matrix-encoded text is the need for intensive use of RAM. Our approach generates a 128×256 matrix, smaller than the 1014×69 generated by [5]. In spite of that, a large set of them quickly occupies the available RAM on a 'regular' personal computer. On the machine we used, there were 16 GB available, which is not uncommon in modern personal computers. Therefore, the use of generators to control the number of matrices generated and sent to GPU is an important detail in the implementation of this optimization algorithm. If your computer has only 8 GB of RAM or less, it will be necessary to reduce the number of superbatchs to fit the memory.

The results obtained strongly indicate that the use of this coding is a possibility. We emphasize that we used is a fairly simple network, enough to demonstrate the feasibility of the encoding approach with the time and computational resources that we have.

The results are very competitive with the approach o [5] and traditional techniques. We see even that we could find parameters that achieve excellent performance on AG's news dataset, following the suggestions of [24]. One of advantage of a faster algorithm is that if your dataset is not so big, you could scan the feature width to find a solution that optimizes accuracy. Another advantage is the possibility to realize k-folds validation, to have a better perspective on how well it will perform for your specific dataset on real life.

TABLE V: Test set accuracy comparison among traditional models the [5] models 'large' and 'small' and our approaches when applied to the AG's news (AG), Sogou news (SOGOU), DBpedia (DBP), Yelp polarity (YLP-P), Yelp full (YLP), Yahoo! answers (YAH), Amazon full (AMZ) and Amazon polarity (AMZ-P) datasets.

Model		**AG**	**SOGOU**	**DBP**	**YLP-P**	**YLP**	**YAH**	**AMZ**	**AMZ-P**
BoW		88.81	92.85	96.61	92.24	57.99	68.89	54.64	90.40
BoW TFIDF		89.64	93.45	97.37	93.66	59.86	71.04	55.26	91.00
Ngrams		92.04	97.08	98.63	95.64	56.26	68.47	54.27	92.02
Ngrams TFIDF		92.36	97.19	98.69	95.44	54.80	68.51	52.44	91.54
Bag-of-Means		83.09	89.21	90.45	87.33	52.54	60.55	44.13	81.61
LSTM		86.06	95.18	98.55	94.74	58.17	70.84	59.43	93.90
Zhang et al. 2015	Large	87.18	95.12	98.27	94.11	60.38	70.45	58.69	94.49
	Small	84.35	91.35	98.02	93.47	59.16	70.16	59.47	94.50
Ours	CNN1	87.67	95.16	97.93	92.04	58.00	68.10	58.09	93.69
	CNN2	91.43	93.96	98.03	91.53	57.03	70.24	55.72	91.23
	LSTM	88.38	94.52	98.34	93.18	59.71	70.27	59.79	94.35

TABLE VI: Time per epoch as reported by [5] models and the ones used by CNN1 and CNN2 on an NVidia GeForce 1080ti GPU. AG's news (AG), Sogou news (SOGOU), DBpedia (DBP), Yelp polarity (YLP-P), Yelp full (YLP), Yahoo! answers (YAH), Amazon full (AMZ) and Amazon polarity (AMZ-P) datasets.

Dataset	**Train/Test sizes**	**Zhang et al, 2015 [5]**		**Ours**		
	SIZE	**LARGE**	**SMALL**	**CNN1**	**CNN2**	**LSTM**
AG	120 k / 7.6 k	1 h	3 h	3 min	4 min	7 min
SOGOU	450 k / 60 k	N/A	N/A	23 min	27 min	42 min
DBP	560 k / 70 k	2 h	5 h	18 min	20 min	36 min
YLP-P	560 k / 38 k	N/A	N/A	21 min	31 min	47 min
YLP	650k / 50 k	N/A	N/A	27 min	46 min	48 min
YAH	1,400 k / 60 k	8 h	1 days	47 min	55 min	1 h 37 min
AMZ	3,000 k / 650 k	2 days	5 days	2 h	2 h 30 min	3 h 53 min
AMZ-P	3,600 k / 400 k	2 days	5 days	2h 13 min	2 h 31 min	4 h 18 min

Another interesting point is that using a LSTM layer with the proposed encoding, we achieved similar, sometimes better results than using an embedding using word2vec [14] as proposed by [5]. Our approach by its own nature take into account morphological aspects of text while word2vec uses pre-trained vectors representing co-occurrence of words on a big corpus of text. Being able to take into account character information even in recurrent networks, we show that this representation is not limited to the domain of CNN or neural networks, for that matter.

Although our LSTM architecture is twice slower than our CNN1 and CNN2 topologies, it consistently outperform them. This indicates that temporal dependence among of words are important, so, potentially other architectures can generate better results taking this information to account and this is a direction that should be explored. In addition to that, the dimensionality reduction achieved by our encoding enables several other architectures and methods to be verified in a reasonable timeframe.

We are certain that our algorithm implementation could be even faster. For instance, when using a GPU Geforce 1080ti and a CNN1 architecture, each of the superbatch of 10,000 arrays have its weights updated in 30 seconds. Only 6 seconds is consumed by GPU, another 24 seconds is spent in encoding all the matrix and delivery it on the GPU. Using a multithread strategy could help in this regard.

V. Final Remarks

In this paper, we have proposed an efficient character-level encoding for text derived from the Tagged Huffman [18] information compression method and applied it as input preprocessing step for character-level CNN text classification.

We have shown that using this compression technique is a convenient possibility to represent text in a convolutional deep learning setup for text classification. This is particularly important as encoding text using characters can be relevant when dealing with less curated texts datasets, as it is robust to spelling errors, typos, slang, language variations, and others usual characteristics of Internet texts.

This novel text encoding allow to represent larger portions of texts in a compact form while preserving the ability to encode any word, even those not present in the training dataset ones. Furthermore, for being compact yet sparse, this approach dramatically reduces the time required for training CNNs and, therefore, makes the application of this novel approach more accessible. This opens the door to more complex applications, the use of devices with lower computational power and the exploration of other approaches that can be coupled with this input representation.

The experimental studies carried out coupled the encoding with two convolutional neural networks architectures and a recurrent LSTM architecture. These experiments showed that we managed to achieve a performance similar to the one achieved by [5] in a fraction of the training time even if we employed a simpler hardware setup. Furthermore, with our results, we endorse [5] conclusions, which state that language can be treated as a signal, no different from any other.

It should be noted that the main objective of the paper was to show the viability of the text encoding, not producing better results *per se*. Consequently, we have focused our efforts on the comparative study. Probably, custom neural network architectures should be devised to this new encoding with that purpose. Our results indicate that combining it with LSTMs is a promising direction in order to overcome the fixed matrix size limitation. In the near future, we will focus on devising these new architectures that may further improve the results.

This study also opens a door to other information-theoretic-based methods for information representation to be used to create a numerical representation of texts. It must be outlined the fact that this compact numeric representation of text is not limited to the domain of CNN or neural networks, for that matter. It could be interesting to assess its impact as a preprocessing step, perhaps with a minor modification, for other classification algorithms.

Acknowledgments

This work was supported in part by FAPERJ APQ1 Project 211.500/2015, FAPERJ APQ1 Project 211.451/2015, CNPq Universal 430082/2016-9, FAPERJ JCNE E-26/203.287/2017, Project Prociência 2017-038625-0, CNPq PQ 312792/2017-4 and the NVIDIA Corporation GPU Grant Program. W. M. is grateful to CAPES for the award of a research scholarship.

References

[1] F. Sebastiani, "Machine learning in automated text categorization," *ACM Computing Surveys*, vol. 34, no. 1, pp. 1–47, Mar. 2002. [Online]. Available: http://doi.acm.org/10.1145/505282.505283

[2] C. C. Aggarwal and C. X. Zhai, "A survey of text classification algorithms," in *Mining Text Data*, C. C. Aggarwal and C. X. Zhai, Eds. Springer US, 2012, pp. 163–222. [Online]. Available: http://dx.doi.org/10.1007/978-1-4614-3223-4_6

[3] A. Hotho, A. Nrnberger, and G. Paa, "A brief survey of text mining," *LDV Forum - GLDV Journal for Computational Linguistics and Language Technology*, vol. 20, no. 1, pp. 19–62, May 2005. [Online]. Available: http://www.kde.cs.uni-kassel.de/hotho/pub/2005/hotho05TextMining.pdf

[4] R. Kosala and H. Blockeel, "Web mining research: a survey," *SIGKDD Explor. Newsl.*, vol. 2, no. 1, pp. 1–15, June 2000. [Online]. Available: http://dx.doi.org/10.1145/360402.360406

[5] X. Zhang, J. Zhao, and Y. LeCun, "Character-level convolutional networks for text classification," in *Advances in neural information processing systems*, 2015, pp. 649–657.

[6] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[7] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," in *Proceedings of the IEEE*, 1998, pp. 2278–2324.

[8] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio, "Theano: a CPU and GPU math expression compiler," in *Proceedings of the Python for Scientific Computing Conference (SciPy)*, Jun. 2010, oral Presentation.

[9] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/

[10] F. Chollet *et al.*, "Keras," https://github.com/fchollet/keras, 2015.

[11] S. Mittal and J. S. Vetter, "A survey of CPU-GPU heterogeneous computing techniques," *ACM Computing Surveys*, vol. 47, no. 4, pp. 69:1–69:35, Jul. 2015. [Online]. Available: http://doi.acm.org/10.1145/2788396

[12] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems (NIPS)*, 2012, pp. 1097–1105.

[13] G. Salton and M. McGill, *Introduction to Modern Information Retrieval*. New York, NY, USA: McGraw-Hill, 1983.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in Neural Information Processing Systems (NIPS)*, 2013, pp. 3111–3119.

[15] Y. Xiao and K. Cho, "Efficient character-level document classification by combining convolution and recurrent layers," *CoRR*, vol. abs/1602.00367, 2016. [Online]. Available: http://arxiv.org/abs/1602.00367

[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[17] P. Blunsom, K. Cho, C. Dyer, and H. Schütze, "From characters to understanding natural language (C2NLU): Robust end-to-end deep learning for NLP (Dagstuhl Seminar 17042)," *Dagstuhl Reports*, vol. 7, no. 1, pp. 129–157, 2017. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2017/7248

[18] E. Silva de Moura, G. Navarro, N. Ziviani, and R. Baeza-Yates, "Fast and flexible word searching on compressed text," *ACM Trans. Inf. Syst.*, vol. 18, no. 2, pp. 113–139, Apr. 2000. [Online]. Available: http://doi.acm.org/10.1145/348751.348754

[19] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *CoRR*, vol. abs/1410.0759, 2014. [Online]. Available: http://arxiv.org/abs/1410.0759

[20] D. Harris and S. Harris, *Digital design and computer architecture*, 2nd ed. San Francisco, CA, USA: Morgan Kaufmann, 2012.

[21] D. A. Huffman, "A method for the construction of minimum-redundancy codes," *Proceedings of the Institute of Radio Engineers*, vol. 40, no. 9, pp. 1098–1101, September 1952.

[22] N. Brisaboa, E. Iglesias, G. Navarro, and J. Paramá, "An efficient compression code for text databases," *Advances in Information Retrieval*, pp. 78–78, 2003.

[23] Y. Kim, "Convolutional neural networks for sentence classification," *CoRR*, vol. abs/1408.5882, 2014. [Online]. Available: http://arxiv.org/abs/1408.5882

[24] Y. Zhang and B. C. Wallace, "A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification," *CoRR*, vol. abs/1510.03820, 2015. [Online]. Available: http://arxiv.org/abs/1510.03820

[25] X. Zhang and Y. LeCun, "Text understanding from scratch," *arXiv preprint arXiv:1502.01710*, 2015.

[26] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *Journal of Machine Learning Research*, vol. 15, pp. 1929–1958, 2014. [Online]. Available: http://jmlr.org/papers/v15/srivastava14a.html

[27] J. Lehmann, R. Isele, M. Jakob, A. Jentzsch, D. Kontokostas, P. N. Mendes, S. Hellmann, M. Morsey, P. van Kleef, S. Auer, and C. Bizer, "DBpedia - a large-scale, multilingual knowledge base extracted from wikipedia," *Semantic Web Journal*, vol. 6, no. 2, pp. 167–195, 2015. [Online]. Available: http://jens-lehmann.org/files/2015/swj_dbpedia.pdf

[28] J. McAuley and J. Leskovec, "Hidden factors and hidden topics: Understanding rating dimensions with review text," in *Proceedings of the 7th ACM Conference on Recommender Systems*, ser. RecSys '13. New York, NY, USA: ACM, 2013, pp. 165–172. [Online]. Available: http://doi.acm.org/10.1145/2507157.2507163

[29] M. D. Zeiler, "ADADELTA: An adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012. [Online]. Available: http://dblp.uni-trier.de/db/journals/corr/corr1212.html#abs-1212-5701