Language Identification: a Neural Network Approach

Alberto Simões[1], José João Almeida[2], and Simon D. Byers[3]

1 **Centro de Estudos Humanísticos, Universidade do Minho**
Braga, Portugal
ambs@ilch.uminho.pt

2 **Departamento de Informática, Universidade do Minho**
Braga, Portugal
jj@di.uminho.pt

3 **AT&T Labs**
Bedminster NJ, US
headers@gmail.com

Abstract

One of the first tasks when building a Natural Language application is the detection of the used language in order to adapt the system to that language. This task has been addressed several times. Nevertheless most of these attempts were performed a long time ago when the amount of computer data and the computational power were limited. In this article we analyze and explain the use of a neural network for language identification, where features can be extracted automatically, and therefore, easy to adapt to new languages. In our experiments we got some surprises, namely with the two Chinese variants, whose forced us for some language-dependent tweaking of the neural network. At the end, the network had a precision of 95%, only failing for the Portuguese language.

1998 ACM Subject Classification I.2.7 Natural Language Processing: Language models

Keywords and phrases language identification, neural networks, language models, trigrams

Digital Object Identifier 10.4230/OASIcs.SLATE.2014.251

1 Introduction

The problem of Language Identification has been addressed for a long time, usually as a language model, that validates how likely a text is modeled by a specific language model [4]. This task can be considered the base when building a natural language processing stack of tools, as before one can apply mostly any kind of language processing tool there is the need to know the text language or, at least, the text alphabet. Only after that identification is done we can apply a tool to the text being certain that it will know how to deal with the characters, the words, or the syntax.

Following the idea presented in the previous paragraph, we can divide the task of identifying a language in two main tasks: first, the alphabet identification (looking to which characters[1] are used) and second, the identification of the language itself.[2]

[1] We are aware that the notion of character change with different alphabets. In this article we refer to character as an entry in the Unicode table.

[2] Here we are simplifying, as there are some languages that can be written in two different alphabets. In this paper we will consider that each language has a preferred alphabet, and that is the one that will be used.

3rd Symposium on Languages, Applications and Technologies (SLATE'14).
Editors: Maria João Varanda Pereira, José Paulo Leal, and Alberto Simões; pp. 251–265

OpenAccess Series in Informatics
OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

At first the identification of some languages can be seen as simple. If a human looks to some Chinese text, he might notice that characters seem different from the ones used in Korean or Japanese text. In the same manner, Hindi characters are not likely to be found in other languages. The truth is that it is not as simple as it might seem, as for example, Chinese, Korean and Japanese share a huge amount of characters.

For the next level there are yet more problems. Consider the large amount of languages that share the Latin characters, and the amount of languages with the same origin, like Portuguese, Galician or Spanish. It can get harder if one tries to distinguish between language variants, like English from United States or United Kingdom.

In this paper we present a tool for learning language identification from tagged corpora into a Neural Network. Although this idea is not new, previous work on this task was published more than 20 years ago, and a lot has changed. Nowadays we have large quantities of text, in most any language existing in the world, and enough computational power to train a Neural Network is a relatively large amount of parameters. Also, and although we are not using that knowledge, there are new studies on methods to train Deep Neural Networks [5, 1], that we are interested to research about.

So, to start with, our main objective was to use a simple Neural Network implementation, making it easy to implement a language identifier in any programming language given the neural network parameters Θ. Then, as this approach is working, we intend to apply new techniques, namely the referred deep neural networks.

At the moment, and as a proof of concept we developed the learning algorithm in Octave (an open-source implementation of the well known MATLab software), and implemented language identifiers in two different programming languages: Perl and Java.

First we will analyze the current language identification approaches, namely the ones already using Neural Networks, and compare them with our approach. This will be discussed in the next section.

The section 3 describes the entire process of training the Neural Network, starting with the dataset preparation. Then, we will discuss how the features were chosen, and which were used. Follows the description of the Neural Network architecture, and its implementation details.

Section 4 will analyze how well the Neural Network performs in different kind of texts and for different language pairs.

Finally, section 5 presents some conclusions regarding our work, and pointers in future directions.

2 Language Identification Approaches

When looking up for works on language identification one might be surprised to find out that most of the recent published work is devoted to spoken language identification. Although the task might be similar, as the main algorithm would be to detect features most common in some languages than in another, the fact is that the searched features are of different kind. In this work we will focus only in the works devoted to identify languages on written text.

In the other hand, there are not many publications on language identification on written text. A reason for that might be the existence of two patents [9, 10], that explore the use of trigrams or generically n-grams for language identification. It is always interesting that some of these patents are granted when previous work like [6] already use this same kind of approach but for sound. Also, one year later, Nakagawa *et al.* [8] published work on language identification based on Hidden Markov Models that use n-grams as well.

In fact, Muthusamy already used neural networks for language identification. So, a question arises: what does our work do that was not done before? First, his main task was to identify language on spoken text. Other than that, in 1993 the amount of data available in text format was quite smaller than the amount of texts available nowadays, and the number of languages in which these texts exist is also very different. In another aspect, the computational power also changed drastically. Muthusamy used at most 10 languages, and not much more than 130 features, which were mostly chosen manually.

Our approach takes advantage of the amount of text available as well as the computational power to compute automatically what features to use. Our experiments gone up to more than one thousands features, having the network to took less than one day to train with a reasonable number of iterations.

3 Neural Network

Neural networks are being used for some time and their design and implementation for standard situations is well known [3]. Most of the work using neural networks aims at the classification of objects. In this case, the network works as a hypothesis function $h_\Theta(X)$ that, based on a set of matrices Θ previously computed on a training set, is able to classify an object based on a set of features extracted from that object. So, in the case of language identification, our training set is a set of texts manually classified in one language and an algorithm to extract features from them. These features are them fed in the neural network training algorithm that will compute the set of matrices Θ.

These matrices are then used to identify the language of new texts. For that, the features X are extracted from the text to be classified, and the hypothesis function is called. The resulting vector will include the probabilities of that text being identified as each one of the trained languages.

This section describes the main approach used to train our neural network. First we will discuss how the training texts were prepared. Follows the algorithm for extracting features from the training dataset, and the details on the neural network, explaining the architecture and the implementation details.

3.1 Dataset Corpora Preparation

In order to gather training data we used text from the TED conference website. This resulted in a core corpus of 105 different languages and language variants. These texts had very different sizes depending on the amount of data available in the TED website.

Given the technical nature of these texts, they include high proportion of technical terms, company and product names, and person names which are not translated. We will be referring to these as named entities [7], although some of them are not, at least in the usual definition of the term. This type of linguistic units is present to a varying degree in many language data sources.

This leads to the problem that text in a target language used for training might have snippets of another language appearing in it. This is exacerbated in translated text and in technical text. Also, in multilingual data on the same subject, particular word and character level features may appear in many languages despite being unrepresentative of most of them. When extracting n-grams, for example, it might happen that the most frequent are part of these terms. The result are features that are not language discriminant, although of high frequency.

In order to obtain clean training data we exploit the fact that the TED data form a multilingual parallel corpus. In particular the initial source language is English in this case. We extract the out-of-vocabulary words and some named entities from the English text using the *Hunspell*[3] spell checker and its default English dictionary. Note that we are extracting named entities that contain non-words, like proper names or trade marks. These words then, if they appear in the non-English tracks for that aligned text, should be removed due to their potential foreign origin.

This process allows us to obtain cleaner training text, where words are more likely to be purely of the tagged language. The drawback is that the resulting text no longer has correct sentences. Nevertheless, if we compute only character n-grams (and not word n-grams) that problem should not be relevant.

Finally, for the Portuguese language, we used the Lince [2] application in order to render the texts compliant to the 1990 Portuguese orthography reform, recently implemented. Given that this reform was established with the explicit goal of better unifying the orthography of the several variants of the Portuguese language worldwide, it is only natural that, in spite of the remaining differences, it has brought closer together the orthographies of European and Brazilian variants. Therefore, we might have considered Portuguese as an unique language and probably should have selected texts from only one of these variants. Nevertheless, we kept the two variants as distinct, and will discuss the obtained results later.

3.2 Feature Extraction

Our main goal, initially, was to use only n-gram features (namely trigrams) from the languages being used in the training process. Unfortunately, when using character trigrams, we are working with word trigrams for the Asiatic languages, like Korean, Chinese or Japanese, as each character represent (roughly) a word. This means that the amount of different trigrams for these languages is huge. To solve this problem we might enlarge the number of features extracted per language, thus making the training process prohibitive. Other option would be to change the number of trigrams for those specific languages. At the end we decided to create character dependent features (instead of some language-dependent features), regarding the number of characters used in some alphabets.

Therefore, currently we have two different levels of features: one related with the characters that are used, and another with the character trigram frequency information. All these features are extracted from 30 different texts for each one of the training languages.

Alphabet Features

As stated in the introduction, it is not possible to create an injective function from used characters to the written language neither from the language to the used characters.

For the Latin alphabet alone there are dozens of languages. For the Chinese, Japanese and Korean languages, they all use Chinese Kanji morphemic script, although Japanese script is sillabary, not an alphabet, and Korean uses a proper alphabet (phonologically based script). The situation gets worse when looking to the traditional and simplified Chinese versions that share most of their characters.

To compute features related to the used characters, we defined 10 different classes C_i:

[3] Details on the Hunspell spell checker and its dictionaries can be obtained from the project webpage at `http://hunspell.sourceforge.net/`

1. Latin characters, only `a-z`, without diacritics;
2. Cyrillic characters, containing Unicode characters in the intervals `0x0410-0x042F` and `0x0430-0x044F`;
3. Hiragana and Katakana characters (used for Japanese), containing Unicode characters between `0x3040-0x30FF`
4. The Hangul characters (used for Korean), from the Unicode classes `0xAC00-0xD7AF`, `0x1100-0x11FF`, `0x3130-0x318F`, `0xA960-0xA97F` and `0xD7B0-0xD7FF`;
5. Kanji characters (used in Japaneses, Korean and Chinese), from the Unicode class `0x4E00-0x9FAF`;
6. Simplified Chinese characters, a list of 2877 characters, hand-curated and available on GitHub[4];
7. Traditional Chinese characters, a list of 2663 characters, hand-curated and also availablefrom GitHub;
8. Arabic characters (used in Persian, Urdu, and different varieties of the Arabic language), in the Unicode class `0x0600-0x06FF`;
9. Thai characters, for the Unicode class `0x0E00-0x0E7F`;
10. Greek characters, in the Unicode classes `0x0370-0x03FF` and `0x1F00-0x1FFF`.

For the text segment being analyzed, the number of characters for each one of these classes are counted, and the relative frequency computed. After some experiments, and in order to reduce the entropy for the neural network, we decided to help by computing discrete values. Therefore, before using these ten values in the neural network a small set of rules make the values binary. When setting a class C_i, the result will have $C_i = 1$ and $C_j = 0, \forall j \neq i$.

Follows the list of rules used in this context:

$$
\begin{aligned}
set\ C_1 &\Leftarrow C_1 > 0.20 \\
set\ C_2 &\Leftarrow C_2 > 0.40 \\
set\ C_3 &\Leftarrow C_3 > 0.20 \\
set\ C_4 &\Leftarrow C_4 > 0.20 \\
set\ C_6 &\Leftarrow C_5 > 0.30 \wedge C_6 > C_7 \\
set\ C_7 &\Leftarrow C_5 > 0.30 \wedge C_6 < C_7 \\
set\ C_8 &\Leftarrow C_8 > 0.20 \\
set\ C_9 &\Leftarrow C_9 > 0.20 \\
set\ C_{10} &\Leftarrow C_{10} > 0.20
\end{aligned}
$$

These percentages were defined empirically. In fact, these rules are specially relevant for the Japanese, Korean and Chinese languages. Note that the two complicate rules are used to distinguish between the two Chinese variants. After running these rules, these features are used directly in the neural network.

Trigram Features

Regarding language information, we chose to store information about character trigrams. There are different reasons why we chose to use three characters:

[4] Check `https://github.com/jpatokal/script_detector`

```
Für mich war das eine neue Erkenntnis. Und ich denke, mit der Zeit, in den
kommenden Jahren, Wir haben Künstler, aber leider haben wir sie noch nicht
entdeckt. Der visuelle Ausdruck ist nur eine Form kultureller Integration.
Wir haben erkannt, dass seit kurzem immer mehr Leute
```

Figure 1 A sample text in the German language.

- bigrams would be too small when comparing very close languages like Portuguese and Spanish;
- tetragrams would be too big for Asiatic languages, where some gliphs represent words or morphemes;
- punctuation and numbers were removed, and spaces normalized, meaning that trigrams would be able to capture the end and beginning of two words that usually occur together, as well as to capture single character words that appear surrounded by spaces.

This task was performed using the Perl module `Text::Ngram`[5], which deals with the task of cleaning the text, normalizing spaces and computing n-grams. The obtained counts were then divided by the total number of trigrams found, thus computing their relative frequency.

As an example, Table 1 shows the result of computing trigrams on the text from Figure 1.

Table 1 Top 25 occurring trigrams from text shown in Figure 1.

en␣	0.02299	er␣	0.02682	␣de	0.01533	abe	0.01533	der	0.01149
hab	0.01149	ich	0.01149	ir␣	0.01149	it␣	0.01149	r␣h	0.01149
␣wi	0.01149	ben	0.01149	ch␣	0.01149	den	0.01149	wir	0.01149
␣ha	0.01149	ine	0.00766	ler	0.00766	lle	0.00766	n␣k	0.00766
mme	0.00766	ne␣	0.00766	nnt	0.00766	r␣l	0.00766	r␣m	0.00766

Features Merging

Although the alphabet features is a limited list of ten different alphabets, there is the need to merge the trigram features into just one list choosing only the more significant.

This process is performed in two stages, first for each language, then for the entire training set:

1. For each of the 30 training texts from a specific language we compute the 20 trigrams with higher frequency. The trigrams are then merged in an unique list that includes the most occurring trigrams from all the training texts in a specific language. Next, this list is reduced, preserving only the 20 trigrams that are present in most texts. Note that we are not interested in their frequency in each training text, but how often they appear in different texts.
2. Next, each group of 20 trigrams computed from a specific language are joined together in a big list of features.

So, the complete features list F includes the alphabet features (F_a) and the trigrams features (F_t): $F = F_a \cup F_t$. With this feature list we can compute the training data, in the form of a matrix. Each line of the matrix is the data collected from each one of the training

[5] Available from `https://metacpan.org/pod/Text::Ngram`.

Table 2 Training data matrix.

	Alphabet Features			Trigram Features						
	Latin	Greek	Cyril.	␣pa	ói␣	par	nia	ест	ати	ата
PT	1	0	0	0.0041	0	0.0038	0.0001	0	0	0
PT	1	0	0	0.0039	0	0.0036	0	0	0	0
RU	0	0	1	0	0	0	0	0.0020	0.0004	0.0003
RU	0	0	1	0	0	0	0	0.0026	0.0005	0.0002
UK	0	0	1	0	0	0	0	0.0003	0.0034	0.0001
UK	0	0	1	0	0	0	0	0.0003	0.0026	0.0001
VI	1	0	0	0	0.0028	0	0	0	0	0
VI	1	0	0	0	0.0029	0	0.0001	0	0	0

texts. Each column of the matrix corresponds to a different feature from F. Each cell of the matrix stores the value of a specific feature in a specific training text. Table 2 shows an excerpt from this matrix.

3.3 Network Architecture

A neural network is composed by a set of L layers, each one composed by a set or processing units. A processing unit is denoted by $a_i^{(l)}$ where l is the layer where it belongs, and i its order.

All units from a specific layer are connected to all units from the next layer. This connection is controlled by a matrix $\Theta^{(l)}$, for each layer l.

The first layer is known as the *input layer*. It has the same number of units as there are features to be analyzed (in our experiment, 565 units). Whenever the network hypothesis function is evaluated each cell $a_i^{(1)}$ is filled in with the values obtained by the features observation.

The next layer, $a_i^{(2)}$ is computed using the previous layer and the matrix $\Theta^{(1)}$, as will be explained in the next section. This process is done for every layer $l \leq L$.

The layer L is known as the *output layer*. There are as many units in this layer as the number of classes K in which the network will classify objects. Therefore, if the network is trained to detect 25 languages, then there are 25 units in the output layer. Each unit in the output layer will, optimally, get a value that is either 1 or 0, meaning that the object is, or is not, in the respective class. Usually, the result is a value in this range, that represent the probability of the object to be of that specific class.

The other layers, $1 < l < L$, are known as the *hidden layers*. There are as many hidden layers as one might want, but there is at least one hidden layer. Adding new layers will make the network return better results but it will take more time to train the network, and take more time to run the network hypothesis function. For our experiments we used only one hidden layer.

Regarding the number of units in the hidden layers, there are some rules of thumb: use the same number of units in all hidden layers, and use at least the same number of units as the maximum between the number of classes and the number of features. But there can be up to three times that value. Given the high number of features we opted to keep that same number of units in the hidden layer.

3.4 Training Details

This kind of neural network implementation is not complicated, but is susceptible to errors. Our neural network was implemented using the more common definition of a neural network [3].

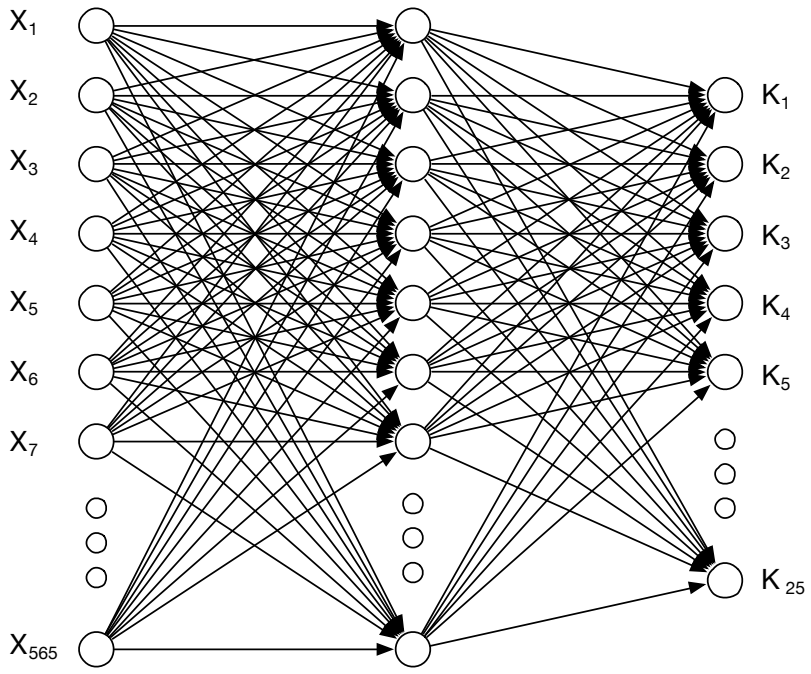

Figure 2 Neural network architecture.

The implementation of the neural network was based on the logistic function defined by $g(z)$. This function range is $[0, 1]$, and its result value can be considered a probability measure. The logistic function is defined as:

$$g(z) = \frac{1}{1 + \exp -z}$$

Our neural network hypothesis function, $h_{\Theta^{(l)}}(X)$ is defined by two matrices, $\Theta^{(1)}$ and $\Theta^{(2)}$. These matrices of weights are used to compute the network. The input values, obtained by the computed features, are stored in the vector X. This vector is multiplied by the first weight matrix, and the logistic function is applied to each value of the resulting vector. The resulting vector is denoted as $a^{(2)}$ and corresponds to the values of the second layer of the network (the hidden layer). It is then possible to multiply $a^{(2)}$ vector by the weights of $\Theta^{(2)}$ and, after applying the sigmoid function to each element of the resulting multiplication, we obtain $a^{(3)}$. This is the output layer, and each value of this vector corresponds to the probability of the document being analyzed to as being written in a specific language. This algorithm is known by *forward propagation* and is defined by:

$$\begin{aligned} &a^{(1)} = x \\ &\text{for } i = 2 \text{ to } L, \\ &\quad a^{(i)} = g\left(\Theta^{(i-1)} x\right) \end{aligned}$$

The main problem behind this implementation is how to obtain the weight values. For that the usual methodology is to define a cost function and try to minimize it, that is, finding the Θ values for which the hypothesis function has a smaller error for the training set.

The cost function with regularization is defined as:[6]

$$J(\Theta) = -\frac{1}{m}\left(\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)} \log(h_\Theta(x^{(i)}))_k + (1-y_k^{(i)}) \log(1-(h_\Theta(x^{(i)}))_k)\right) + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{S_l+1}\left(\Theta_{j,i}^{(l)}\right)^2 .$$

The regularization is controlled by the coefficient λ which can be used to tweak how the Θ weights absolute value will increase. Although our implementation supports regularization the experiments performed did not use any regularization ($\lambda = 0$).

The minimization of the cost function $J(\Theta)$ is computed by an algorithm known as *Gradient Descent*. This algorithm uses the partial derivatives

$$\frac{\partial}{\partial \theta_{i,j}^{(l)}} J(\Theta)$$

to compute the direction to use to obtain the function minimum. The algorithm continues iterating until the difference between the obtained costs is very small, or until a limit number of iterations it met.

Gradient Descent can be implemented using an algorithm known as *Backwards Propagation* to compute efficiently the partial derivatives. Our implementation runs a number of iterations and save the Θ values. It is then possible to continue the training from those values. In the future this will allow us to create a test set and stop training when it has a sufficiently high precision. Nevertheless, at the moment we are performing tests with a fixed number of iterations (check next section).

4 System Evaluation

Our experiment used 25 languages: Arabic (AR), Bulgarian (BG), German (DE), Modern Greek (EL), Spanish (ES), Persian (FA), French (FR), Hebrew (HE), Hungarian (HU), Italian (IT), Japanese (JA), Korean (KO), Dutch (NL), Polish (PL), Portuguese (PT), Brazilian Portuguese (PT-BR), Romanian (RO), Russian (RU), Serbian (SR), Thai (TH), Turkish (TR), Ukrainian (UK), Vietnamese (VI) and, Traditional and Simplified Chinese (ZH-TW and ZH-CN).

The neural network was trained using these 25 languages and the corpora described in section 3.1. The next subsection explains the creation and characterizes the test set for these languages. Note that although the training corpora was cleaned, removing some words that are not likely to be in that language, the test corpora is noisy (namely including some words from other languages).

4.1 Test Set Characterization

For each language to be identified we collected 21 documents. Given we do not master all these languages we had some difficulties on collecting documents for some languages. To be sure of the languages of the test files we often resorted to other language identification software. All the texts were collected from on-line newspapers. Therefore, the texts have

[6] It goes beyond of focus of this article to discuss and explain what is the regularization and how it works. The same is true regarding the Gradient Descent or the Backwards Propagation algorithms.

Table 3 Training and test set statistic for each language. Values are in number of Unicode characters.

	Training Set				Test Set			
Language	Smaller	Larger	$\bar{x}$	σ	Smaller	Larger	$\bar{x}$	σ
AR	871921	969387	907562	21392	863	4618	2366	1210
BG	988450	1087435	1027581	23663	660	2099	1091	378
DE	588200	653508	618463	16475	677	3890	1554	842
EL	773265	885770	841203	22653	550	3297	1590	705
ES	578806	651240	617341	17637	897	3850	2342	935
FA	651807	766206	697212	28994	600	5221	1338	967
FR	639582	705675	673414	15377	936	4088	1879	689
HE	806098	877218	836222	20545	559	3649	1586	878
HU	406271	454506	431797	13131	729	6045	2175	1356
IT	588147	643252	616391	14348	1260	6607	2991	1370
JA	538033	606053	569956	18871	323	785	495	133
KO	737118	817651	773168	20550	530	1603	780	233
NL	533497	580313	557724	14033	552	1949	1115	381
PL	521184	591299	551259	17938	435	3092	1605	694
PT-BR	596158	643215	617734	14028	920	3189	1953	589
PT	338272	378872	355800	10605	486	5875	2031	1169
RO	592714	650375	616051	15442	718	3254	1438	695
RU	1019789	1144200	1069884	31232	662	2470	1444	526
SR	349389	433221	379344	20560	834	6493	1813	1263
TH	529484	601244	565082	18551	334	3242	1396	734
TR	494191	549998	524271	12774	332	5390	1559	1121
UK	370785	434683	395312	16641	299	15435	2430	3553
VI	470057	541930	510409	17246	680	6237	1555	1359
ZH-CN	536438	595027	562728	14457	495	6331	1695	1559
ZH-TW	514993	588860	542879	16000	270	1721	925	428

plenty of named entities (that our training corpus misses) and vary on size. In fact, in some situations the news texts were not copied completely, in order to have smaller texts. Unfortunately the task of collecting these texts was done ad-hoc, resulting in some very different sizes for different languages. Check Table 3 for some more information on the number of characters per test file.

Curiously, when building this test set we found some texts that were being wrongly identified because we collected them in the wrong language. Although this fact is not relevant, it was curious that a collected text in Catalan was identified as French. This means that the neural network is able to detect languages by proximity.

4.2 Accuracy

Our first experiments did not include the alphabet features. Although it worked relatively well for most languages, the trained neural network failed for the four Asiatic languages. The main reason for that is the large proportion of characters that are shared among these languages, while each one has a structurally different type of base script. This leads to a large amount of different trigrams and therefore the neural network would need many more features per language (or for these specific language).

Table 4 Accuracy on test set, when training with 1500 and 4000 iterations.

Language	1500 iters.	4000 iters.	Comments
AR	100%	100%	
BG	100%	100%	
DE	100%	100%	
EL	100%	100%	
ES	100%	100%	
FA	100%	100%	
FR	100%	100%	
HE	100%	100%	
HU	100%	100%	
IT	100%	100%	
JA	100%	100%	
KO	100%	100%	
NL	100%	100%	
PL	100%	100%	
PT	**5%**	**52%**	wrongly classifies as PT-BR
PT-BR	100%	**76%**	wrongly classifies as PT
RO	100%	100%	
RU	100%	100%	
SR	100%	100%	
TH	100%	100%	
TR	100%	100%	
UK	100%	100%	
VI	100%	100%	
ZH-CN	100%	100%	
ZH-TW	100%	100%	

After adding the alphabet features, we trained the neural network with two different number of iterations: 1500, and 4000. Table 4 presents accuracy values for each language when analyzing the test set. Globally, with 1500 iterations we were able to get 96% of precision, and with 4000 iterations it gets up to 97%.

Looking to the results' table one can see that the most problematic languages are the two Portuguese variants, for which many texts are being attributed to the Brazilian variant. This is probably the result from the 1990 orthographic reform, whose aim was, precisely, an orthographic unification of the Portuguese language across its variants, just like the tests demonstrate.

In order to compare our (bad) results we did some experiments with the Perl module `Lingua::Identify::Blacklists` [11] that uses lists of words that are blacklisted for some languages. The results for the Portuguese variants were 66% of accuracy for the European variant, and 100% accuracy for the Brazilian language.

Looking to this module blacklists we found out that, more than identifying the variant, the tool identifies the *topic* of the text. For example, the module states that if a text includes the word "Brasília" (the capital of Brazil) or "Pará" (a state from Brazil), then the text should be in the Brazilian variant. It happens in the inverse direction as well, with other proper names, like "Madaíl" (a controversial person in the Portuguese soccer) or "Louçã" (a left-wing deputy from the Portuguese parliament). Also, the module uses a list of words that changed in the 1990 orthographic agreement, meaning that for new Portuguese texts they are useless.

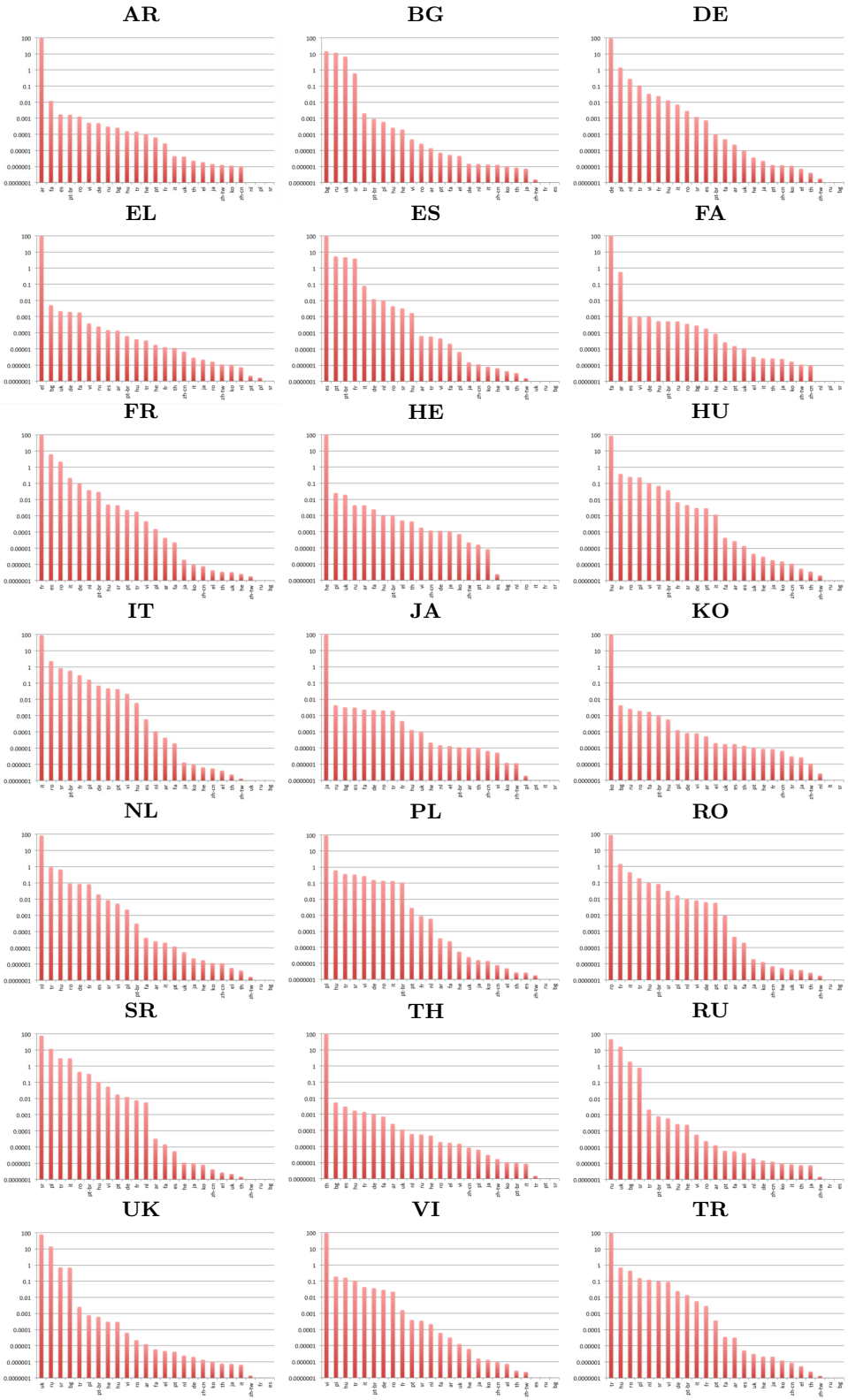

Figure 3 Language identification distribution.

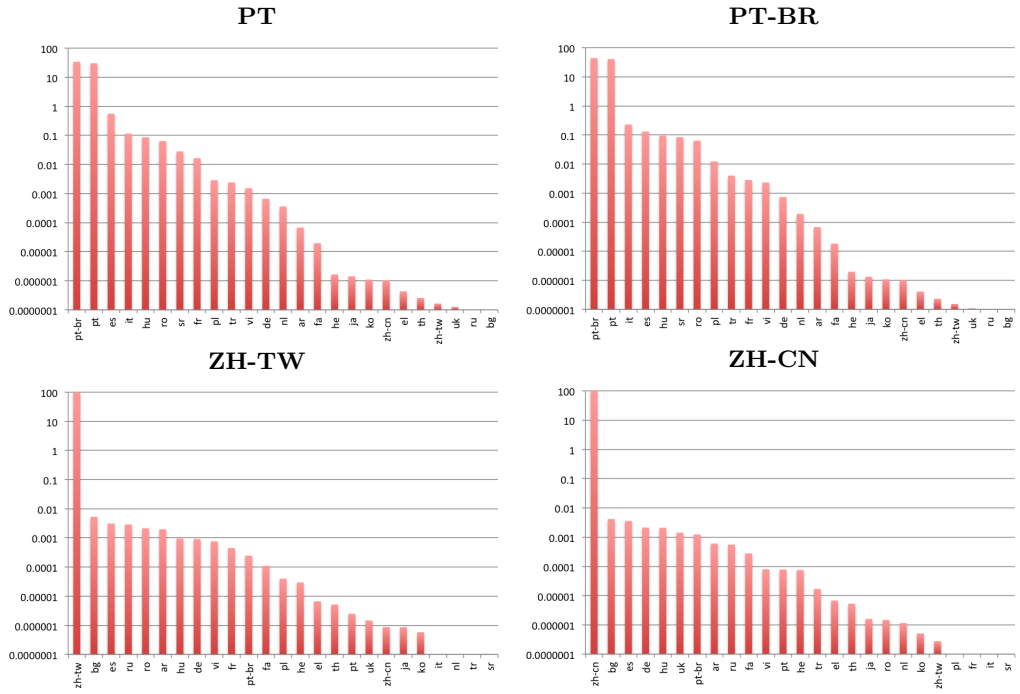

Figure 4 Language identification distribution for the two Portuguese and Chinese variants.

A good way to evaluate and compare the results from this module and our neural network would be the use of a good parallel corpus European/Brazilian Portuguese. This would allow us to evaluate the language identification and not the *topic* identification.

4.3 Probability Distribution

For each language we chose randomly one of the test files, and computed the language identification probabilities. Figure 3 show them for most languages.[7] Although the graphs are small and not readable, it is easy to notice that there is a big difference from the first language identified (the correct one) and the second choice. From these twenty one graphs the only relevant for analysis is the Bulgarian, which is very near Russian and Ukrainian.

Figure 4 presents the same graph for the remaining four languages, that include the two Portuguese and the two Chinese variants. Note that, for the Chinese variants, the difference from the first probability to the rest is very high. This is not a result of the trigrams features, but the fact that our alphabet identifier is working well to differentiate the two orthographies. Regarding the two Portuguese variants, it is clear the confusion between the European and Brazilian variants, with probabilities around 45%.

5 Conclusions and Future Work

In this article we present a neural network that is able to identify languages with 96% or 97% of accuracy, depending on the number of iterations performed during the training process.

[7] Note that graphs are using an exponential y axis.

For that we used two kind of features: one related with the language alphabet, and another related to the character trigrams with higher occurrence.

Given that we are able to use binary features to classify the alphabet (at the moment we have ten binary features) and they are mutually exclusive, the neural network is able to learn much faster to distinguish some collections of languages.

A problem with our approach is that it will perform badly on short snippets of text (like instant messages or mobile messages), because of the low number of trigrams selected by language. We are investigating how to deal with this problem without compromising the time needed to train the neural network.

Regarding the problem with the Portuguese variant we are mostly convinced to merge the two variants in a single one, given that with the so mentioned Orthographic Agreement it does not make sense to keep distinguishing between the two.

On using a neural network, we should be reminded that the result is not deterministic: the same number of iterations to train a network might yield different results, depending on the values used to initialize the Θ matrices.

Future Work

The next (certain) steps on this project would be (and probably, in this order):

1. Remove the Brazilian Portuguese and/or merge it with the European Portuguese variant;
2. Add the English language, that was not included at first because of some technical problems when preparing the training corpora;
3. Release the Perl and Java identification modules publicly;
4. Add more languages;
5. Go to point 3, and iterate.

Nevertheless, every time we train the neural network we find new experiments we would like to perform. These steps are likely to be done, but in any order:

- Try to reduce the number of trigrams per language and add some bigrams or one-grams. These tests' main rationale would be to reduce the number of features, as adding new languages are likely to include more features and make the training process slower.
- Compute distribution differences between near languages and, instead of using just the more occurring trigrams, use those that are most distinctive;
- In order to make the neural network smaller, train a different neural network for each alphabet. This will allow modularization when making the language identifier available. The user could then download only the modules relevant for her task.
- Our experiments with more than 4000 iterations gave worst results than the ones presented here. This happens because the algorithm is not using any regularization, and therefore the neural network is being biased by the training data and is unable to generalize. Further experiments are needed to study good values for the regularization coefficient.
- Neural networks are known to have difficulties to scale. Nevertheless, recent work in deep learning [5, 1], and deep neural networks might be relevant to analyze and use.

Acknowledgments. The authors would like to thank Catarina Sousa for the help compiling the test dataset, and the three reviewers, Lluís Padró, António Teixeira and Jorge Baptista, for their comments, insights and corrections.

References

1 Yoshua Bengio. Learning Deep Architectures for AI. *Foundations and Trends® in Machine Learning*, 2(1):1–127, January 2009.

2 José Pedro Ferreira, António Lourinho, and Margarita Correia. Lince, an end user tool for the implementation of the spelling reform of Portuguese. In Helena de Medeiros Caseli, Aline Villavicencio, António J. S. Teixeira, and Fernando Perdigão, editors, *Computational Processing of the Portuguese Language - 10th International Conference (PROPOR)*, volume 7243 of *Lecture Notes in Computer Science*, pages 46–55. Springer, 2012.

3 Simon Haykin. *Neural networks: a comprehensive foundation*. Prentice Hall PTR, 1994.

4 Daniel Jurafsky and James H. Martin. *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. Prentice-Hall, second edition edition, 2009.

5 Hugo Larochelle, Yoshua Bengio, Jérôme Louradour, and Pascal Lamblin. Exploring strategies for training deep neural networks. *J. Mach. Learn. Res.*, 10:1–40, June 2009.

6 Yeshwant Kumar Muthusamy. *A Segmental Approach to Automatic Language Identification*. PhD thesis, B. Tech., Jawaharlal Nehru Technological University, Hyderabad, India, October 1993.

7 David Nadeau and Satoshi Sekine. A survey of named entity recognition and classification. *Lingvisticae Investigationes*, 30(1):3–26, 2007.

8 Seiichi Nakagawa and Allan A. Reyes. An evaluation of language identification methods based on HHMs. *Studia Phonologica*, 28:24–26, 1994.

9 John C. Schmitt. Trigram-based method of language identification. US Patent Number 5.062.143, February 1990.

10 Bruno M. Schulze. Automatic language identification using both n-grams and word information. US Patent Number 6.167.369, December 1998.

11 Jörg Tiedemann and Nikola Ljubešić. Efficient discrimination between closely related languages. In *Proceedings of COLING 2012*, pages 2619–2634, Mumbai, India, December 2012. The COLING 2012 Organizing Committee.