Language Identification with Deep Learning

Ahmet Tarık KAYA
Computer Engineering
Hacettepe University
Çankaya, Ankara
ahmet.tarik.kaya@gmail.com

Kaan Mersin
Computer Engineering
Hacettepe University
Çankaya, Ankara
kaanmersin07@gmail.com

***Abstract*—Natural Language Processing is a very popular topic in Machine Learning and it has many aspects. These aspects are really different from each other because many of them are only focused in a specific language. Every language written or spoken comes with its own rules and structure. We decided to work on Language Identifacation because we wanted to experience the work that we would need to work without thinking a specific structure of a sentence. This way we believe that we will learn the main logic in Natural Language Processing. In this paper, we will explain the model that we developed in this purpose.**

1. Introduction

Language Identification is one of the first tasks of Natural Language Processing according to many different papers. Nowadays most of the effort of the scientist that are working on this topic is focused on methods that has Neural Networks and Deep Learning approaches. We are focusing on the same topic.

While we were searching for a model to take as baseline we learned different approches. These approches thought as that, there are several steps in creating a model and training it. In this paper we are going to explain the information that we learn from similar papers and the main approach that we used while we develop a model.

2. Related Work

Language Identification is a popular topic that computer scientists have been working on for many years. Because Language Identification is a old topic, many of the models that have been developed on the topic doesn't work with a Neural Network approach. Our goal is developing a model that has a Neural Network, so we focused on the papers that has same approach.

We mainly focused on three different papers. These papers are focusing on different problems that data scientists faced while dealing with Language Identification.

The first paper that we studied was a paper with title "Language Identification: a Neural Network Approach" [1]. This model is using a dataset that contains 25 different languages. This is a basic Neural Network model. With this approach, model was able to work with a precision of 95% and only failed in identification of Portuguese language. Model uses the dataset in two steps; using training set for alphabet identification and creating a trigram for the languages itself. For alphabet identification, model creates a trigram and then merges it with trigram that contains languages. For the merge operation, models firstly merge these two trigrams for each language and then does the same operation for the entire dataset. They used forward propagation in output layer and gradient descent as the models cost function.

The second paper that we studied was a paper with title "Text Language Identification Using Attention-Based Recurrent Neural Networks" [2]. This model is using Attention-Based Recurrent Neural Networks(ARNN). Other model are using Neural Networks and this models developers are saying that basic Neural Networks requires large texts as input to make a good language identification but this will slow down the training process. ARNN is a solution for this problem. The model starts with Bidirectional Long Short-Term Memory Neural Network (BiLSTM). BiLSTM has two hidden states and when a hidden state is calculating, the Neural Networks is using both previous and next hidden state. This process is one of the main solutions for short texts in training dataset. In the identification process, ARNN uses softmax activation function for detecting the characters and relations between characters with the highest effect in the language identification and after returns vector of probability as an output for each texts. The language with the highest probability will be taged as the language of that text.

The third paper that we studied was a paper with title "Language Identification from Text Documents" [3].This model achieved 95.12% accuracy. In this model, they tried three different approaches and find the best qualities of each approaches. First approach is Multinomial Naive Bayes(MNN). This approach is working good in the identification between two non-similar languages. They experimented MNN with word n-grams and character n-grams. But this approach is not useful in this topic because MNN is using n-grams. In different languages same n in n-grams are not working sufficiently. Some n's are too small for the language and some n's are too large. As a result of only using MNN, character and word n-grams works in nearly same performance. Second approach is using Logistic Regression while the MNN model is working. With Logistic Regression character level n-gram's performance increased and it become a better method then word n-grams. Last approach was Recurrent neural Network(RNN). As I mentioned before, MNN is a good method when the dataset contains non-similar languages. So to solve this problem, they used RNN. Like the ARNN in the previous paper, RNN is a cyclic neural network between the hidden states. So as a result, they used single hidden layer RNN with gated recurrent units in their model.

3. Model

We created a model that consists of 4 main parts in order to solve our problem. To be more precise, these parts are a vectorizer, an embedding layer, a BiLSTM(Bidirectional Long-Short Term Memory) layer and a dense layer. Adam was our choice of optimizer for training the model since we lacked computation power and Adam is faster and easier to optimize compared to SGD. We used mini-batch gradient descent on training process.

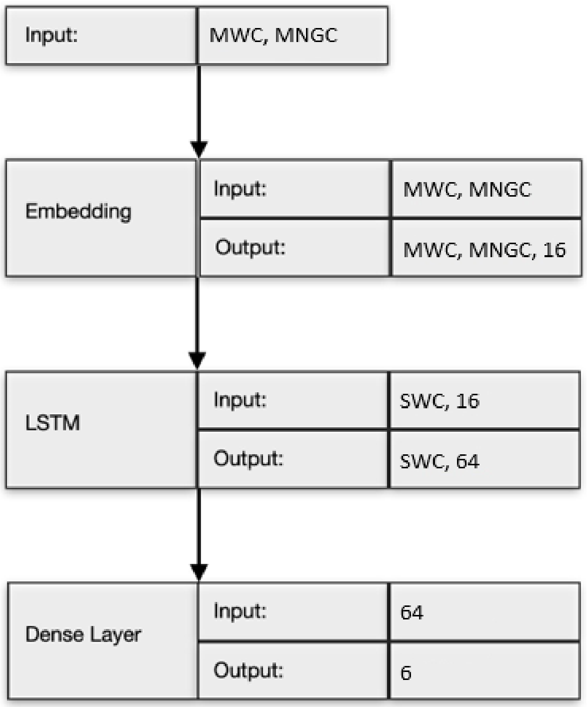


Figure 1. Architecture of our model, MWC: Max word count vectorizer can vectorize, MNGC: Max n-gram count vectorizer can vectorize, SWC: Word count in the input sentence

Here is how our model is trained:

- Fit all sentences in our dataset and train the vectorizer.
- Create vector for input sentence using trained vectorizer. Vectorizer creates character n-grams vectors for every word and concat those vectors.
- Feed output vector to embedding layer in order to extract embeddings for words.
- Feed word embeddings to the BiLSTM layer to evaluate them. Output will contain features evaluated for every word. Average of features between words are calculated.
- That average vector is feeded to dense layer that has 6 output nodes. Every node represents a different language. This layer uses softmax as activation function.
- The node with highest probability will be taken as prediction of our model and a loss is calculated.
- Model is trained by backpropagation using the calculated loss.

3.1. Vectorizer

Purpose of this step is to create a vectors for our string inputs since we can't feed a string to the neural network. This vectorizer creates n-grams for every sentence, represents every n-gram with an ID. So every word is a vector consists of n ID's. These vectors are a part of a higher level vector which represents sentences.

Since the model represents n-grams with IDs, first those IDs has to be learned. In order to do that, we trained our vectorizer with our dataset first. As for this part, vectorizer creates n-grams from sentences. If an n-gram has never been occured before, it takes the first positive integer that isn't assigned to another n-gram.

This vectorizer returns a 3 dimensional output. Elements of first dimension represents a input text. Second dimension represents words and the third dimension represents n-grams. Size of second dimension and third dimension is decided on training process. Second dimension's length is the word count of the text that contains the most words and third dimension's length is the word that contains the most n-grams occured in the training proccess. When you try to vectorize a text with 3 words, only first 3 elements of second dimension will be filled. The other elements with filled with 0's.

The most powerful part of our vectorizer is the fact that it can work with multiple n values for n-grams. You can train and use it to vectorize several n-grams. For example, if you want to work with both bigrams and trigrams, model will give unique ID's to all ngrams and trigrams occured and add all bigram and trigrams to the n-gram vector when vectorizing a word.

3.2. Embedding Layer

In Natural Language Processing, embeddings has an important role. Most language models first gives a unique ID to all words and completes further operations based on these ID's. The weakness of this approach is that it fails to use similarities between different words. Embeddings are useful for that matter. It takes extracts features from the words and processes them based on these features. Since similiar words will have similiar features, similarities between words will be part of evaluations.

In this project, we're using embedding layer to extract features from n-grams. It takes n-grams of the vectorized texts and extracts features from n-grams. Average of every n-gram features are calculated in order to represent the embedding of the words. These embeddings are feeded to the LSTM Layer.

3.3. LSTM Layer

LSTM stands for Long Short-Term Memory. It is a modified and more advanced version of a standard RNN. RNN applications have a big problem called vanishing gradients. This problem occurs when values of a gradient is either too low which results the model to learn too slowly, sometimes even stops its learning process.

LSTMs contains 3 different gates which are input gate, forget gate and output gate [6]. These gates are sigmoid neural network layers followed by point wise multiplication operators. Output gates calculates the output from the memory. Forget gate decides which information from the previous states should be discarded and lastly input gate discovers which values should be added to the cell state. Since LSTMs selects which of the information will be used on the model instead of using all the values, vanishing gradient is prevented [5] and it helps the model when processing longer sequences.

Bidirectional LSTMs analyzes a serial data in both directions, from end to the beginning and from beginning to the end. In this project, we use Bidirectional LSTM to analyze sentences by feeding it the word embeddings as our serial data. Some features are extracting from these analysis for every word.

3.4. Dense Layer

Our LSTM returns a number of features for every word in a sentence to be fed to dense layer. This means there will be different number of features extracted for different sentences. However dense layers has a fixed input size. There is two possible solutions to this problem. First one is feeding the dense layer with a vector that has the word count, max possible word counts our vectorizer can vectorize. In this case, most part of our input vector will be irrelevant for shorter sentences.

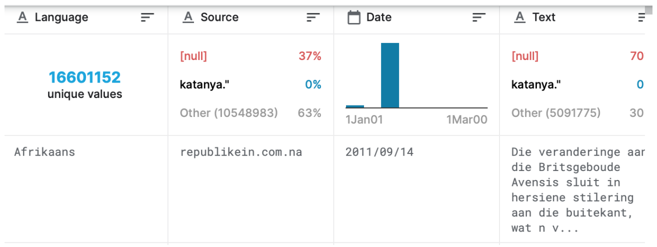
In order to solve this problem, we take average of word features and and feed it to the dense layer. Output layer has 6 neurons that represent our language. Softmax is applied to the values in output layer. Results are probabilities of every language to be predicted for that sentence.

4. Experiments Results

4.1. Dataset

As we stated in the Proposal Report, we are using Old Newspapers Dataset [4]. This dataset is a created by using HC Corpora. HC Corpora contains many texts of different languages from newspapers, social media posts and blog pages. Old Newspapers Dataset is a cleaned version of HC Corpora which only contains data from newspapers.

Our dataset is in ".tsv" format. This format stores data in tables. "old-newspapers.tsv" contains a table with four columns. These columns are "Language", "Source", "Date" and "Text". In our model we are not using "Source" and "Date" data. They are irrelevant to our models purpose. We are organizing our data by "Language" columns and using the data in the "Text" column as a training and testing date.



Language	Source	Date	Text
16601152 unique values	[null] 37% katanya." 0% Other (10548983) 63%	1Jan01 1Mar00	[null] 70 katanya." 0 Other (5091775) 30
Afrikaans	republikein.com.na	2011/09/14	Die veranderinge aar die Britsgeboude Avensis sluit in hersiene stilering aan die buitekant, wat n v...

Figure 2. A example row of the Old Newspapers Dataset Table

Old Newspapers Dataset contains 16,806,041 sentences/paragraphs in 67 different languages. There are different types of alphabets in these dataset. These many languages is making it hard to develop a model in language identification. Therefore we decided that, in the first steps of development we are only going to use 6 languages. These languages are: "English", "Spanish", "Italian", "French ", "Portuguese" and "German".

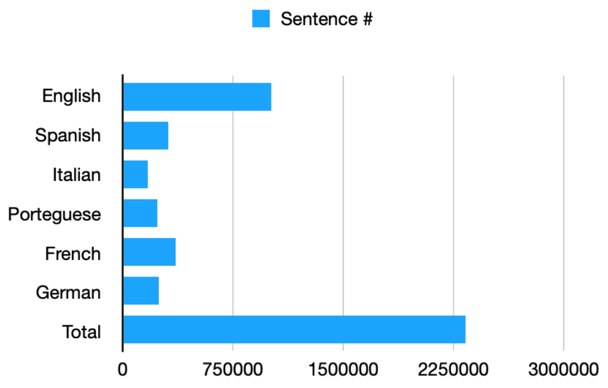


Figure 3. Language Distribution of Filtered Dataset

After we filter our dataset with the languages that we stated before by uploading our data to a SQL database, to total number of sentences reduced to 2329178. 1010242 of them are in English, 309918 of them are in Spanish, 358001 of them are in French, 236644 of them are in Porteguese, 169630 of them are in Italian and 244743 of them are in German. It didn't suprise us that the nearly half of sentences are in English. Because in our six languages, English is the most used one by biggest population.

4.2. Evaluation Metrics

In our model we are working on a dataset that contains 8 languages and when we were determining these languages we tried to selected challenging languages. The languages that we selected contains languages that has similiar words and structures. Because with these kind of languages we thought that we can see the evoluation of our model. We analyse our model when we see it mis-identifying these similiar languages to each other. These is a goal of our model too; to identifiying languages truely when similiar languages are in the model.

Like every other Deep Learning models, accuracy and loss values are the best way to see wheter our model is working proparly or not. So while we were developing our model with different hypermeters, we always compare these values. So our goal was to develop a model that has low loss value and high accuracy.

Language Identification is a consept that is popular in Deep Learning but it is hard to compare two models accuracies and loss values. Because the most important part is the dataset that model was trained with. We searched for paper to compare our model with but we couldn't find a model that we can compare results. Because we created our own language data and we were not able to find a model that uses exact 6 languages as we are.

We mentioned a baseline model that we used to develop our model. That model was using 40 languages but we modified that model a bit to be able to compare results with our model. That model was working with a 95.12% accuracy in our dataset. Our main goal is to get better results than 95.12% accuracy.

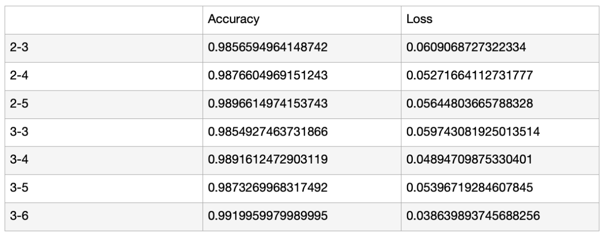
4.3. Hyperparameters

As we explained in the "Model" part of our report, we used a vectorizer of each character. The model that we analyse in our Progress Report was using a pre-trained vectorizer class for every language that it was identifying. That pre-trained vectorizer contained 2-4 vector for each character. "2-4" means that while vectorizing each character, it was using 2-3-4'grams. We didn't want to use that pre-trained vectorizer because, we wanted to train our model from scratch and that model was identifying 40 languages and we are only working with 8. So we created our own vectorizer.

We tried different range of ngrams and runned that vectorizers with a basic 2 epoch, 8 batch size and 0.01 learning rate of our model to try the best fit. The losses and accuracy that we got from this tests are in the table below.

We made more then 15 tests in the range of ngrams. In this table we only showed 7 of them to be able to talk on them. In the papers that we red, we saw that in the matter of Language Identifacition trigrams are the best fit. But even trigram is not enough by itself. Because of that information, we tried to add trigram in the every test ranges that we used.

As it shown in the table, while the range increases the accuracy increases thill 6-grams. When the ranges got bigger then 3-6, the accuracy stops to change. Because there are so little of possibilties that our model learns from 7 or more grams. So we selected 3-6 as a best fit for our model to use and we didn't want to fill our storage with redundant data.



	Accuracy	Loss
2-3	0.9856594964148742	0.0609068727322334
2-4	0.9876604969151243	0.05271664112731777
2-5	0.9896614974153743	0.05644803665788328
3-3	0.9854927463731866	0.059743081925013514
3-4	0.9891612472903119	0.04894709875330401
3-5	0.9873269968317492	0.05396719284607845
3-6	0.9919959979989995	0.038639893745688256

Figure 4. Ngrams Table in 2 Epoch, 8 Batch Size and 0.01 Learning Rate

This ngram choose was more of a process that we done before the train. The hypermeters that we test in the training process are "Epoch", "Batch Size" and "Learning Rate". We will start by showing the data of "Epoch".

Choosing the epoch in Deep Learning is so critical because noone wants their model to memorize data instead of learning it. We test our epoch with 32 batch size and 0.01 learning rate. The tables for the Loss and Accuracy that we got from this epoch are in the graphs below.

As it is shown in the graphs, after epoch 8 validation accuracy gots lower then training accuracy. It means that our model starts to memorize the training data. Because validation accuracy starts to decrease and validation loss starts to increase. So we choose 8 epoch as the best fit for our data and model.

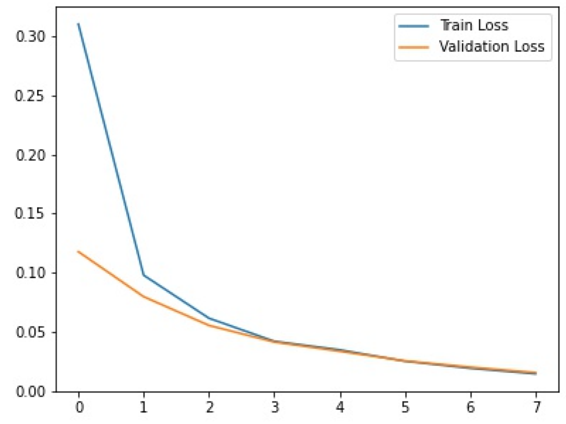


Figure 5. Epoch table for Loss while Batch Size is 32 and Learning Rate is 0.01

After choosing the right epoch, we train our model with different "Batch Size" - "Learning Rate" combinations. The values that we tried for "Batch Size" are 8,16 and 32. The values that we tried for "Learning Rate" are 0.01, 0.005 and 0.001. So we trained our model in 9 different combinations.

Batch size is the number of sample that the model worked

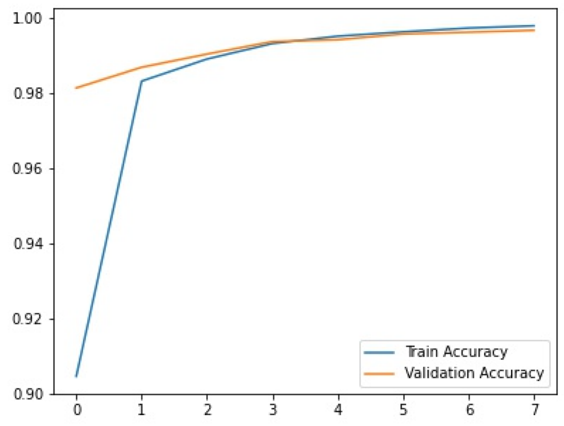
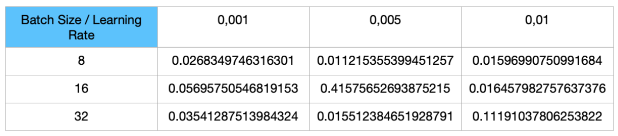
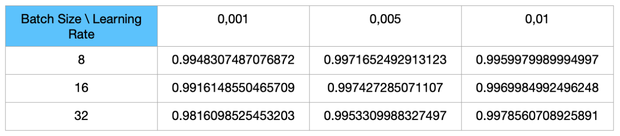


Figure 6. Epoch table for Accuracy while Batch Size is 32 and Learning Rate is 0.01



Batch Size / Learning Rate	0,001	0,005	0,01
8	0.0268349746316301	0.011215355399451257	0.01596990750991684
16	0.05695750546819153	0.41575652693875215	0.016457982757637376
32	0.03541287513984324	0.015512384651928791	0.11191037806253822

Figure 7. Loss table for 8-16-32 Batch Size and 0.01-0.005-0.001 Learning Rate in 8 Epoch



Batch Size \ Learning Rate	0,001	0,005	0,01
8	0.9948307487076872	0.9971652492913123	0.9959979989994997
16	0.9916148550465709	0.997427285071107	0.9969984992496248
32	0.9816098525453203	0.9953309988327497	0.9978560708925891

Figure 8. Accuracy table for 8-16-32 Batch Size and 0.01-0.005-0.001 Learning Rate in 8 Epoch

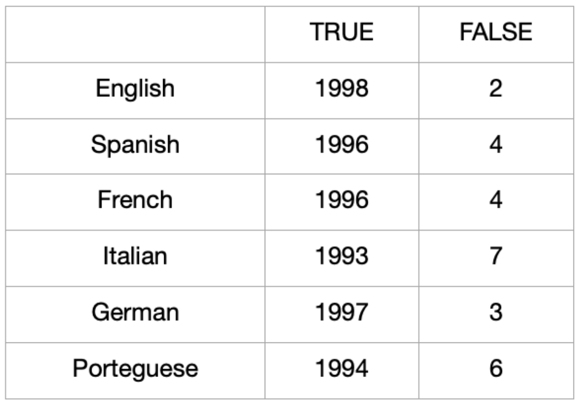
through before updating any model parameters. It is hard us to make a generalization like loss is always decreasing or accuracy is always increasing while batch size is increasing. Because after some point, when the batch size is too high then it gets hard for the model to train itself better. But when we looked in the tables that contains this data, we can see that in the best learning rate, while batch size increased to 32 the accuracy of our model increased and loss of our model decreased. So we decided that 32 batch size is the best fit for our model and dataset.

Learning rate is the number that determines the rate of a data effect on the model parameters while training process. Like batch size the increase or decrease of learning rate can increase the accuracy of a model. We tried three different values in this process and the values were real different from each other. So as it shown in the table there are significant changes in loss and accuracy with different learning rates. We can make a generalization that higher learning rate works better in our model. So we decided that 0.01 learning rate is the best fit for our model and dataset.

4.4. Results

The model that we developed was a project that was avaiable on internet with pre-trained model parameters. When we runned that baseline model with our datase we got 95.12% accuracy. After we made the alterations that we explained in this paper and we increase this accuracy to 99.82%.

This paper will have more Error Analysis in next part but as a result we can say that, in Language Identification the database is important as the model. Because even when model learns the structure of the language perfectly there will be some words that model is not familiar with and that words can be in short sentences. So that words will effect the models output in a very large way. In mis-identified sentences with our model has this problem. But it is only 26 in 6000.



	TRUE	FALSE
English	1998	2
Spanish	1996	4
French	1996	4
Italian	1993	7
German	1997	3
Porteguese	1994	6

Figure 9. True/False Label Numbers for Each Language with our Model

As we stated in our previous paper(Progress Report), we were expecting that Spanish, Italian and Porteguese will be mis-identified as each other because the sentence structures and words are really similiar to each other. This is the biggest mis-identifying problem in our model. These languages are the top 3 languages with false labels. As we are going to talk about in this report, we trained our model with different hypermeters and after many many trains we picked the perfect combination of these hypermeters according to our data and model.

As a result of all this testing, we picked our hypermeters as 8 epoch, 32 batch size and 0.01 learning rate. But to be able to add good results to this paper, we increase the size of our dataset and run our model with it for a one last time with 12 epoch. The ”Loss” value and ”Accuracy” of our model is stated below.

So our loss is so low and our accuracy is so high. These means that our model is nearly working perfectly. After many many alteration in our model we came to these point.

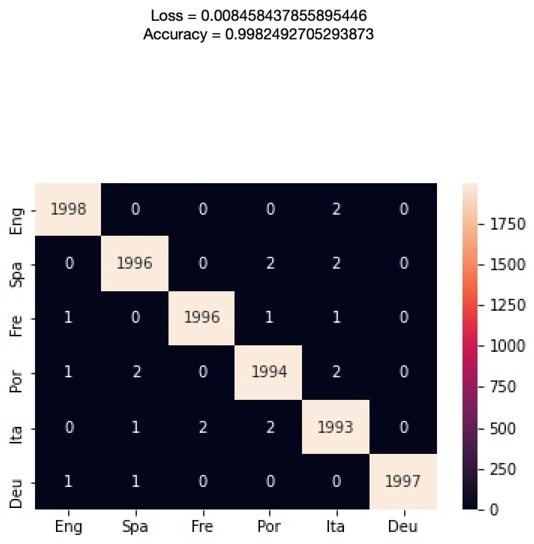


Figure 10. Confussion Matrix at of Test after Training with 12 Epoch, 32 Batch Size and 0.01 Learning Rate

Our Confussion Matrix is in the figure. We have so little errors in our model. While the testing process, we stored the wrongly taged words too. Because it will make us understant the errors in our model.

There is no language in our 6 language database that our model was able to identify perfectly. Each language has 3-5 errorly identified sentence. This number is highest in Italian with 7 sentences and lowest in English with only 2 sentences. These are the main information about our Evaluation Metrics. This report will have more informations and comments about this Confussion Matrix in Error Analysis(4.5) part.

4.5. Error analysis

Our accuracy is 99.82% which is nearly perfect. But still there are mis-identified sentence and we were only working with 6000 sentences in test. These number probably increases when the test dataset gets bigger because the reason of this mis-identified sentences is obvius. The reason is the words that our model sees for the first time.

We think that this is not a problem that can be solved by the modifacations in model. Model is not only working by the words itself, it is working with the characters in the words. This means that model is actually learning the word structures in a language too. But proper names are not always in a structure in a language. So our model is having problems with the propers names in a language. The errors that occur in the model are because of these proper names. When the sentence is not long and contains a proper name that our model can 't identify or identify wrongly, then this results with a false label.

There can be two different solutions for these problem and both of them are about the data that we can use and one of them will require additional training process. First one is basic, only increasing the size of our dataset. This is not a problem accutually because our dataset was originally bigger then we used but we were not able to perform with a big data set because of our computers. Second one is that we will have a additional dataset for each language that contains every proper name that is in that language. After that we will only add this words to our vectorizers that our model will know that these words are in that specific languages vocabalary. But the biggest problem with this is that this set is very very large and we don 't even know wheter there is a dataset like this or not. So such a big dataset will increase our vectorizer and this will efect the accuracy of our model. So this will cause a small problem to effect the rest of the model. But we know that the only solution for this unknown proper words is to teach these words to our model somehow.

5. Conclusions

In this paper we explained the dataset that we are using, the model that we created and our train process in this model. We used different hypermeters, we used different approaches then our baseline model and we developed a model that can make language identification in a good rate.

The baseline model that we were using was a model that was developed for identification of 40 languages. It was a model that using its own pre-trained vectorizer and it was using 2-4 ngrams. This was our first change in the model. We tried different ngrams for vectorize and we choose the best fit one for our purpose and dataset. We decided that 3-6(3gram, 4gram, 5gram and 6gram) is the best fit for our dataset and model.

After that, while we are training our model with the dataset we created, we used different hypermeter combinations and compare the outputs of those hypermeters with each other. The hypermeters that we used was ”Epoch”, ”Batch Size” and ”Learning Rate”. After this comparization process we decided that 8 epoch, 32 Batch Size and 0.01 Learning Rate is the best combination for our dataset and model.

In the end, we run our model with 12 epochs because we used a bigger dataset then the previous prediction steps and got a accuracy of 99.84%, which is saying that our model is nearly perfectly predicting the language. s

References

[1] https://pdfs.semanticscholar.org/17b2/c9ae27d2ef1b6b901a0afd5aa8649f7795bf.pdf

[2] https://www.researchgate.net/publication/333392357_Text_Language_Identification_Using_Attention-Based_Recurrent_Neural_Networks

[3] http://cs229.stanford.edu/proj2015/324_report.pdf

[4] https://www.kaggle.com/alvations/old-newspapers

[5] https://medium.com/datadriveninvestor/how-do-lstm-networks-solve-the-problem-of-vanishing-gradients-a6784971a577

[6] https://towardsdatascience.com/understanding-rnn-and-lstm-f7cdf6dfc14e