EDAN20 Language Technology
-Lab 3.

Nils Romanus

September 2022

1 CLD3 Overview

CLD3 is a model for language identification. It works by firstly creating a feature map of a piece of text consisting of embeddings of character n-grams with their relative occurance in the text. This feature map is subsequently passed through a relatively simple network with one hidden layer, using the ReLu activation function, and a softmax layer.

2 Understanding the X Matrix

An example of such a feature map, for the example sentences in table 1, is displayed in the matrix 1. A similar matrix constitutes the feature map that is passed to the model, however, matrix 1 shows the counts of the character uni- and bigrams `#a, #b, #n, #an, #ba, #na` instead of their relative occurrence.

```
1276    eng     Let’s try something.
1277    eng     I have to go to sleep.
1280    eng     Today is June 18th and it is Muiriel’s birthday!
...
1115    fra     Lorsqu’il a demandé qui avait cassé la fenêtre,
tous les garçons ont pris un air innocent.
1279    fra     Je ne supporte pas ce type.
1441    fra     Pour une fois dans ma vie je fais un bon geste... Et ça
ne sert à rien.
...
337413  swe     Vi trodde att det var ett flygande tefat.
341910  swe     Detta är huset jag bodde i när jag var barn.
341938  swe     Vi hade roligt på stranden igår.
...
```

Table 1: Example Sentences

$$\mathbf{X} = \begin{bmatrix} 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 3 & 1 & 2 & 1 & 0 & 0 \\ 8 & 0 & 8 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 \\ 3 & 1 & 5 & 1 & 0 & 0 \\ 3 & 0 & 1 & 0 & 0 & 0 \\ 4 & 2 & 2 & 0 & 0 & 0 \\ 2 & 0 & 1 & 1 & 0 & 0 \end{bmatrix} ; \mathbf{y} = \begin{bmatrix} \text{eng} \\ \text{eng} \\ \text{eng} \\ \text{fra} \\ \text{fra} \\ \text{fra} \\ \text{swe} \\ \text{swe} \\ \text{swe} \end{bmatrix} \tag{1}$$

3 Extracting Features

To create the character n-gram features that are later to be passed to one must first create a program that can extract these from a piece of text. Luckily this was done in lab 2. One can therefore modify the programs from lab 2 to count the occurrences of *character* n-grams in a text instead of *word* n-grams by simply letting each token be a character. The frequency of each n-gram can subsequently be divided by the total number of n-grams in the text to generate a mapping between n-gram and its relative occurance in the text.

4 Building X and y

The X feature map can subsequently be created by using the n-gram to relative occurrence map. In order to reduce computational the features were restricted to containing only unigrams. One may use the `DictVectorizer` from the `sklearn` api to transform the n-gram to relative occurrence map into a matrix. In order to transform the response y one may simply give each language an index, for example 0 corresponding to swedish, and store these in a map. Using this mapping y can be transformed into a numerical vector.

5 Building- and Fitting the Model

The network described in section 1 can be implemented using the `sklearn` api. The main difference between the architectures is that our model only uses unigrams instead bigrams and trigrams as the standard CLD3 does. Another difference is that our model does not make use of an embedding layer where embeddings are averaged and concatenated. As shown in figure 1 one may view our network as the last part of CLD3. Instead of using an embedding layer we manually engineer our X feature map and input it to the last part of the CLD3 network.

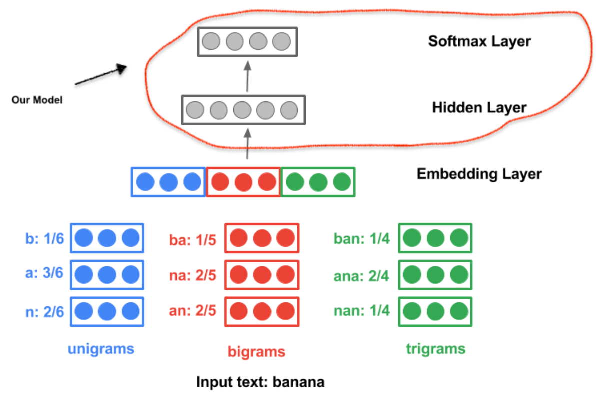

Figure 1: Our model

The model is subsequently trained by doing a standard random-shuffle split and subsequently validated on the validation set. Using our trained and tested model one may use it for inference on test sentences given the test sentences are transformed using the steps listed in section 4.

6 Building- and Fitting the model with Keras

Using the `Keras` API one can create a copy of the `sklearn` model. The main difference between `Keras` and `sklearn` implementation of an MLP is ease-of-use and customizability. The `sklearn` MLP can be created using one line of code whilst the `Keras` MLP requires a bit more boilerplate code. On the other hand `sklearn` does not have the same customizability options. One can for example not add regularization components such as dropout- or batch-norm layers when working with `sklearn`. Furthermore `sklearn` does not have GPU-support whilst software built on the `Keras` api can run on a CUDA engine.