Named entity recognition with Bidirectional LTSM for Hungarian language

-

Entitás kinyerés magyar nyelvű szövegekből kétirányú LSTM-mel

Zalán Béla Bokor
Budapest University of Technology and Economics
Budapest, Hungary

Bálint Mészáros
Budapest University of Technology and Economics
Budapest, Hungary

Balázs Scheier
Budapest University of Technology and Economics
Budapest, Hungary

***Abstract*— Nowadays we have a lot of data accessible, but a sizeable amount of them is in text format. With deep learning we can only have numerical inputs and outputs, and words have different meaning in different context. For these reasons it's obvious we categorize the task processing natural languages together in the the group NLP. One of the big tasks in NLP is Named entity recognition. It's useful for identifying the key elements out of the sentences. So, for many NLP related problems NER is one of the tools to process the data better. Because of this it's important to have a functioning NER model at hand when doing other NLP tasks.**

Manapság nagyon sok adat áll rendelkezésünkre, viszont ezeknek nagyrésze szöveges formátum. A probléma ott adódik, hogy deeplearning modellek csak numerikus be- és kimeneteket tud kezelni. Ahhoz, hogy ezeket az adatokat kezelni tudjuk pár átalakításon kell átesni. Mivel ezek az átalakítások hasonlóak sok féle feladatnál, ezért ezeket mind az NLP-be (természetes nyelvfeldolgozás) soroljuk. Az egyik nagy NLP feladat az entitás kinyerés. Ez arra hasznos, hogy a kulcsszavakat kinyerjünk a szövegből. Ezért nyilvánvaló, hogy a NER egy elég hasznos eszköz a komolyabb NLP feladatok során, ezért nagyon fontos, hogy legyen egy jól működő megoldásunk rá.

I. Introduction

Named entity recognition (NER) is the task where you have an input sequence of words and decide for every word whether it's a named entity or not. For example, you have the following sentence: John is going to the United Kingdom for holiday. The named entities are "John" and the "United Kingdom". Other difficulties arise when we try to do NER in different languages. In this paper we introduce our solution for Hungarian. Because the Hungarian language uses inflection determining NE-s are way harder. For example, "Budapest" is NE but "budapesti" is not. Because of this we can't use the lemmas of our words, this leads to higher dictionaries.

II. System design

We are using an embedding layer connected to two bidirectional LSTM layer for last we are using a fully connected dense layer. This network while being terribly slow will work quite nicely.

III. Implementation

A. Gathering the data, and preprocessing

For our NER project we use the database HuWiki 1. It's a database consisting of sentences being tagged by ConLL U. We downloaded and extracted the dataset. The importation was handled by pandas. The data consisted of multiple tags. We only used the word and the entity tags.

With our data ready we had to transform it into processable form. First, we reduced the size of our data. Originally, we had 2.3 million rows. We found that handling that much data was unnecessary, so we only used the first 200000 rows. After this we filtered out the null lines.

After all this we converted our output tags into numbers using the following dictionary:

Entity tag	Meaning	Integer Id
B-LOC	Location beginning	0
B-MISC	Miscellaneous beginning	1
B-ORG	Organization beginning	2
B-PER	Person beginning	3
I-LOC	Location beginning	4
I-MISC	Miscellaneous inside	5
I-ORG	Organization inside	6
I-PER	Person inside	7
O	Nonentity	8
PAD	Padding	9
BOS	Beginning of sentence	10
EOS	End of sentence	11

We will be using additional tags, so we added them into the dictionary.

Then we turned our attention to the input lines. First, we created sentences out of the lines. We considered the following characters sentence ending: ".", "?", "!", ":". After the sentences were done, we also removed the punctuation.

To reduce our possible input values, we tokenized our input words. For this we used "SZTAKI-HLT/hubert-base-cc" with AutoTokenizer.

We wanted to use as little padding as necessary, so we decided to use bigrams instead of sentences. Because of this choice we didn't have to pad every sentence and handling our outputs because also way easier since every bigram belonged to one word. To have the ngrams working we needed a beginning of sentence (BOS) tag and an end of sentence (EOS) tag. After these got into their places, we padded the words and outputs, and created the bigrams. To have an easier time with our input data we flattened the data.

For the output data we used one-hot encoding. For the Input data we used bag of words (BoW) to convert our strings into numerical input.

B. Training

After these conversions were done our data was ready so we split them into train validation and test datasets.

We optimized for accuracy even though in our first epoch our accuracy was very high. While our validation accuracy had fluctuating values our loss kept getting lower and lower. We used earlistopping with 5 patience.

For hyperparameter optimizing we used hyperas. But it wasn't that comprehensive because our training took a lot of time mainly because of our still massive input size was rough for the LSTM network which can't work parallel.

C. Evaluation

We tried to use a custom loss function, which multiplied each named entity dimension of the vector of the difference of the predicted label and the true label by a constant. But we didn't achieve a better result, so in the end we didn't change loss function categorical crossentropy. For hyperparameter optimization we used at most 10 evaluation runs, and the five best results are in the table at the bottom of the page.

After the training was done our result were the following:

Train accuracy	0.9973
Validation accuracy	0.9861
Test accuracy	0.9874

Although those numbers are high it's mainly because our data set is flawed. We have much more words with "O" tags than any other.

D. Testing

We tested our neural network with sentences out of training dataset. The network identified correctly named entities, but it wasn't sure about the category of the entity. So for example, "Ferenc" is a "B-PER" entity, but the network decided it is a "B-LOC". We've tested the modell on a couple of sentences and our accuracy was: 0.7857.

IV. FUTURE PLANS

If we'd want to improve on our solution, we'd use a transformer-based model since it would highly increase our training speed while also having better performance. Another upgrade would be to replace the BoW with an embedding as it would make using words out of the dictionary better.

With the increased speed transformer modells would provide we could train on bigger data, leading to better accuracy.

We should also try to increase the accuracy on the false negative entity predictions.

Embed-ding layer size	**First LSTM layer size**	**Second LSTM layer size**	**Optimizer function**	**Number of batches**	**Best validation accuracy**
64	256	256	Adam	256	0.9905
512	256	512	RMSprop	256	0.9883
512	512	64	Adam	64	0.9881
512	512	512	RMSprop	256	0.9876
128	128	128	Adam	128	0.9875