Hungarian named entity recognition using huBERT [1] - Magyar névelem felismerés huBERT [1] használatával

Élő Henrik Rudolf, Zahorán Marcell

ABSTRACT This semester, we have dived deep into the vast and varied field of Natural Language Processing, particularly into the sparsely explored area of Hungarian Named Entity Recognition (NER). In this paper we will share in depth the experiences we have collected and the results we have reached along the way creating this Hungarian NER model. We trained on the NYTK-NerKor [2] Hungarian labeled NER data. In our research we will talk about how we fine-tuned the huBERT [1], the Hungarian NER model that is based on BERT[3]. We reached a 92% accuracy, but our model didn't manage to differentiate between the classes that well, but it opened up a lot of opportunities to improve our work in the future.

KIVONAT A félév során belevettük magunkat a természetes nyelv feldolgozás hatalmas és sokszínű világába, azon belül is a magyar névelem felismerés (NER) területébe. Ebben a beszámolóban részletesen kifejtjük a magyar névelem felismerő modell elkészítése során felszedett tapasztalatokat és a modell által elért eredményeket. A modellünket a NYTK-NerKor [2] felcímkézett magyar névelem felismerő adathalmazon tanítottuk. A kutatásunkban kitérünk arra, hogy hogyan sikerült tanulásra bírni a huBERT-et[1], a magyar BERT modellt, és arra is hogyan egészítettük ki saját rétegekkel azt. A tanítás végére sikerült 92%-os pontosságot elérni, de ez nem ad okot arra, hogy hátradőlhessünk, mert a modellünk nem képes hatékonyan megkülönböztetni az egyes osztályokat. Ennek ellenére az idei munkánk utat nyitott jövőbeli fejlesztő szándékú projektek számára.

I. INTRODUCTION

This semester, within the university subject "Deep Learning in Practice with Python and LUA", our task for homework was to create a Hungarian NER model.
We used the labeled Hungarian NER data from NYTK-NerKor [2].
We will discuss how we prepared and transformed the data to make it easier to use it with the NER models. We explored the NER models that are currently available and decided to use a Hungarian BERT [3] based model, huBERT [1]. We will go into detail how we managed to get the huBERT [1] model to learn and how we fine-tuned it.

II. PREVIOUS WORK

There are many deep learning models that have been used for named entity recognition (NER) tasks before the development of huBERT [3]. Some of the most popular ones include recurrent neural networks [10] (RNNs), convolutional neural networks [9] (CNNs),and long short-term memory (LSTM) networks [7].

RNNs are a type of neural network that are well-suited for NER tasks because they can process sequential data, such as natural language text. RNNs have the ability to remember information from previous inputs, which allows them to capture contextual information about the words in a sentence. This makes them effective at identifying named entities, such as people, places, and organizations within a text.

CNNs are another type of neural network that have been used for NER tasks. Unlike RNNs, which are designed to process sequential data, CNNs are designed to process grid-like data, such as images. However, they can also be applied to text data by representing each word as a grid of numerical values, called a word embedding. This allows CNNs to capture information about the local relationships between words in a sentence, which can be useful for identifying named entities.

LSTM networks are a type of RNN that have been specifically designed to better capture long-term dependencies in sequential data. They do this by using gating mechanisms, which allow them to selectively forget or retain information from previous inputs. This makes LSTMs particularly effective at NER tasks, as they are able to take into account the context of a word within a sentence in order to accurately identify named entities.

III. huBERT

It is difficult to say whether huBERT [1] is definitively better than other deep learning models for named entity recognition tasks, as this can depend on a number of factors, including the specific dataset and evaluation metrics used. However, huBERT [1] is a recently developed model that incorporates novel techniques, such as transformers and self-supervised learning, which may make it more effective than some older models in certain situations.

Transformers are a type of neural network architecture that has been shown to be highly effective for natural language processing tasks. They are particularly well-suited for NER tasks because they are able to capture global contextual information about a sentence, which allows them to accurately identify named entities. huBERT uses transformers as part of its architecture, which may make it more effective at NER than models that do not use this type of architecture.

Another reason why huBERT may be better than some older models is that it uses self-supervised learning. This means that it is trained on large amounts of unannotated text data, which allows it to learn about the structure and patterns of natural language on its own. This can be beneficial for NER tasks, as it allows the model to learn from a much larger amount of data than would be possible with supervised learning alone.

Overall, huBERT may be better than some older deep learning models for NER tasks because it incorporates novel techniques, such as transformers and self-supervised learning, which may make it more effective at identifying named entities. However, the exact extent to which it is better than other models will depend on the specific situation in which it is used.

IV. ABOUT THE DATA

We got our labeled data from the Research Group of Language Technology, NYTK. They have a large, labeled NER database, called NYTK-NerKor [2], in CoNLL-U Plus Format [1]. The data contains words from various genres. It has a fiction genre containing text from MEK, Project Gutenberg and subtitles from OpenSubtitles. The legal genre contains words from varied EU legal documents. The news genre consists of data from the Press Release Database of the European Commission, Global Voices and NewsCrawl Corpus. The web genre is a selection from the Hungarian Webcorpus 2.0[2]. Lastly the Wikipedia genre is texts from Hungarian Wikipedia pages.

[1]CoNLL-U Plus Format :https://universaldependencies.org/ext-format.html
[2]Hungarian Webcorpus 2.0: https://hlt.bme.hu/en/resources/webcorpus2

V. PREPARING THE DATA

The data we used was in CoNLL-U Plus Format[1]. The genres were divided into multiple *.conllup* files. These files contained 6 pieces of information about each word. The first information was the FORM, which was the actual word. LEMMA contained the lemma of the token, UPOS was the part-of-speech tag, XPOS was the language specific part-of-speech tag, FEATS contained the list of morphological features, and finally in CONLL:NER were the NER tags stored.

The tags are LOC for locations, PER for persons, ORG for organizations, MISC for miscellaneous names and O for others. Each tag except the O tag has a prefix which tells us whether it is the first part of a named entity or some other part of it. The first part of each tag is indicated with a *"B-"* prefix. (B-PER, B-LOC, B-ORG, B-MISC). Any other parts are preceded with the *"I-"* prefix. (I-PER, I-LOC, I-ORG, I-MISC). That gives us nine tags.

We came to the conclusion that we will only need the FORM and CONLL:NER for each word, but we put the other information into the final data frame to have them just in case.

Fortunately, the division of data into train, devel and test datasets was already done in the used dataset, so splitting the data required no additional work from us.

The *.conllup* format was not convenient for deep learning purposes as there were few CoNLL-U reader libraries in python let alone CoNLL-U Plus readers. So we had to implement our own CoNLL-U Plus format reader. We come about doing that by creating a *.conllup* file reader that read each line of the file and appended it to the structurally similar pandas data frame. We converted the *.conllup* files in each genre into a single data frame and then saved it as a *.csv* file. This way the loading of data got much faster, and we were ready to start exploring out data.

VI. EXPLORING THE DATA

For the data exploration we used pandas and matplotlib.

First, we collected the word counts of each genre. We did that by loading the previously created *.csv* files and loaded them into a pandas data frame and the data frame's length was the word count of each genre. We also wanted to know how many unique words a genre contains. It could be easily obtained as pandas data frames have a function called *unique()* which returns the unique records of the data frame. We had to call that function on the sub data frame created by only selecting the FORM column of the original data frame, because we only wanted the unique forms and nothing else.

The result of the word count analysis can be seen in FIGURE 1, FIGURE 2 and FIGURE 3. As can be observed the number of unique words is much lower, 2-6 times lower, compared to the number of all words across all of the datasets in all of the genres. This happens because the dataset contains Hungarian stop words, but we decided to keep them in the dataset for now and remove them later if needed.

We also wanted to get a better overview about how each tag is distributed in the datasets across the genres. Getting this data was quite easy using pandas' helper functions. We have found, to no one's surprise, that most of the tags are O. Around 92% percent of the tags are tagged other, this is something we have to look out for, because later a good accuracy could mean that all of the tags are predicted as O.

Luckily, we found that the distribution of each tag in each dataset is similar, so none of the datasets are skewed, so we had no additional task with data separation.

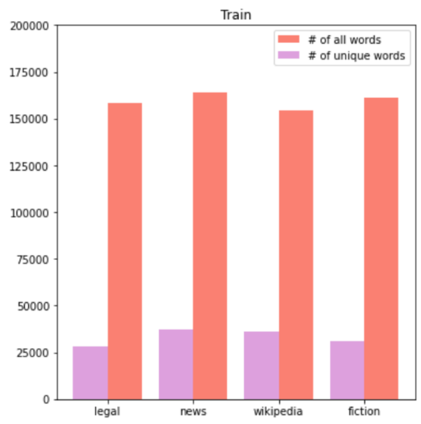

FIGURE 1. **The number of all words (red), and the number of unique words (purple) across the genres in the train dataset.**

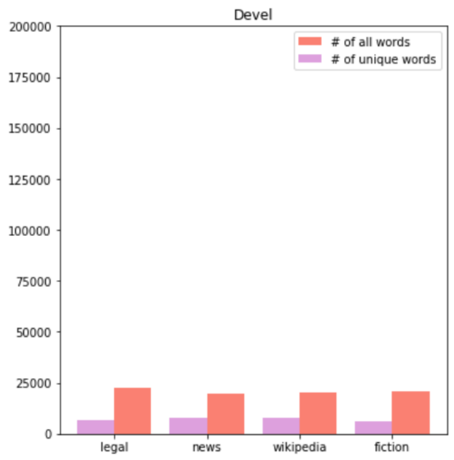

FIGURE 2. **The number of all words (red), and the number of unique words (purple) across the genres in the devel (validation) dataset.**

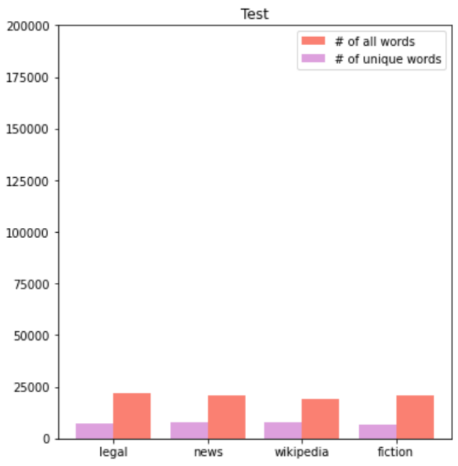

FIGURE 3. **The number of all words (red), and the number of unique words (purple) across the genres in the test dataset.**

VII. EXPLORING THE DIFFERENT MODELS

We wanted to use a state-of-the-art model, so we looked at the best performing models on the CoNLL 2003 dataset [4] on paperswithcode.com. We have found ACE [5], which finds concatenation of word embeddings for better performance. We looked at FLERT [6], which achieves high performance by capturing document-level features. And we also considered LUKE [8] that outputs a contextualized representation of tokens. All of these models were great but in the end, we went with BERT [3], because there was a model, huBERT [1] pretrained for Hungarian named entity recognition and we decided to fine-tune that model.

VIII. THE MODEL AND THE FIRST TRAINING

Getting the huBERT[1] model to learn turned out to be quite the challenge. Even though the creator of huBERT[1] uploaded his model, making it functional in PyTorch wasn't that easy. We had to read numerous pages of documentation until it was finally ready for configuration. Our model uses the pretrained huBERT [1] model that outputs a vector with 768 elements. We took ideas from the huggingface NER model[3] when adding the following layers. After the base BERT layer we added a dropout layer with a 10% dropout rate then a Linear layer that outputs a 512 element vector, followed by a ReLU activation layer. After that another Linear layer is added that has an output size equivalent to the number of labels, followed by a sigmoid activation layer.

We started training the model in PyTorch for 5 epochs, with a 128 batch size, a 10^{-4} learning rate and Stochastic

[3] AutoModelForTokenClassification: https://huggingface.co/transformers/v3.0.2/model_doc/auto.html#transformers.AutoModelForTokenClassification

Gradient Descent optimizer. We also freezed the gradients of the base BERT layer, so it won't be trained. The training went smooth, we managed to achieve a pretty high, 92% accuracy, which made us happy until we looked into these results more.

After examining the confusion matrix, seen on FIGURE 4, we have found that the model has such a high accuracy because it predicts almost everything as other (O).

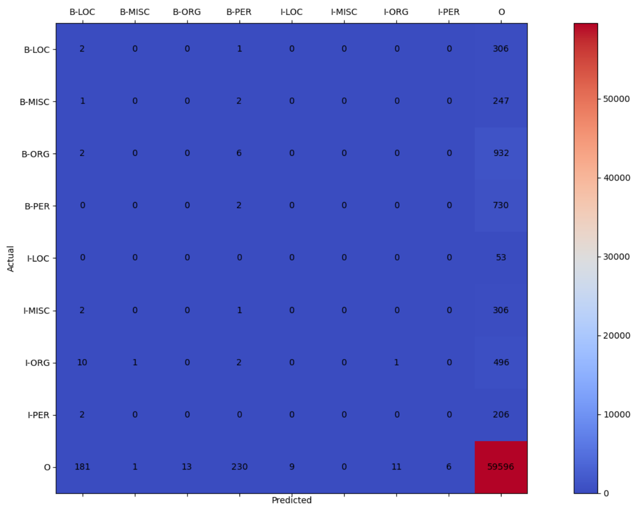

FIGURE 4. **The confusion matrix of the model after the first training.**

IX. AUGMENTING THE DATA AND THE SECOND TRAINING

After consulting with our teacher, he gave us great solutions on how to tackle an unbalanced dataset. One of these ideas was to remove the other (O) type words from all of the sentences and append that to the train dataset.

Doing that we managed to get the ratio of other tags down from 92% to 87%.

With the augmented data we started a new training with almost the same parameters, we only had to lower the batch size to 64, because we ran out of GPU compute units on colab.

Sadly, we had to stop the training after the second epoch as the accuracy did not improve and after evaluating our results it turned out our model did the same thing, predicting everything as other (O).

X. CONCLUSION AND FUTURE PLANS

Unfortunately, we ran out of time, and just when it got interesting. Even though our model didn't achieve much, we still learned a lot, and this work laid a strong foundation to follow-up projects in which we could improve our results. Personally, I would like to see whether more or different augmentation would improve our accuracy. Or would removing the stop words help us. Or experiment with the hyperparameters to maybe stumble across a combination of them that makes the predictions just right. As we can see this is just the beginning of an extensive and fascinating journey.

REFERENCES

[1] D. M. Nemeskey, Natural Language Processing Methods for Language Modeling, Eötvös Lóránd University, 2020.

[2] E. Simon és N. Vadász, „Introducing NYTK-NerKor, A Gold Standard Hungarian Named Entity Annotated Corpus," in *Text, Speech, and Dialogue - 24th International Conference, TSD 2021, Olomouc, Czech Republic, September 6-9, 2021, Proceedings*, 2021.

[3] J. Devlin, C. Ming-Wei, K. Lee és K. Toutanova, BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding, https://arxiv.org/abs/1810.04805v2: Google AI Language, 2018.

[4] F. De Meulder és E. F. T. K. Sang, „Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition," 2003.

[5] X. Wang, Y. Jiang, N. Bach, T. Wang, Z. Huang, F. Huang és K. Tu, „Automated Concatenation of Embeddings for Structured Prediction," 2021.

[6] S. Schweter és A. Akbik, „FLERT: Document-Level Features for Named Entity Recognition," 2020.

[7] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami és C. Dyer, „Neural Architectures for Named Entity Recognition," 2016.

[8] I. Yamada, . A. Asai, H. Shindo, H. Takeda és Y. Matsumoto, „LUKE: Deep Contextualized Entity Representations with Entity-aware Self-attention," 2020.

[9] J. P.C. Chiu, E.Nichols: "Named Entity Recognition with Bidirectional LSTM-CNNs", 2015

[10] Z.Yang, R. Salakhutdinov, W. W. Cohen: "Transfer Learning for Sequence Tagging with Hierarchical Recurrent Networks", 2017