# Named Entity Recognition For Macedonian Language

Ivan Krstev*, Fisnik Doko†, Sasho Gramatikov‡, Miroslav Mirchev§ and Igor Mishkovski¶
Faculty of Computer Science and Engineering, Skopje, Macedonia
Email: *Ivan.Krstev.1@students.finki.ukim.mk, †fisnikdoko@gmail.com, ‡miroslav.mirchev@finki.ukim.mk,
§sasho.gramatikov@finki.ukim.mk, ¶igor.mishkovski@finki.ukim.mk

Index Terms—NER, Entity Recognition, FLAIR, NLP, Machine Learning

Abstract—Named Entity Recognition (NER), an outstanding technique for information extraction from unstructured texts, is lately becoming the central problem in the field of Natural Language Processing (NLP). In the last few years, multiple *Python* libraries, like *SpaCy*, *NLTK* and *FLAIR*, accomplished state-of-the-art performances for this problem. As *NER* is developing into a powerful technique, its real-live applications are becoming more and more numerous: from customer-message categorization to ease of document analysis in greater corporations. In this research, we use a ML-based system with the help of the *FLAIR* library in *Python*, which has already provided optimal results for *NER* in few world-class languages (English, German, Russian, French etc.), for financial entity recognition in financial texts written in Macedonian language. For the NER task on 13 distinct labels using our dataset in Macedonian language on the proposed ML model we have obtained F1-score of around 0.75.

## I. Introduction

Today we have access to a huge number of textual documents, mostly available in an unstructured form (internet portals, social networks, web apps). These documents can be used for automating business processes which eases the overall work of one company or institution. However, working with textual data in its original form is a challenging task since it is unstructured, specific to its author, and, in some cases, ambiguous because of the existence of words with different meanings in different contextual uses, also known as polysemy.

When talking about NLP and extracting information from text, most of the time, we mean *NER* or *PoS* (Part-of-Speech) tagging of texts. Both methods treat the text as a sequence of words assigning linguistic tag (or label) to each word. To accomplish this task, the current state-of-the-art models use $Bi-LSTM$ (Bi-directional Long Short Term Memory), although other methods like the language $Transformers$ with *Attention Layers* are widely used too [1].

NER involves processing a text and identifying certain occurrences of words or expressions as belonging to particular categories of Named Entities (NE) - (A Mikheev, M Moens, C Grover - 1999) [2]. The main goal is to assign a predefined class (or label) to the words, depending on the concept they describe and they belong to. There are several approaches to build such a model, like, lexical approach, rule-based systems, ML-based systems (deep learning) and hybrid systems.

In our work, we will stick to the Deep Learning based models with the *FLAIR* library from *ZalandoResearch*. This paper covers gathering and structuring data from Macedonian on-line portals, building a corpus from it and using it for the NER labeling problem. In total, 13 labels are used, "PERSON", "PUBLICATION", "COMPANY", "NUMBER", "PRODUCT", "EVENT", "PERIOD", "LOCATION", "AMOUNT", "SECTOR", "MEDIA TYPE", "POLITICAL ENTITY", and "PERCENT".

The rest of the paper is organized as follows. In section II we explain the constituent part of the language model architectures offered by FLAIR. The word embeddings used for the Macedonian NER Model are described in section III, whereas the data description and the used approach for the training of the model is shown Section IV. In Section V we evaluate the obtained model from Section IV and we present and explain the results. Section VI concludes this work.

## II. Language Model Architectures (FLAIR)

Using Machine learning to understand natural languages is becoming excessively used tool in business process automation. Training the neural networks with the minimum amount of data in shortest period of time is a crucial task for these tools. A tool that achieves this goal is Zalando Research's *FLAIR* [3] which currently offers state-of-art solutions for multiple classical *NLP* tasks, such as, *NER* and *PoS* (Part-of-speech) tagging and text classification with pre-built language models for more than 20 languages.

*FLAIR* is *PyTorch*-based framework which is pretty intuitive for use in *Python* for training your models using the embeddings they offer. It uses

an approach for modeling natural languages with deep, recurrent neural networks which helps it learn powerful and contextual information for a language by a given corpus, which is often huge. This type of representation contains a lot of semantic and syntax information, crucial for the NLP problems. *FLAIR*, as a sequence labeling architecture, which operates upon a Neural Language Model.

Along-side *corpus*, we also have *sentence* as a basic data structure, which is an input, represented as a sequence of characters in a pre-trained language model. From this model, for each word, we obtain an embedding (contextualized or static) which is later used as an input for a $Bi-LSTM-CRF$ (Bi-directional Long-Short-Term Memory - Conditional Random Field) Model.

## A. LSTM

As mentioned before, Language Models are usually very sensitive to its "surroundings", like the author, the state of the data and most importantly, the domain of the data. In the past, statistical methods such as the Hidden Markov Models and Conditional Random Fields were widely used for the classical NLP tasks, but they required a lot of "manual" feature extraction and data pre-processing and yet, they were so task-specific, that it was very hard to adapt one model to a new domain.

In the recent years, the *RNNs* took the "hot-spot" in the NLP world. They are recognized as very powerful networks, being able to capture the dependencies in the data over a longer period of time. This turned out to be problematic due to the fact that these networks are using the *backpropagation* algorithm which is very prone to the *vanishing gradient* problem [4]. Some of the proposed solutions include changing the activation function from *sigmoid* to *ReLU*, like the case with Convolutional Neural Networks (*CNNs*) which are also used in NLP tasks, especially for capturing the morphology of one word. In this paper, we focus of the *LSTM* variant of the RNNs.

The LSTM's architecture consist of a cell with three multiplicative gates (regulators), as shown in Figure 1. The cell is responsible to keep the hidden state for some period of time so it can track the dependencies of the chronologically ordered data. The 3 regulators - the input, output and forget gate, are responsible for the flow of information through the cell, which information to forget and which to pass forward.

Mathematically, the state of the LSTM Unit at a fixed time $t$ is written by the following equations [4]:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f)$$

$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i)$$

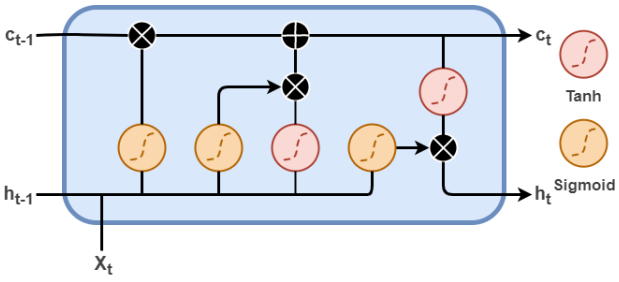


Figure 1. The architecture of a LSTM Unit. The 3 sigmoid functions denote the gates (Forget, Input and Output), $x_t$ is the input (word embedding in our case), $h_t$ is the hidden state of the cell and $c_t$ is the cell state vector. (The figure is recreated based on the article [5])

$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o)$$

$$\tilde{c}_t = tahn(W_c x_t + U_c h_{t-1} + b_c)$$

$$c_t = f_t \odot c_{t-1} + i_t \odot \tilde{c}_t$$

$$h_t = o_t \odot tahn(c_t)$$

The capital letters denote matrices, whilst the lower case letters are vectors. $x_t \in \mathbb{R}^d$ is the input vector for the LSTM unit, $f_t, i_t, o_t \in \mathbb{R}^h$ are the activation vectors for the forget, input(update) and output gates respectively, $\tilde{c}_t \in \mathbb{R}^h$ is the cell input activation vector and $c_t \in \mathbb{R}^h$ is the cell state vector, $W \in \mathbb{R}^{hxd}$ and $U \in \mathbb{R}^{hxh}$ are the weight matrices and $b \in \mathbb{R}^h$ is the bias. The weight matrices and the bias are set at first to their initial values and they are later learned (optimised) in the training phase. $d$ and $h$ are the dimensions of the input vector and the number of hidden states respectively.

1) Bi-LSTM: One word in a sequence labeling task is not only dependent on the words that occurred in the past, but also on the upcoming words. The previous LSTM architecture's hidden state was only taking into consideration the past occurrences which is not enough to shape the whole context. The idea behind the Bi-LSTM is, in addition to the forward model training, to train a separate backward model with the exact same architecture and later concatenate both hidden states in one state that encodes both the past and the future in one sequence.

## B. GRU

The Gated Recurrent Unit or *GRU* is a lite variant of the LSTM network as it works on the same principal, but it lacks the Output Regulator (as seen on Figure 2). It has fewer parameters which makes it more suitable for smaller, unbalanced and rare data sets. The update gate of the GRU has a similar function to the Input and Forget gates of the LSTM, deciding which past information will be kept for the future. The Reset gate, on the other hand,

is responsible for deciding on the amount of past information that will be forgotten. One fully gated unit has the following parameters [6]:

$$z_t = \sigma(W_z x_t + U_z h_{t-1} + b_z)$$
$$r_t = \sigma(W_r x_t + U_r h_{t-1} + b_r)$$
$$\hat{h}_t = tahn(W_h x_t + U_h(r_t \odot h_{t-1}) + b_h)$$
$$h_t = (z - z_t) \odot h_{t-1} + z_t \odot \hat{h}_t,$$

$z_t$ is the update gate vector, $h_t$ is the output vector, $\hat{h}_t$ is the candidate activation vector and $r_t$ is the reset gate vector.

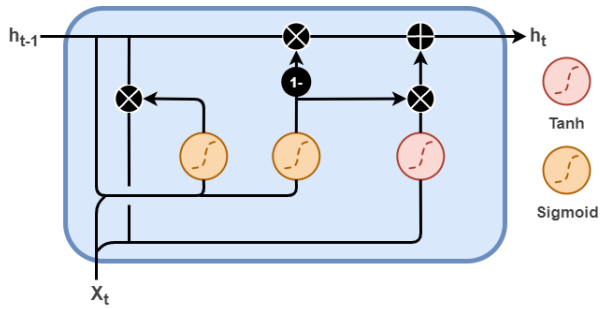


Figure 2. The architecture of a GRU Unit. The sigmoid functions denote the gates (Reset and Update), $x_t$ is the input (word embedding in our case) and $h_t$ is the hidden state of the cell. (The figure is recreated based on the article [5])

### C. CRF - Conditional Random Field

Until now, we have discussed about bi-directionality and how to encode the surrounding of a word in order to extract its full context. However, we might make further improvements if we take into consideration not only the words around the observed one, but also their own labels. When speaking about NER, it is very intuitive that most of the time we see certain group of labels together. On the other hand, we are pretty sure that a tag $t_1$ can never be followed by a tag $t_2$ (e.g. I-PER cannot follow B-LOC [7]), so, if we train our model with these rules, better results could be achieved. The most appropriate tool that can achieve this goal is the CRF-decoder.

The CRF is a discriminating statistical modeling technique based on uni-directional graphs used to calculate the conditional probability of values (tags) on designated output nodes that takes into consideration the distribution of neighbouring nodes' values when assigning a tag [8]. Suppose we have a random variable $X$ consisting of words that need to be labeled and a random variable $Y$ with the corresponding label for each $x_i \in X$ and this $Y$ is a subset of the set $\mathcal{Y}$ that is the space of all possible labels in our world and $P(Y|X)$ is the conditional probability. Then, according to [9], we will define the CRF as a graph $G = (V, E)$, s.t. $Y = (Y_v)_{v \in \mathcal{V}}$, or with other words, our subset of labels Y is denoted by the vertices of G and we say that $(X, Y)$ is a conditional random field when, given $X$, the random variable $Y_v$ satisfies the Markov property (the conditional probability distribution of current state, depends only on the previous state) with respect to the graph:

$$P(Y_v|X, Y_u, u \neq v) = P(Y_v|X, Y_u, u \sim v),$$

s.t. the $\sim$ symbol denotes adjacency between two vertices, or with other words the state $Y_i$, given its neighbours, is conditionally independent from all the other states in the graph $G$.

Our experiments only confirmed the importance of the Conditional Random Field, namely, from the results in Table I, we can observe that using the RNN alone, we obtained almost 0.1 lower F1-Score compared to the case where, additionally, the CRF was used, both for LSTM and GRU.

Table I
Impact of the usage of the Conditional Random Field Layer to the score of the NER_MK model.

| NER - Mcedonian Language | | |
|---|---|---|
| RNN | F1-Score(macro) | F1-Score(micro) |
| GRU + CRF | 0.77 | 0.75 |
| LSTM + CRF | 0.73 | 0.70 |
| LSTM | 0.68 | 0.65 |

## III. Word Embeddings

We can consider word embeddings as the basic concept in NLP. Each word in the corpus must be converted to a tensor with the help of these (mostly) pre-trained models in order to be understandable for the machine. In FLAIR, each *Embeddings* class inherits from the *TokenEmbedding* or *DocumentEmbedding* interface. Both the interfaces require implementing the *.embed()* method for converting a *Sentence* object or a *list* of Sentence objects to a tensor. We can divide these Word Embeddings into two categories: static and contextualized.

### A. Static Embeddings

Static Embeddings have the same representation for each word that has the same form, no matter of its semantic meaning i.e., they do not take into consideration the distribution of words used, before and after the target word we are modeling the embedding for. Static embeddings included in *FLAIR* are *WordEmbedding*, *BytePairEmbeddings*, *CharacterEmbeddings*, *FastTextEmbeddings* and *OneHotEmbeddings*.

## B. FLAIR Contextualized String Embedding

Contextual string embeddings are considered a $break - through$ for the contemporary processing of natural languages. They combine few very important aspects in this field : (1) the possibility to be pre-trained on large unlabeled corpora; (2) they catch the semantic meaning of a word and enable different representations on polysemous words depending on the context; (3) it models the words as a sequence of character, which cancels the problems when a word is not in the given dictionary or simply, it is not spelled right [10].

Most of the time, obtaining the word embedding in a form of a vector (or tensor) is done using the $LSTM$ version of the recurrent neural network. More sophisticated embeddings like $BERT$ use language model transformers with special $Attention$ layers.

They work on a character-level and their main goal is to find a sufficiently good prediction of the distribution $P(c_{0:T})$ for a sequence of characters $(c_0, c_1, ..., c_T) = x_{0:T}$. With the training of the model, it is obtained $P(c_t|c_0, ..., c_{t-1})$ or the prediction for the distribution of the next character, given the distribution of the previous ones. The joint distribution of the whole sentence can be represented as the product of the predicted distributions of all the characters, conditioned from the distributions from their predecessors:

$$P(c_{0:T}) = \prod_{t=0}^{T} P(c_t|c_0, ..., c_{t-1})$$

At LSTM's architecture, the conditional probability, $P(c_t|c_0, ..., c_{t-1})$ is approximately, function of the output of the network $o_t$.

$$P(c_t|c_0, ..., c_{t-1}) \approx \prod_{t=0}^{T} P(c_t|o_t; \theta)$$

where $\theta$ is the vector of all the parameters of the model.

The hidden gem of FLAIR's embeddings is perhaps the wholesome utilisation of the hidden layers of the LSTM network. In addition to the forward model, a backward model is also trained in a complete identical way, but complete opposite direction.

$$P^r(c_t|c_{t+1}, c_{t+2}, ..., c_T) = \prod_{t=0}^{T} P^r(c_t|c_{t+1}, ..., c_T)$$

$$P^r(c_t|c_{t+1:T}) \approx \prod_{t=0}^{T} P^r(c_t|o_t^r, \theta),$$

s.t. $r$ stands for reversed order and $o_t^r$ is again a function of the output of the LSTM network that, in a way, encodes all the data that the model has seen to the given character.

For both cases, a fully connected $softmax$ layer is used which outputs a normalized vector whose dimensions represent the probability of having the certain character next in the sequence.

We can see that the forward model captures the semantic meaning of a sentence up to the point of the observed character, and the backward language model captures the exact same thing, but starting from the end of the sentence up to that character. The main idea is to concatenate both models to obtain the semantic information and the context of the whole word and its surrounding by stacking the outputs of the forward and backward model:

$$we_i^{charLM} = \begin{bmatrix} o^f_{t_{i+1}-1} \\ o^r_{t_i-1} \end{bmatrix}.$$

## C. BERT Embedding

BERT stands for Bidirectional Encoder Representations from Transformers. Its aim, just like the previous model, is to build a pre-trained model from unlabeled data using the deep bi-directional representations from both sides of a word [11]. Although similar, conceptually, as discussed in [12], it differs in the architecture with the FLAIR LMs in a way that FLAIR's LM concatenates the representations for the forward and the backward model, while BERT deeply incorporates the bi-directional context.

The structure of a BERT model consists of an encoder that is a stack of recurrent units (like LSTM or GRU) to process one element of the input sequence (in the word embedding case - 1 character) and to forward the obtained information and a decoder consisting of recurrent units, where each one generates a prediction of an output $y_i$, one at a time with a time step $i$.

1) Encoder: The encoder is a stack of $N$ identical recurrent unit layers, usually 6, 12 or 24 with separate weight vector. Furthermore, each of those layers consists of two subunits, the $Self - Attention$ sublayer [13] and a $Feed - Forward$ Network. The input of the of the encoder is the initial embedding of the word (token, segment and position embedding) and is usually constructed to facilitate the Fine Tuning task for other NLP problems.

The input firstly passes through the Self-Attention layer so that the model can look up for other semantically and syntactically connected words in the sentence. The output of this sublayer is then passed as an input to the feed-forward network.

2) Self-Attention Sublayer: Self-Attention is a concept of finding the relation between particular words in a sentence i.e., if we have the sentence "The quick brown fox jumped over the lazy dog", the aim is to

teach an algorithm that the first "the" and "brown" refer to "fox" and the second "the" and "lazy" to "dog".

From the perspective of an algorithm, the Self-Attention is a process consisting of 6 steps [14]:

- Constructing Query, Key and Value vectors for each of the encoder's input – During the training phase, 3 matrices are optimised $\theta^Q$, $\theta^K$ and $\theta^V$, and each of the inputs is multiplied with one of these matrices to obtain the corresponding vector with fixed length.
- Calculating a score – If we are calculating an embedding for a word in a sentence, say the word number 1, then we are calculating the dot product $QueryVector \cdot KeyVector$ for this word against the remaining words in the sentence. The score for the observed word with itself is always the greatest, because we are doing a dot product between the same vector and obtaining its squared norm. The higher the score, the stronger the connection.
- Divide the score – The 3rd step is to divide the score with the square root of the dimension of the Key vectors in order to obtain a more stable gradient.
- Normalize – The output from the previous step is now passed through a softmax function in order to make all the dimensions positive and normalize the score vector.
- Multiply score with value vector – We multiply the value vector with the softmax score in order to get rid of irrelevant connections with tiny softmax scores.
- Sum – The last step is to sum all the weighted value vectors and obtain the self-attention output for the observed word as

$$Attention(\vec{Q}, \vec{K}, \vec{V}) = softmax(\frac{\vec{Q}\vec{K}^T}{\sqrt{d_k}})\vec{V}.$$

3) Decoder: Just like the encoder, the decoder consists of the same N number of units divided into the same subunits. The input for each of the decoder units are the Self-Attention vectors $K$ and $V$, used for determining the appropriate focus of the decoder in the input sequence. The output of the decoder, which is a vector of floats, is then passed to the final fully connected Linear Unit which transforms it to a much wider vector having one dimension for each word in the vocabulary. The output of this layer will give us a vector of probabilities corresponding to the words in the vocabulary such that the cell for the word with the highest probability is chosen.

### D. Stacked Embeddings

Most of the time, there is a need of combining more embeddings, which in FLAIR is enabled by the *StackedEmbeddings* class. It is initialized with a list of the desired embeddings, stacked on one another,

$$we_i = \begin{bmatrix} we_i^{charLM} \\ we_i^{static} \end{bmatrix},$$

and later on, they function as a regular Embedding class, i.e., they inherit the *TokenEmbedding* interface and define the *.embed()* method. Apart from FLAIR Forward-Backward and BERT embedding, *Flair* also includes *PooledFlairEmbeddings*, *ELMoEmbeddings* and *TransformerWordEmbeddings* which include a few classical variants of pre-trained transformers, such as *RoBERTa*, *XLM*, etc.

Table II
Impact of the embedding types on the score of the NER_MK model.

| NER - Mcedonian Language | | |
|---|---|---|
| Embeddings | F1-Score(macro) | F1-Score(micro) |
| FlairEmbeddings+ BERT | 0.77 | 0.75 |
| FlairEmbeddings+ BytePair+ BERT | 0.67 | 0.64 |
| WordEmbeddings+ BytePair+ BERT | 0.63 | 0.59 |

Remark 1. Although there is a statement from *ZalandoResearch* that the best results for *NER* are obtained by combining contextual and static embeddings, our model worked better when only the contextual embeddings were taken into consideration.

## IV. NER model - Macedonian Language

The experiment for the NER model built for the Macedonian language consisted of obtaining the data set, which included scraping it from the WEB, labeling and conversion to *CoNLL* format used by FLAIR, and than, building and testing few different models, further discussed later on.

### A. Data

The data that used in this experiment was scraped from Macedonian web pages for on-line news, *time.mk* and *greed.mk*. We considered news from Macedonia, the Balkan and the world belonging mostly to finance, economy, politics, life, chronicle, culture, technology and scene. Part of the data was obtained using *fetchRSS* in a *csv* format, while the the other part was scraped using the *BeautifulSoup* library in Python. The labeling of the data was done using the *doccano* software. At the end, we obtained 13 relevant classes of news, each one containing sufficiently big number of instances for training.

FLAIR uses all of the CoNLL formats for data representations in textual files. Each token is given in a new, separate row with its appropriate label and tag, that can be $B-tag$, $I-tag$ and $O-tag$ ($BIO$ stands for Beginning, Inside, Outside [15]). When a given token does not have any label, it is marked with the O tag. When a token is part of some entity, it is marked with either the I tag or the B tag, so that if that particular token is the beginning of the entity, it is assigned the B-tag (as shown in Figure 3). The I-tag is only used when a given token has the same entity as the previous one without any O tags between them (multi-token entities). There are also extensions to this format, such as the $BIOES$ format.

| | |
|---|---|
| Велика | *B-Location* |
| Британија | *I-Location* |
| и | *O* |
| САД | *B-Location* |
| почнаа | *B-Event* |
| преговори | *I-Event* |

Figure 3. IOB-tagging example.

1) Data Tokenization: Another very important aspect, when it comes to processing a text, is the sentence tokenization. The correctness of the distribution of the labels in the text is directly conditioned on the type of the tokenizer used to split the sentences into separate words i.e., tokens. Tokenizers can be as simple as splitting sentences on white spaces, but there also exist more advanced and contemporary models like the $SpaCy$, $NLTK$ and $Segtok$ tokenizers.

Choosing the tokenizer mainly depends on the nature of the problem we are working with and the granulation of the tokens we expect to achieve. Some NLP problems that are tightly dependant on the grammatical composition of the words i.e., text summarization, where the output is a sequence of words that we expect to be grammatically correct, may not work well if we perform tokenization by simply splitting on white spaces. These kind of problems require pre-trained language models as tokenizers which are able to split the sentence depending on its grammatical formulation. Let us consider the string $aren't$. If we use a simple split on white spaces, this remains as a single token. Otherwise, two separate tokens $are$ and $n't$ are obtained.

For our particular task, named entity recognition for Macedonian language, we compared the two different approaches when it comes to text tokenization: the simple split lines function on white spaces and the $Segtok$ tokenizer, which is the default tokenization technique used by the FLAIR framework. One interesting conclusion we came to after obtaining the results is that the later worked the best, but only when we filtered the punctuation-only tokens that were not part of any label. Otherwise, the simple split lines did a better job than Segtok itself. Considering that punctuation is quite important part of the labeled tokens for the NER problem, we used the simpler tokenization approach.

## B. Training

The training phase was mostly experimenting with the embeddings, the type of the RNN and the hyper-parameters of the model.

Taking into consideration the similarity of the Macedonian language with the other Slavic languages (Serbian, Russian, Slovenian, Czech etc.), the idea was to stack all these WordEmbeddings together with the BytePairEmbeddings for the Macedonian language in order to get satisfying results. However, not only that the results were unenviable, but the training process took to much time as a result of the huge dimensions of the input embeddings.

The next step was to get a subset of the previous languages (the South Slavic group) and combine them with contextualized string embeddings like FlairEmbeddings and BERTEmbeddings, which slightly improved the results, but as mentioned before, the best results were obtained when only the contextualized multi-language embeddings were taken into consideration, in our case FlairEmbeddings('multi-forward'), FlairEmbeddings('multi-backward') and BERT (as shown in Table II).

When it comes to the type of the RNN that should be used in a NER Language Model, there is one question that should be asked – How big is the data set? The differences between LSTM and GRU are already discussed, so we can conclude that when having a huge and complex data set, the LSTM is expected to get better results. When the data set is smaller and rare, then the GRU variant is expected to perform better in most of the cases. For best results, the safest variant is to check both LSTM and GRU!

The most reliable hyper-parameter in any ML model is probably the learning rate, the step used in the gradient descent algorithms for optimising the loss function. The most common and recommended value for this parameter in the NER tasks is 0.1, which proved true in our experiments. Increasing the learning rate would only increase the number of epochs it takes the loss-function to converge, and decreasing it would make the loss function to converge too early and leave the model with way less information about the data. Other useful hyper-parameter that can be used is the reproject_embeddings parameter, which adds a trainable layer on top of the fully connected

layer for re-projecting the word embeddings. Often there is a problem with an unbalanced data set and selective learning, or simply, catching a certain class of labels is more important then an other. For any of these cases, the weight of each tag can be set independently with the loss_weight parameter. There are several other hyper-parameters not mentioned because they were left as the defaults or simply the FLAIR recommendation was taken into account.

The discussed model is trained with 110 epochs over a network with hidden size of 256 layers, starting with a learning rate of 0.1. From Figure 4 we can conclude that the model learned the most information with the learning rates of 0.1, 0.05 and 0.025. As shown in Figure 5, the F1 of the training set starts to converge at epoch 80.

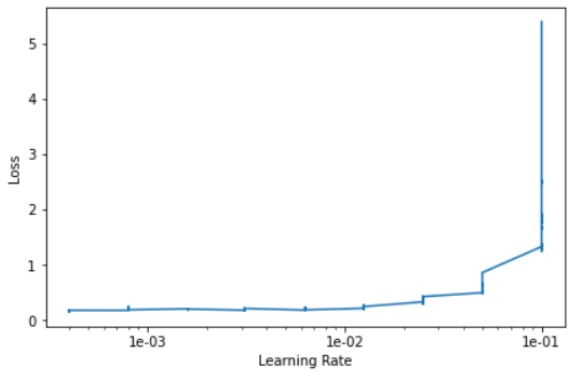


Figure 4. loss vs. learning rate

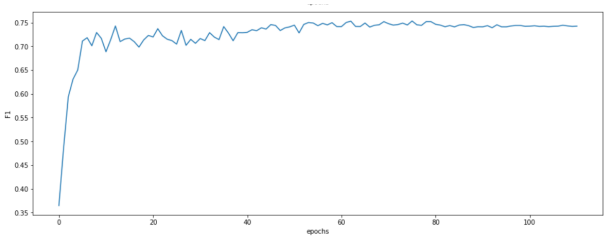


Figure 5. f1 vs. epochs

## V. Results and Evaluation

The main task of this model was to label words in unseen sentences with one of the 13 previously mentioned classes. From the results, as shown in Table III, the classes with more static surroundings like the "PERCENT" and "AMOUNT" were not a problem for our model at all. The reason for the good results is that the labeled segments from the "PERCENT" class contain the % sign or the word "percent" with very high probability. The same applies to the labeled segments from the "AMOUNT" class which most certainly contain currency as a word or a symbol. The more challenging class, that still performs well, is the "NUMBER" class. The main issue with the labeled segments from this class is that they are mistaken with the "PERIOD" class containing numeric time ranges. Labeling entities with the "EVENT" class gives the worst results since it events in texts are very stochastic in nature and they usually do not have fixed distribution of surrounding words like the "PERCENT" and "AMOUNT" class. As we mentioned before, a possible way to improve the results for certain classes with unsatisfactory results is to increase their weight, but in this case resetting the weights only affected the results of the model negatively i.e. the amount of the improvement we achieved for the "EVENT" class was not worth the decrease of the overall model's score.

Table III
Distribution of TP, FP and FN for the testing ser in the classes.

| NER - Mcedonian Language | | | |
|---|---|---|---|
| Label | TP | FP | FN |
| AMOUNT | 9 | 0 | 4 |
| COMPANY | 78 | 22 | 33 |
| EVENT | 71 | 58 | 78 |
| LOCATION | 110 | 21 | 26 |
| MEDIA TYPE | 16 | 3 | 9 |
| NUMBER | 117 | 40 | 23 |
| PERCENT | 3 | 0 | 0 |
| PERIOD | 94 | 18 | 20 |
| PERSON | 116 | 20 | 14 |
| POLITICAL ENTITY | 32 | 8 | 7 |
| PRODUCT | 59 | 32 | 26 |
| PUBLICATION | 19 | 7 | 7 |
| SECTOR | 10 | 4 | 5 |

For a comparison, the same corpus with data in Macedonian language was fed to a BERT Transformer with Attention Layer model, sharing the same hyper-parameters. As shown in Table IV, The FLAIR model gave better results for the observed parameter space, however, this may be due to unexplored parameter space in our implementation, as we focused more on achieving better results with FLAIR's bi-directional language model.

Table IV
Comparison of FLAIR and BERT models.

| NER - Mcedonian Language | | |
|---|---|---|
| | F1-Score(macro) | F1-Score(micro) |
| FLAIR | 0.77 | 0.75 |
| BERT | 0.55 | 0.52 |

As Macedonian language is part of the South group of the Slavic spoken languages, we found it challenging to see what information our model can extract from another language of the same group. Due to the similarity, we chose to make this experiment with a

Serbian corpus with data obtained from on-line Serbian portals. Most of the common classes like "PERSON", "LOCATION" and "POLITICAL_ENTITY" were detected with high accuracy, having f1 scores of 0.83, 0.8 and 0.66. However, the "EVENT" class did not perform well in this case too. The micro f1-score for the whole corpus is 0.61 and the macro f1-score is 0.48. The bit gap between these two values is due to the small size of the testing data set and the absence of sufficient number of data points to test on.

We also tried to check how a multi-lingual model would behave when fed with data from two completely different languages belonging to different language families, like Macedonian and Albanian. As expected, FLAIR's multi-lingual model did a great job in combining these two, such that there were no obstructions from one language to another in the training phase. This turned out to be a great technique when dealing with corpora with inefficient amount of data, as the whole process acts in a kind of a data augmentation way. The Macedonian corpus was supported by the Albanian and vice versa, which eventually helped the loss function to converge faster, since we had more training data and increased the test scores for both the languages.

## VI. Conclusion

With the rapid displacement of the real world towards the machines and the Internet, there is a rapid increasing of the need for changing the way we do the everyday tasks from manually to automatically. The Natural Language Processing algorithms are designed to go through and process huge amount of texts and extract information from it, that can be not foreseen with the naked eye of a human. The Named Entity Recognition is only one branch from the wide palette, that is helping us to focus on the right grain in the send of words. When building such a model, one should always be aware in which domain and by who that model is going to be used. FLAIR is one modern solution to this problem, that minimises the manual work with the data and optimises the results, but most importantly of all, it supports and eases the development of models for smaller languages (as Macedonian is) with little to no resources available. Furthermore, we used corpus from a language from the same language group as Macedonian and obtained satisfactory results for the same classes as for Macedonian, proving how powerful the concept of multi-lingual language models is. We believe that this Named Entity Recognition model is an important point for the Macedonian language processing and information extraction and that, it will encourage further researches in the field of NLP, as well as broaden the amount of available labeled data in Macedonian.

## References


[1] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," 2017. [Online]. Available: https://arxiv.org/pdf/1706.03762.pdf

[2] A. Mikheev, M. Moens, and C. Grover, "Named entity recognition without gazetteers," in Ninth Conference of the European Chapter of the Association for Computational Linguistics, 1999.

[3] A. Akbik, T. Bergmann, D. Blythe, K. Rasul, S. Schweter, and R. Vollgraf, "Flair: An easy-to-use framework for state-of-the-art nlp," in NAACL 2019, 2019 Annual Conference of the North American Chapter of the Association for Computational Linguistics (Demonstrations), 2019, pp. 54–59.

[4] X. Ma and E. Hovy, "End-to-end sequence labeling via bi-directional lstm-cnns-crf," arXiv preprint arXiv:1603.01354, 2016.

[5] M. Phi, "Illustrated guide to lstm's and gru's: A step by step explanation," 2018. [Online]. Available: https://towardsdatascience.com/illustrated-guide-to-lstms-and-gru-s-a-step-by-step-explanation-44e9eb85bf21

[6] K. Irie, Z. Tüske, T. Alkhouli, R. Schlüter, and H. Ney, "Lstm, gru, highway and a bit of attention: An empirical overview for language modeling in speech recognition." in Interspeech, 2016, pp. 3519–3523.

[7] G. Lample, M. Ballesteros, S. Subramanian, K. Kawakami, and C. Dyer, "Neural architectures for named entity recognition," arXiv preprint arXiv:1603.01360, 2016.

[8] P. M. Shishtla, K. Gali, P. Pingali, and V. Varma, "Experiments in telugu ner: A conditional random field approach," in Proceedings of the IJCNLP-08 Workshop on Named Entity Recognition for South and South East Asian Languages, 2008.

[9] J. Lafferty, A. McCallum, and F. C. Pereira, "Conditional random fields: Probabilistic models for segmenting and labeling sequence data," 2001.

[10] A. Akbik, D. Blythe, and R. Vollgraf, "Contextual string embeddings for sequence labeling," in Proceedings of the 27th International Conference on Computational Linguistics, 2018, pp. 1638–1649.

[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," arXiv preprint arXiv:1810.04805, 2018.

[12] S. Schweter and A. Akbik, "Flert: Document-level features for named entity recognition," 2020.

[13] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in Advances in neural information processing systems, 2017, pp. 5998–6008.

[14] J. Alammar, "The illustrated transformer," 2018. [Online]. Available: http://jalammar.github.io/illustrated-transformer/

[15] L. A. Ramshaw and M. P. Marcus, "Text chunking using transformation-based learning," in Natural language processing using very large corpora. Springer, 1999, pp. 157–176.