FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

MASTER THESIS

Eric Lief

Deep Contextualized Word Embeddings from Character Language Models for Neural Sequence Labeling

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Doc. RNDr. Pavel Pecina, Dr.

Study programme: Computer Science

Specialization: Computational Linguistics (4I3)

2019

I would like to express my gratitude to Pavel Pecina, without whom this thesis would not have been possible, for his kindness and encouragement from the start.

I would also like to thank Milan Straka. His knowledge, enthusiasm, patience, and helpfulness are truly inspiring.

I am indebted to Dušan Varis for helping me with the many technical aspects of this thesis.

Many thanks to my colleagues at MFF for creating a positive, supportive, and collaborative study environment.

Finally, I would like to thank my family for their love and support which have kept me going.

Title: Deep contextualized word embeddings from character language models for neural sequence labeling

Author: Eric A. Lief

Institute of Formal and Applied Linguistics

Supervisor of the master thesis: Doc. RNDr. Pavel Pecina, Dr. Institute of Formal and Applied Linguistics

Abstract: A family of Natural Language Processing (NLP) tasks such as part-of-speech (PoS) tagging, Named Entity Recognition (NER), and Multiword Expression (MWE) identification all involve assigning labels to sequences of words in text (sequence labeling). Most modern machine learning approaches to sequence labeling utilize word embeddings, learned representations of text, in which words with similar meanings have similar representations. Quite recently, contextualized word embeddings have garnered much attention because, unlike pretrained context-insensitive embeddings such as word2vec, they are able to capture word meaning in context. In this thesis, I evaluate the performance of different embedding setups (context-sensitive, context-insensitive word, as well as task-specific word, character, lemma, and PoS) on the three abovementioned sequence labeling tasks using a deep learning model (BiLSTM) and Portuguese datasets.

Keywords: [part-of-speech tagging, named entity recognition, multiword expression, embedding, deep learning, Portuguese]

Contents

PREFACE

The major contribution of this thesis is its thorough exploration into the use of different experimental setups with embeddings applied to sequence labeling tasks (the assigning of labels to sequences of words in text) using deep learning[1]. Three tasks are carried out using Portuguese datasets: part-of-speech (PoS) tagging, named entity recognition (NER), and verbal multiword expression (VMWE) identification. Secondary contributions are in the advancement of the state of the art in PoS tagging for Portuguese and near state of the art results without the use of handcrafted features for Portuguese NER and VMWE identification.

The structure of this work is as follows.

Chapter 1 opens with a light introduction to deep learning methods and architectures, word embeddings, and language models.

In Chapter 2, we explore the use of character language models (CharLMs) in state-of-the-art sequence taggers.

Chapter 3 describes the deep learning architecture of the sequence tagger model used for all tasks as well as the different experimental setups.

Chapters 4, 5, and 6 present the tasks of PoS tagging, NER, and VMWE identification respectively, as well as the Portuguese data, the state of the art, the parameters of the tagging model, and the results of the experimental setups.

Chapter 7 concludes this thesis with final remarks.

1. All code and data is freely available at https://github.com/ericlief/sequence-tagger.git

CHAPTER 1

Introduction

In this introductory chapter, I present an overview of sequence labeling tasks and a brief introduction to popular current deep learning architectures, embeddings, and language models.

1.1. Sequence Tagging

Sequence tagging or labeling commonly refers to a group of related Natural Language Processing (NLP) tasks such as part-of-speech (PoS) tagging, Named Entity Recognition (NER), Multiword Expression (MWE), chunking/shallow parsing, and semantic role labeling/semantic slot filling. All of these tasks can be treated as cases of supervised multinomial classification, in which a tag or label is assigned to some linguistic construction, be it a word (PoS tagging) or a grouping of words referring to some unique entity (NER) such as *the Czech Republic*.

When neural networks are employed to solve these sequence labeling tasks, it is essential that each input, whether a character or a word, maps to a target label. In order to achieve this, usually some variation of the BIO (Beginning, Inside, Outside) annotation scheme is used. Below in (1), the tag for Location can be assigned to either a single or a multiword entity:

(1)	*Prague*	*is*	*the*	*capital*	*of*	*the*	*Czech*	*Republic*
	B-LOC	O	O	O	O	B-LOC	I-LOC	I-LOC

The presence of the multiword entity is thus signaled by the Inside (I) *LOC* tag following a same Beginning (B) tag, and all words that do not span any entity are marked as Outside (O). As we will see below in the discussion of multiword expressions (MWEs), there are slight variations of this annotation scheme, but the idea is for the most part the same: every word form in a string of words (text)—instead of some larger linguistic entity[2]– gets its own tag or label.

Current state of the art approaches to sequence labeling use some type of language model (LM), possibly combined with pretrained word embeddings, and for the labeling task, a bidirectional recurrent neural network (BiRNN), such as the LSTM variety, with a conditional-random field (CRF), henceforth BiLSTM-CRF (Akbik et al 2018, Peters et al 2018). Before discussing these architectures, I first present some basic machine learning and neural network concepts.

1.1.1 Conditional Random Fields for Sequence Labeling

Most current SOA approaches to sequence labeling employ a conditional random field (CRF) to decode the output label sequence. CRFs are much more powerful than generative models such as Hidden Markov Models (HMMs) and do not suffer from the label bias problem (Lafferty et *al*. 2001). The label bias problem refers to the fact that finite state transitions from one state to the next are evaluated locally rather than globally. In other words, the best transition (i.e. the *argmax*) is only the best leaving the current state, but may not be the best one overall. For some sequence labeling tasks, we may be able to make certain independence assumptions when assigning one label after another, e.g. if we are classifying a stream of images or the like. For linguistic data, however, we are not always able to assume such independence. PoS tagging, for example, reflects hard linguistic constraints, e.g. in English a determiner cannot follow a noun, *NOUN + DET. Thus when assigning DET, it would benefit the model to have access to the previous tag. A CRF, in effect, achieves this but looking at the whole sequence when assigning labels.

CRFs are a log-linear model much like logistic regression, and are formally represented as weighted feature functions. A feature function f_j takes as arguments the label of the preceding segment y_{i-1}, the current label y_i, and the current sequential position, and assigns a binary score of 0 or 1 if the condition is true. For instance if the preceding label is a determiner and the current label is noun. Next, a positive or

2. Some recent deep learning approaches assign a label to some larger chunk, e.g. Zhai et al (2017).

negative weight λ_j is associated with a feature function, reflecting how favored the feature is. Below in (2), for a sentence of length *n*, there are *m* features functions:

$$P(\mathbf{y}|\mathbf{x}) = \frac{exp[\sum_{j=1}^{m}\sum_{i=1}^{n}\lambda_j f_j(y_i, y_{i-1}, x_i)]}{\sum_y exp[\sum_{j=1}^{m}\sum_{i=1}^{n}\lambda_j f_j(y_i, y_{i-1}, x_i)]} \tag{2}$$

Thus sequence labeling is not done for each token in isolation; rather, the best tag sequence is computed on the basis of the probabilities of all labels assigned in the surrounding context.

1.2. Deep Learning

Deep learning refers to a family of machine learning methods and architectures, which typically use many hierarchical layers of non-linear processing (Deng and Yu 2014). According to Deng et. al (2014), there are three basic classes of deep networks:

1. Unsupervised or generative deep networks, with the goal of gaining insight into the correlation of observed data, in the absence of target labels

2. Unsupervised learning or discriminative deep networks, with the goal of classing data in the presence of target labels

3. Hybrid deep networks, which are a combination of supervised classification, aided by the outcomes of unsupervised generative nets.

In this thesis focusing on sequence labeling (i.e. classification), we will be concerned only with supervised learning. In the next section, an overview of the most popular and successful deep learning architectures is presented.

1.2.1 Recurrent Neural Networks (RNNs)

Recurrent Neural Networks (RNNs) are a type of artificial neural network specialized for processing sequential data such as text, which can send feedback signals (Chen, 2016). Normal vanilla flavored neural networks suffer from two major problems. First they cannot be fed variable length inputs, whether characters, words, or sentences. There are nevertheless workarounds, such as padding the sequence with zeros up to some fixed maximum length. Second, they do not share features learned across different sequential positions. In NLP, a word like *Prague* occurring initially bears no relation to the same word occurring word medially or finally. RNNs can handle sequences of different lengths, and parameter sharing allows them to generalize across different textual positions (Ian Goodfellow, Bengio, & Courville, 2016). Thus they would need separate parameters for each sequential position *t* in order to recognize that the named entity *President Trump* (Person) is the same in the following two sentences.

(3) ***President Trump*** *promises to make America great again.*

(4) *During his election campaign,* ***President Trump*** *promised to make America great again.*

As the name suggests, RNNs compute the current state recursively. The previous state serves as input to the current state, yielding a recurrence such as the following $\boldsymbol{h}_t = f(\boldsymbol{x}_t, \boldsymbol{h}_{t-1}; \theta)$. We thus see that each state contains information (features) from all previous states. We can visualize this recurrence better by unfolding it (image to the right) in Figure 1 below.

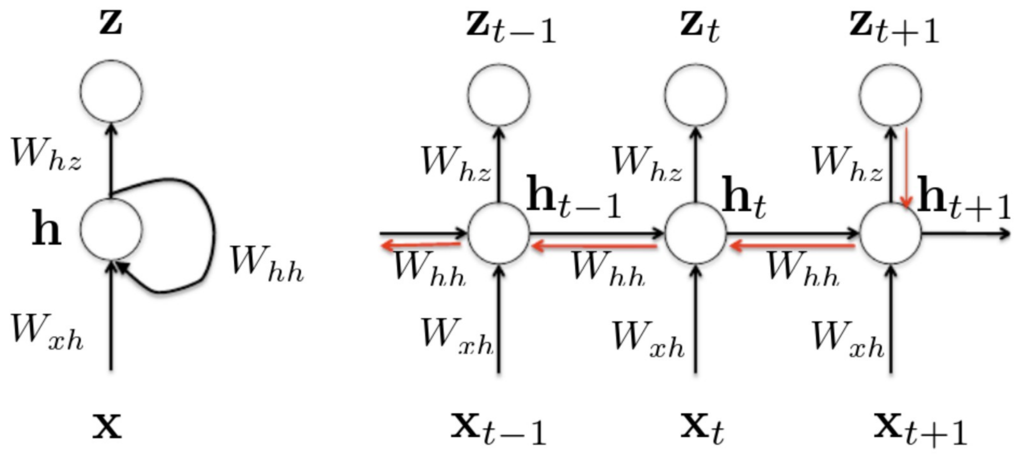


Figure 1: RNN (Folded and Unfolded), from Chen (2016)

RNNs are thus capable of capturing longer range dependencies than a vanilla flavored neural network, e.g. a multilayer perceptron (MLP). For instance, an RNN maps a sequence $\boldsymbol{x}_1$, $\boldsymbol{x}_2$,...$\boldsymbol{x}_t$ to a state $\boldsymbol{h}_{t+1}$, which depending on the length of the sequence, may be lossy (Ian Goodfellow et al., 2016). A language model, which models the distribution of words in a language and can predict the next word (or character) based on previous words, begins to lose information beyond a certain window size.

1.2.2 Long Short-Term Memory (LSTM)

Long Short-Term Memory (LSTM), along with Gated Recurrent Units (GRUs), are gated RNNs, which employ this specialized gated architecture to control what information is allowed to pass through. As illustrated in the figure below, an LSTM contains a memory cell ($\boldsymbol{c}_t$) which hold the state, an input gate ($\boldsymbol{i}_t$) which controls what new input features are incorporated into the state, a forget gate ($\boldsymbol{f}_t$) which controls which information from the state at the previous time step ($\boldsymbol{c}_{t-1}$) recursively passes into the state, and an output gate ($\boldsymbol{o}_t$) which can selectively shut off the output of the cell.

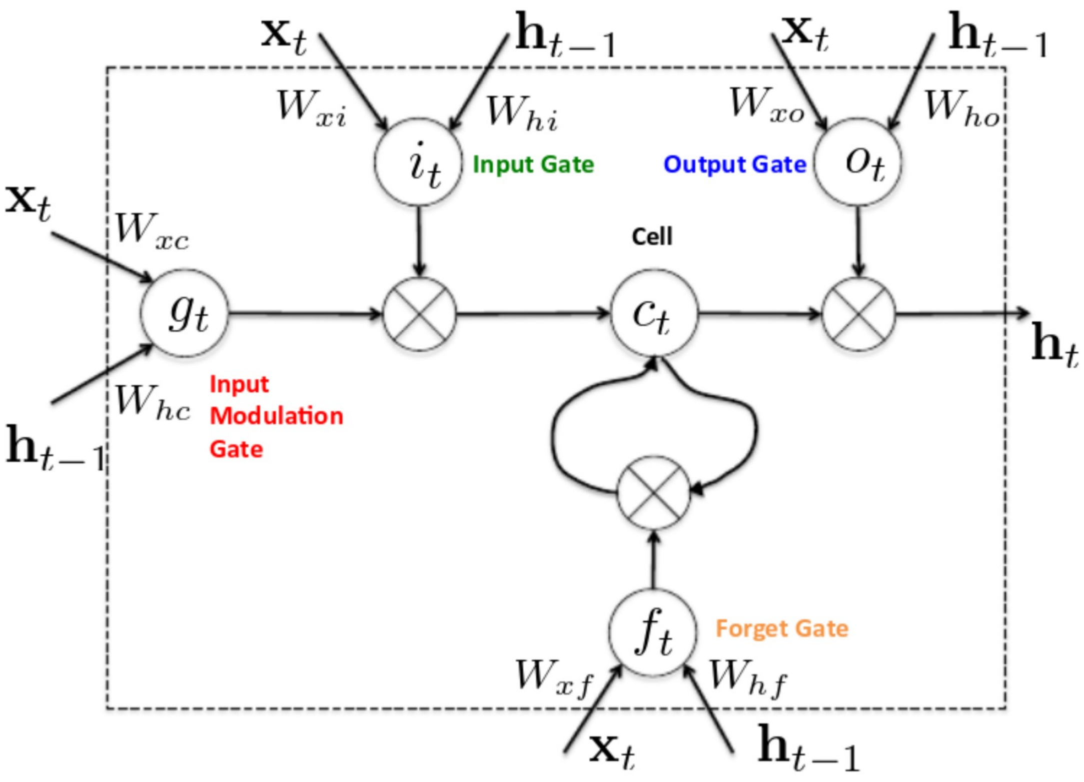


Figure 2: LSTM (Chen 2016)

The computations of these gates and cell state are given below in (5).

(5) LSTM gate computations

$$
\begin{aligned}
\mathrm{i}_t &= \sigma(W_{ii}x_t + b_{ii} + W_{hi}h_{(t-1)} + b_{hi}) \\
f_t &= \sigma(W_{if}x_t + b_{if} + W_{hf}h_{(t-1)} + b_{hf}) \\
g_t &= \tanh(W_{ig}x_t + b_{ig} + W_{hg}h_{(t-1)} + b_{hg}) \\
o_t &= \sigma(W_{io}x_t + b_{io} + W_{ho}h_{(t-1)} + b_{ho}) \\
c_t &= f_t c_{(t-1)} + i_t g_t \\
h_t &= o_t \tanh(c_t)
\end{aligned}
$$

Note that in the standard LSTM, each gates receives to inputs, the input $\boldsymbol{x_t}$ and previous state $\boldsymbol{h_{t-1}}$, and that the activation of the gates is normally sigmoid σ, but that the activation of the external input gate ($\boldsymbol{g_t}$), which by element-wise multiplication (*) controls what input enters the state, is *tanh*, as is the final state output by the cell.

1.3 Embeddings

A word embedding (Collobert et *al*. 2011, Mikolov et *al*. 2013c), also known as a distributed word representation, is a learned representation of text, in which words with similar meanings have similar representations (Bengio, Ducharme, Vincent, & Janvin, 2001). This differs from merely representing a word as a vector containing no semantic features, an idea which has been around for some time. If we are dealing with true word embedding vectors, the representation of *vec*(*Prague*) can be obtained arithmetically by *vec*(*Rome*) - *vec*(*Italy*) + *vec*(*The Czech Republic*). One of the advantages of using word embeddings is that they in effect project a discrete high dimensional space of a magnitude of the number of words in the vocabulary to a much more manageable continuous lower dimensional space. Thus a word with a discrete one-hot encoding of dimensionality in the tens of thousands is mapped to a continuous representation encoding normally only in the hundreds. Word embeddings are one of the breakthroughs in NLP in the last fifty years.

1.3.1 Context-Insensitive Word Embeddings

There are two basic techniques to embed words: continuous bag of words (CBOW) and skip gram (Zheng, Shi, Guo, Li, & Zhu, 2017). In the CBOW model, a word is predicted by its context of surrounding words, while in the skip gram model, the process is reversed and the context of a given word is predicted.

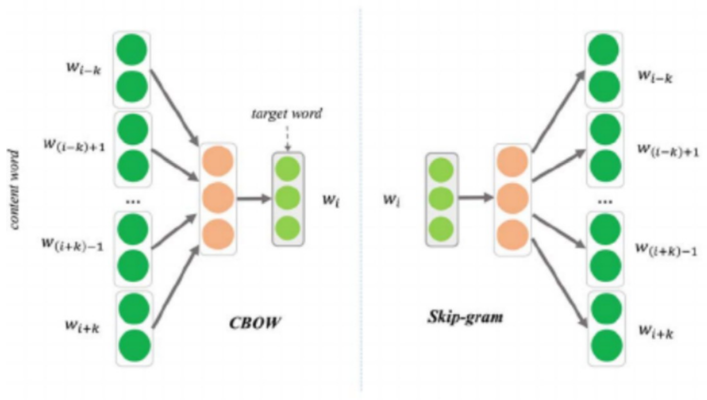


Figure 3: CBOW and Skip Gram (Zheng et al. 2017)

Word2vec can create embeddings using either one of these techniques, and it has been shown that of the two methods the skip gram is the best at capturing word similarity (Mikolov et *al*. 2013 d). Although pretrained distributed word embeddings such as *word2vec*, FASTTEXT, and GloVe have proven quite useful in modeling the latent semantic and syntactic similarities of words, they suffer the drawback that they do not model how words very frequently have different meanings (polysemy). This is the case because different meanings and representations are collapsed into one final representation. Schütze (1998) is one of the first to identify what he called the meaning conflation deficiency. One of obvious outcome of this is that an Natural Language Understanding (NLU) system would be less likely at distinguishing these senses. Another perhaps less obvious one is that often unrelated words are pulled toward each other in vector space. For instance, the unrelated word *rat* and *screen* are pulled toward one another due to one of the senses of *mouse* and its closeness to *screen* (Camacho-Collados & Pilehvar, 2018).

1.3.2 Contextualized Word Embeddings

One obvious solution to the meaning conflation deficiency described in §1.3.1 is to have embeddings for every meaning of a word. Such sense representations or embeddings can be produced either in an unsupervised fashion or with the help of a knowledge base (KB). Unsupervised approaches are favorable because of the lack of KBs and make use of clustering, which can lead to unclear semantic mappings and consequently difficult integration into downstream models (Camacho-Collados & Pilehvar, 2018). Recently an emerging field of research has sought to integrate unsupervised context-sensitive embeddings into downstream tasks. While pretrained word embeddings like word2vec yield one single static representation per word, these contextualized word embeddings are sensitive to context and their representation is dynamic, changing according to the context in which the word appears (Cassani, Tomadoni, Ponce, Agüero, & Moreira, 2017; Melamud, Goldberger, & Dagan, 2016; Peters, Ammar, Bhagavatula, & Power, 2017; Akbik, Blythe, and Vollgraf 2018)

1.3.2.1 Context2vec

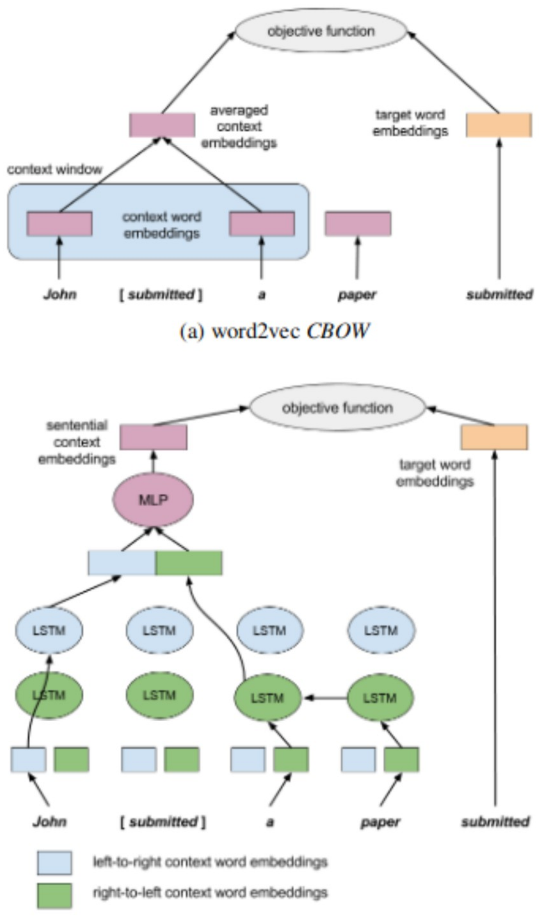
Like word2vec, context2vec (Melamud et al., 2016) learns both target word and context representations simultaneously, but with a much more robust sentential context. As in Figure 5 below, the model uses two separate LSTMs, one for the left-to-right context and one for the right-to-left, with separate parameters and context embeddings. The outputs of each one are then concatenated and thus encapsulate the context, in much the same way as the averaging over context vectors in the word2vec model. This context vector is next fed to a Multilayer Perceptron (MLP) which outputs a representation of the entire sentential context. (Melamud et al., 2016)

Context2vec shares a lot in common with Language Models (LMs). Both generally use bidirectional LSTMs (BiLSTMs) such as that described above, with the objective of predicting the target word based on its context. While LMs, however, are mainly concerned with predicting conditional probabilities of target words based on their histories, the main goal of context2vec is to produce useful context representations of target words. However, as we will soon see, character language models (CharLMs) also often share this goal. In the next section we discuss LMs in more detail.

1.4 Language Models

A statistical language model (LM) is formally a probability distribution over words. Using a LM, one can estimate the probability of observing a particular sequence of words, whether a sentence or a whole document e.g. $P(w_1,..,w_T)$

(6) $$P(w_1,..,w_T) = \prod_{t=1}^{T} P(w_t|w_1,...,w_{t-1}), \quad let\, T \equiv text$$



(a) word2vec *CBOW*

(b) *context2vec*

Figure 4: Context2vec (Melamud et al. 2016)

Language modeling has applications in speech recognition, machine translation, part of speech tagging, text generation, and information retrieval, among others. Due to the data sparsity problem, the Markov assumption, limiting the history to *n*-terms, is usually employed in practice, and *n*-gram probabilities are often computed by maximum likelihood estimation (i.e. counting), followed by smoothing to counteract out of vocabulary (OOV) words. Therefore, by an nth order Markov property, the probability distribution, which is a product of terms predicting the next word given the history, can be approximated by a product of conditional probabilities confined to a window of size *n*.

(7) $$P(w_1, .., w_T) = \prod_{t=1}^{T} P(w_t | w_1, ..., w_{t-1}) \approx \prod_{t=1}^{T} P(w_t | w_{t-n+1}, ..., w_{t-1})$$

In order to accurately estimate the likelihood of sequences of words, LMs must be trained on a huge amount of data. Nevertheless, a sentence at test time may still be different from those seen during training, and with more data comes a larger vocabulary size, more possible sentences, and greater data sparsity, a phenomenon known as the ‘curse of dimensionality’ (Bengio et al., 2001). Traditional approaches overcame this problem by stringing together n-grams to build the sentence.

In the last decade, neural LMs (NLMs) have gained popularity because of their ability to alleviate the curse of dimensionality via their use of continuous word representations (embeddings). As discussed in §1.3, the use of word embeddings leads to dimensionality reduction. In an NLM, word embeddings are learned during training. One simply initializes a lookup table randomly and embeds the sequences using this table. The lookup table is a parameter of the network and gets updated like the other parameters during backpropagation. The goal of a NLM is to learn an optimal model $f(w_{t-n+1},..,w_{t)=P(w_t|w_1,...,w_{t-1})}$. This process is illustrated below in Figure 6. After encoding the input as a word vector (embedding) using the lookup table C, the model maps the input sequence to a conditional probability distribution over all words in the vocabulary for the next word given its history.

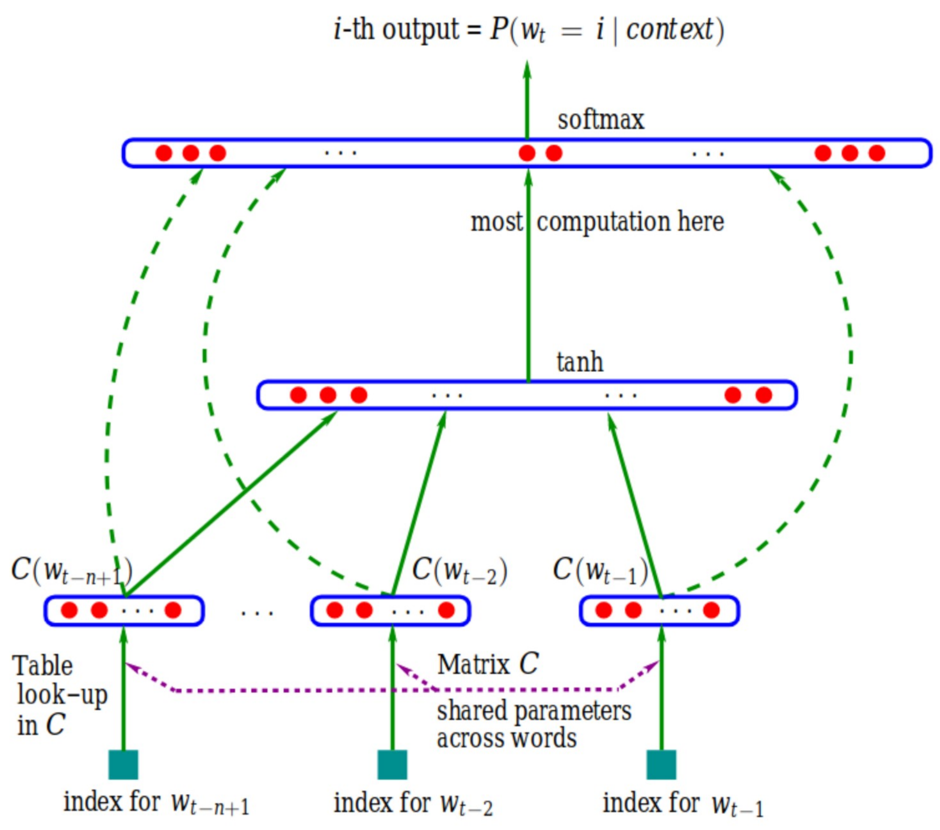


Figure 5: Classical Neural LM, from Bengio et al. (2003)

Although as we will see below the classical model of (Bengio et al., 2001) has evolved, its basic components have remained the same, consisting of the following layers:

1. Embedding or projection layer, which makes use of a lookup table to vectorize the input
2. Hidden or encoding layer, which internalizes the context around the target word
3. Output or softmax layer, which outputs a normalized probability over all words in the vocabulary.

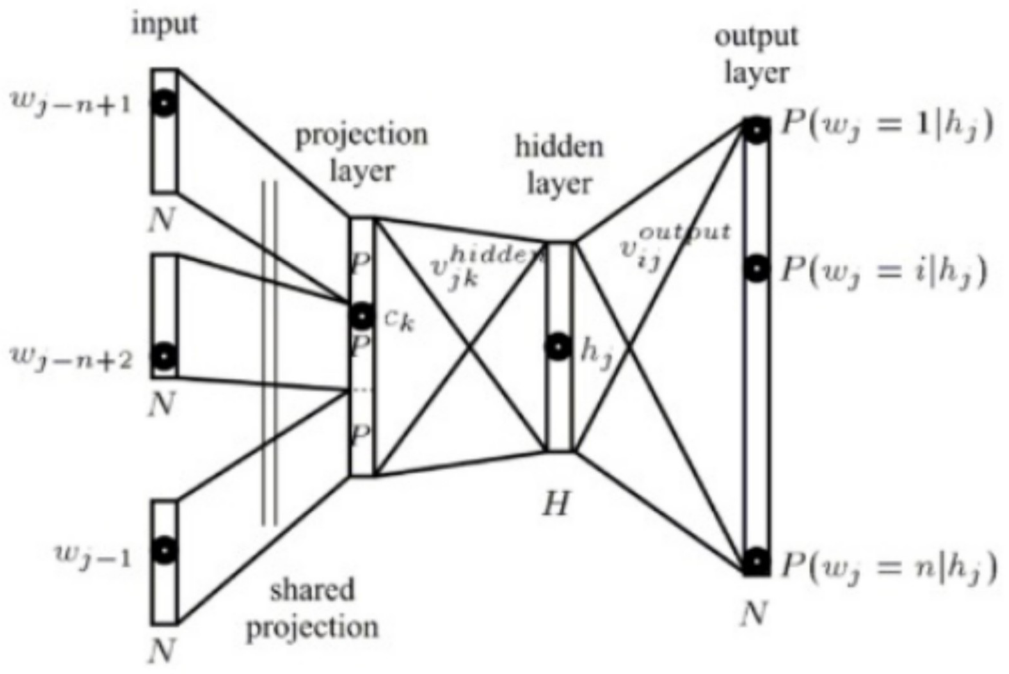


Figure 6: Components of a NLM, from Bengio et al. (2003)

The way a NLM computes the final probability distribution $f(w_{t-n+1},..,w_{t)=P(w_t|w_1,...,w_{t-1})}$ is also fundamentally the same. The output of the hidden layers, whether classic MLP or the BiLSTM of a modern NLM (see below) approximates the Maximum Likelihood Estimation of an n-gram model

(8) $$P(w_t|w_{t-1},\cdots,w_{t-n+1}) = \frac{count(w_{t-n+1},\cdots,w_{t-1},w_t)}{count(w_{t-n+1},\cdots,w_{t-1})}$$

via the softmax layer, which computes the following normalized probability.

(9) $$P(w_t|w_{t-1},\cdots,w_{t-n+1}) = \frac{\exp(e_{w_t h})}{\sum_{w_i \in V}\exp(e_{w_i}h)}$$

The unnormalized log-probability (logit) of the final fully connected output layer is $z = e_{w_t}h$, which is the inner product of the context vector *h* and computed embedding of *w_t*, the row of the weight matrix E of this output layer. This in practice is really a matrix-matrix multiplication of the weight matrix E of the output later, which contains the embeddings of all words in the vocabulary, and the batch outputs

H of the previous hidden layer(s) (for a given time step in the case of an RNN), i.e. $E \cdot H$. The softmax squashes this logit *z* into a probability for word $w_{t,}$ given the context

Using softmax, it is then possible to compute the loss of the LM, whose objective it is to maximize probability of the whole training data. This can be formalized by maximizing the averaged log probability of the data.

(10) $$L = \frac{1}{T} \log P(w_1, \cdots, w_T)$$

In practice, this is searching for the parameters θ which maximize the log likelihood (Bengio et al., 2001) :

(11) $$L = \frac{1}{T} \sum_{t=1}^{T} \log f(w_t, w_{t-1}, \cdots, w_{t-n+1}; \theta)$$

Recall that the function *f* is the model output (i.e. the conditional probability of a word as calculated by the softmax layer) and *n* is the number of previous time steps (window size) seen so far. (Bengio et al., 2001) identifies that the bottleneck of the network is, in fact, the softmax layer, since even with efficient matrix multiplication and powerful GPUs, the major computational cost is the inner product which needs to be computed for the whole vocabulary, often in the hundreds of thousands.
Because of this, LMs are notoriously hard to train, often requiring huge computational power and sometimes months of training time. This is one of the appeals of character LMs (explored in §2.4 below).

CHAPTER 2

Sequence taggers

2.1 LM-Fueled Sequence Taggers

The classical LM architecture just discussed can be implemented by MLPs or RNNs. Current RNN LMs, both LSTM and GRU, outperform *n*-gram LMs by virtue of their ability to encode long-range dependencies in their hidden states (Graves 2013) In this chapter, several recent NLM architectures and their applications to sequence labeling tasks are discussed.

2.2 TagLM, Peters et *al*. (2017)

Peters et al. (2017), leveraging a pretrained bidirectional LM, improved on the state of the art for several sequence labeling tasks, NER and Chunking. Their Language Model Augmented Sequence Tagger (TagLM) embeds an input sequence using a combination of pretrained LM embeddings, pretrained context insensitive word (token) embeddings, and in task character embeddings.

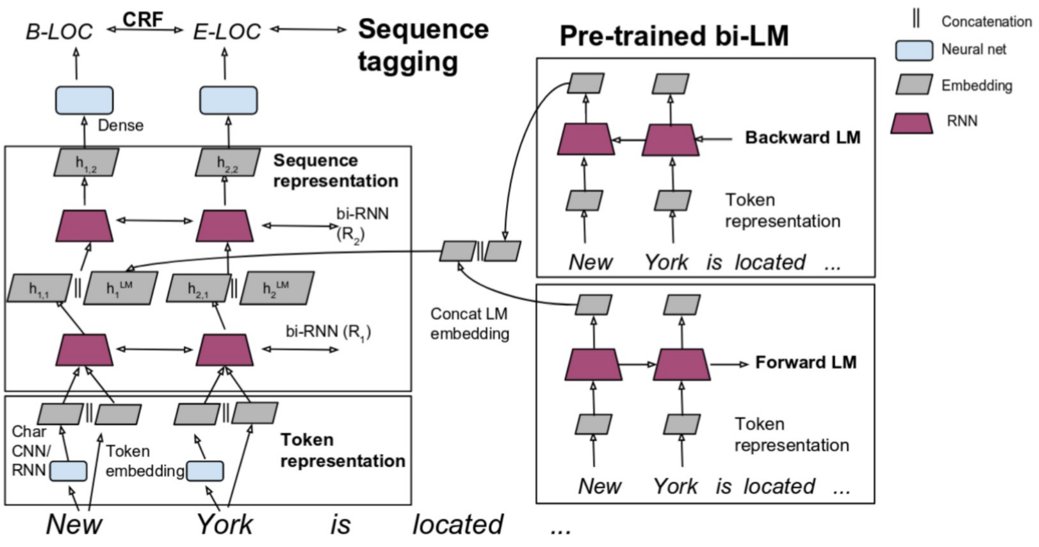


Figure 7: TagLM (Peters et al. 2017)

As is depicted above for the task of NER, the model contains several layers or components. In the embedding layer, embeddings over characters are obtained via a RNN or a Convolutional Neural Network (CNN). Using a character-level RNN for illustration, a character sequence for a word such as *cat* [‘c’, ‘a’, ‘t’] is fed to a RNN, usually bidirectional (BiRNN), which embeds the context to the left of each character (forward) and to the right (backward). A backward RNN is like the forward variant described above, but from right to left, encoding the word in reverse. The forward and backward outputs representing each word are then concatenated.

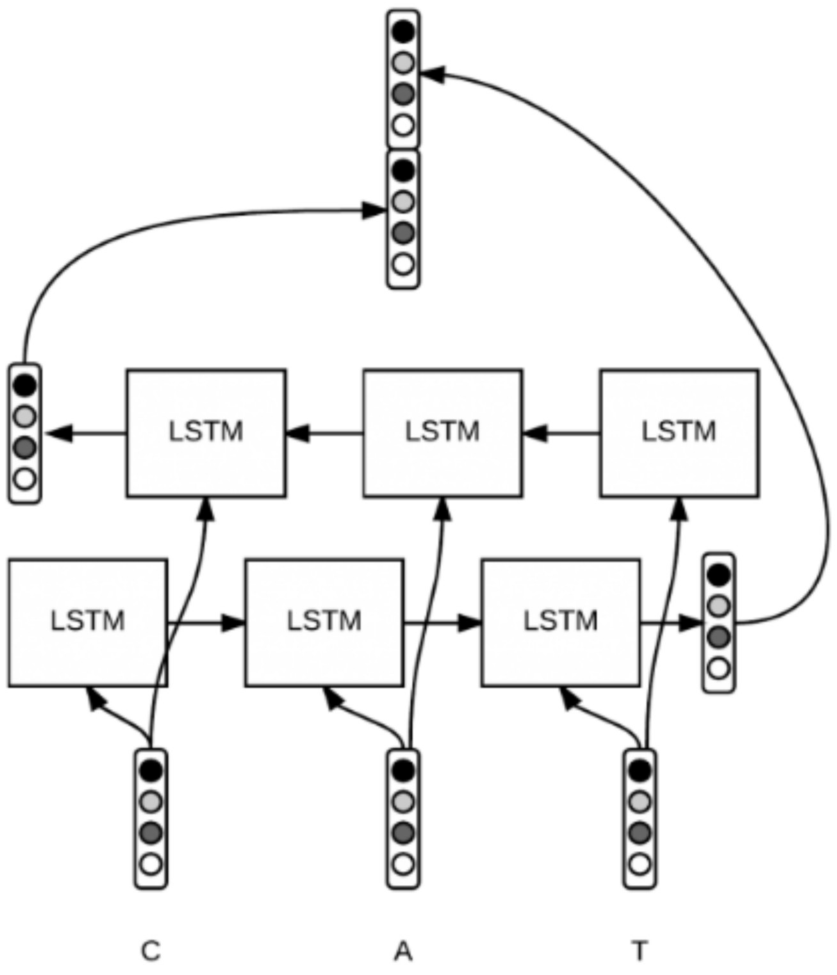


Figure 8: Character-level embeddings from an LSTM, from https://guillaumegenthial.github.io/sequence-tagging-with-tensorflow.html

These character-level embeddings are then concatenated with pretrained word embeddings such as word2vec. For instance, *York* is now represented by the embedding $h = [h_c; h_w]$.

The LM component of the model predicts the probability of the input sequence $w_1, ..., w_T$, using one or more layers of BiRNN, whether BiLSTM or BiGRU.

$$P(w_1, .., w_T) = \prod_{t=1}^{T} P(w_t | w_1, ..., w_{t-1}), \tag{12}$$

In the forward pass, the history of each token is embedded in the internal state which is output at each step t, until reaching the end of the sequence at which point $\vec{h}_t^{LM}$ is output as the final state representing the whole history. If this is the top layer, this is the forward embedding, and a prediction can be made for token w_{t+1}. Next, the process in repeated but in the opposite direction for the reverse sequence $w_T, ..., w_1$, yielding the backward embedding $h_t^{\leftarrow LM}$.

(13) $$P(w_1, .., w_T) = \prod_{t=1}^{T} P(w_t | w_{t+1}, ..., w_T)$$

These top layer representations are then concatenated $h_t^{LM} = \left[\vec{h}_t^{LM}; h_t^{\leftarrow LM}\right]$.

In the next step, these embeddings are combined with representations obtained from the sequence labeling task BiLSTMs. The authors chose to concatenate the LM with outputs from the first BiRNN layer, $h_{k,1}^{LM}$, which is fed to the second BiRNN. This stacked representation is as follows:

(14) $$h_{k,1}^{LM} = \left[\vec{h}_{k,1}^{LM}; h_{k,1}^{\leftarrow LM}; h_k^{LM}\right]$$

The outputs of this layer are input to a final dense layer, which is decoded by a CRF.

Analogous to the forward only LM, the objective of a biLM is to maximize the probabilities in each direction. The parameters for each BiLSTM LM are separate (θ_{fw}, θ_{bw}), but the remaining parameters are shared, e.g. those of the input and output, and softmax layers, here simplified to θ.

(15) $$\frac{1}{T} \sum_{t=1}^{T} \log \left(P(w_t | w_1, ..., w_{t-1}; \theta_{fw}; \theta\right) + P(w_t | w_{t+1}, ..., w_T; \theta_{bw}; \theta)\big)$$

The above model is evaluated using CoNLL 2003 NER and CoNLL 2000 Chunking tasks.

Parameter	Value
Character-level embedding (CLE) dimension	25
BiGRU X 2 state size for CLE	80
BiGRU X 2 state size for tagger	300
Dropout for each GRU input	.25

Table 1: Hyperparameters for NER

Parameter	Value
Character-level embedding (CLE) dimension	30
Number of CNN filters	30
CNN kernel size	3
Tagger BiLSTM X 2 state size	200
Dropout for CLE, each LSTM input and final layer	.50

Table 2: Hyperparameters for Chunking

For both of these tasks, the LM employed was a publicly released pretrained best model of Jozefowicz et *al*. (2016), trained on around 800,000 tokens from the One Billion Word Benchmark (Chelba et al., 2014), a data set designed for LMs, for three weeks on 32 GPUs. This LM achieved a test perplexity of 30.0. The hyperparameters of this LM, referred to as CNN-BIG-LSTM, are given below.

Parameter	Value
Number of CNN filters (character-level)	4096
CNN kernel size	?
BiLM layers	2
BiLSTM state size	8192
Projection dimension (BiLM)	1024
Dropout before & after every LSTM layer	?

Table 3: CNN-BIG-LSTM LM

Note that kernel size and dropout parameters were not published. The results of for these two tasks are also given below:

MODEL	NER	CHUNKING
PREVIOUS SOA	91.62	95.28
BASELINE (NO LM)	90.87	95.00
TAGLM	**91.93**	**96.37**

Table 4: Task results for TagLM

There are several task-independent results which are worth mentioning here. First as mentioned, it was found that the best results were obtained when the embeddings were concatenated after the first LSTM layer. Second, the addition of a backward LM boosts F1 scores .22-.27%. Third, the authors experimented with other sizes of BiLSTM for the LM and found that the best result were obtained from the largest model (CNN-BIG-LSTM), with gains of .26-.32 over a smaller LSTM-2048-512. Thus size matters. Fourth, the LM was trained on a smaller task-specific data set, which resulted in much higher perplexities, which reflects the necessity of seeing a wide range of contexts during training, i.e. RNN LMs learn composition functions (Peters et *al*. 2017). Another interesting experiment included removing the task specific RNN and just using the LM embeddings followed by a dense and CRF layer. This also resulted in inferior results, which confirms the importance of a supervised task specific architecture.

2.3 ELMo, Peters et al. (2018)

Embeddings from Language Models (ELMo) build on the just presented TagLM model of Peters et *al*. (2017). There are some differences which will now be explained. The ELMo model is also a biLM, with an objective the same as that of above, namely to optimize model parameters so as to maximize the probabilities of the data:

$$(16) \quad \frac{1}{T}\sum_{t=1}^{T} \log\big(P(w_t|w_1,...,w_{t-1};\theta_{fw};\theta) + P(w_t|w_{t+1},...,w_T;\theta_{bw};\theta)\big)$$

Like TagLM, the BiLM component of ELMo also follows the architecture of (Jozefowicz, Schuster, Wu, Com, & Brain, 2015), with the exception of modifications to allow training in both directions and the addition of a residual connection between LSTM layers. Unlike TagLM, which utilizes the CNN-BIG-LSTM, ELMo halves the size of embeddings and dimension of hidden units. The hyperparameters are given below

Parameter	Value
Number of CNN filters (character-level)	2048
CNN kernel size	?
BiLM layers	2
BiLSTM state size	4096
Projection dimension (BiLM)	512
Dropout before & after every LSTM layer	?

Table 5: ELMo BiLM

The smaller hyperparameters lead to a rise in perplexity to 39.7 (averaged in both directions), cf. 30.0 for the CNN-BIG-LSTM. However, this is justified as balancing performance and computational requirements. The model can also be fine-tuned with domain-specific data, leading to lower perplexity and better performance on downstream tasks.

In contrast to TagLM, the authors mention that some parameters of each direction of the LM are shared. For each token an L-layer BiLM computes two representations, one in the forward and another in the reverse direction. Since for layer j = 0, the token layer *x*, only one representation is computed, there are 2L + 1 total computations:

$$h_{k,j}^{LM} = \{x_k^{LM}, \vec{h}_{k,j}^{LM}, h_{k,j}^{\leftarrow LM} | j = 1, \cdots, L\} = \{h_{k,j}^{LM} | j = 1, \cdots, L\} \tag{17}$$

Note that the first member of the set here is the context-independent token embedding (e.g. pretrained word embedding or character-level embedding, or both). Given that there are two layers used in the model, there are thus three total representations per token. Recall that for TagLM, the top (final) layer of the BiLM

$h_{k,L}^{LM}$ is used as the embedding for downstream tasks. ELMo, however, uses a task-specific weighting of all layers.

(18) $ELMo_k^{task} = \gamma^{task} \sum_{j=0}^{L} s_k^{task} \mathbf{h}_{k,j}^{LM}$

In the equation above, s^{task} are softmax normalized weights and γ^{task} is for scaling the final embedding vector. These parameters can be used to improve the task model. The representations from all BiLM layers are fed to the task model, which learns a linear combination of them.

2.4 Flair, Akbik et *al.* (2018)

The character language model of Akbik et *al.* (2018), Flair, like TagLM and ELMo, is a BiLM, with the major difference that it is over characters rather than words. This model has the following properties:

1. It can be pretrained in an unsupervised fashion on large unlabeled corpora.
2. It yields contextualized embeddings (referred to as *contextual string embeddings*).
3. It models words in their context as sequences of characters.

Properties (1) and (2) are the same as those of a word LM such as ELMo. Property (3), however, is unique to a character LM, hence CharLM, and is perhaps the biggest benefit of using one over a word LM. When a word not in the training data is encountered, a normal LM, much like a pretrained embedding, is forced to assign the representation of an unknown symbol (<UNK>). Although it has been noted that CharLMs often perform somewhat worse than word LMs, reflected in higher perplexity (Graves 2014), we will see that Flair, in fact, performs better than LM-fueled sequence taggers such as ELMo.

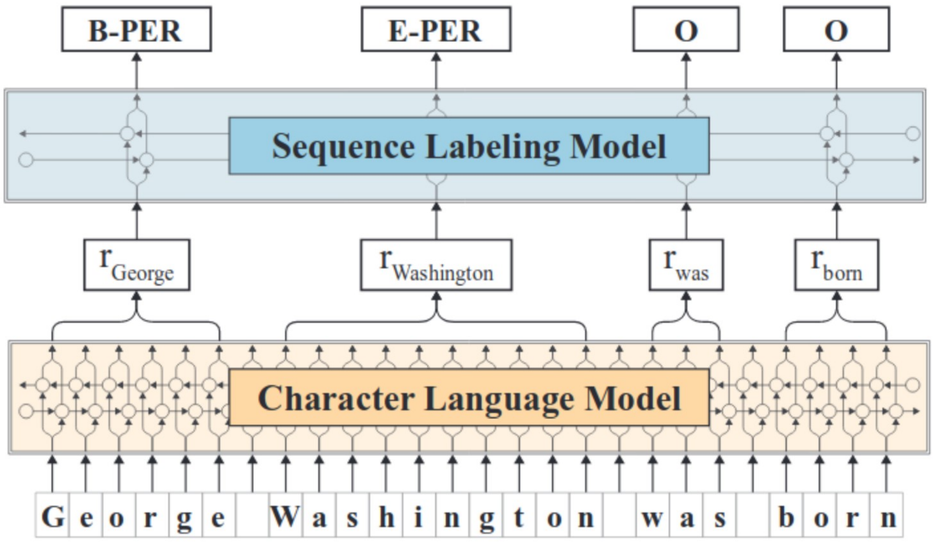


Figure 9: Character Language Model (Akibik et al. 2018)

There is a growing body of research demonstrating that neural models trained to predict the following character given a preceding character history internalize morphosyntactic and semantic knowledge (Akbik et *al.* 2018; Graves 2013). This is most surprising since these networks are trained without any grammatical data, or even explicit notion of word or sentence boundaries, yet are perfectly capable of generating grammatically correct sentences.

A CharLM works in an analogous fashion to a word LM, except that now the task is to predict the probability of an input character sequence $c_1, ..., c_T$, using one or more layers of BiRNN, whether BiLSTM or BiGRU.

(19) $$P(c_1, .., c_T) = \prod_{t=1}^{T} P(c_t | c_1, ..., c_{t-1}),$$

The forward LM internalizes the history of characters seen so far in its internal states, which are output at each step t, until reaching the end of the sequence. Like in a word LM, a prediction can be made for the next character at any point c_{t+1}. Next, we reverse the process for $c_T, ..., c_1$, yielding the backward states.

(20) $$P(w_1, .., w_T) = \prod_{t=1}^{T} P(c_t | c_{t+1}, ..., c_T)$$

In the case of the word model, recall that $\vec{h}_t^{LM}$ and $h_t^{\leftarrow LM}$represented the word embedding. In the case of a stream of characters with no word markers, the initial and final states need to be extracted, but can be done easily with offsets, analogously to extracting a word from a sentence embedded in a word LM.

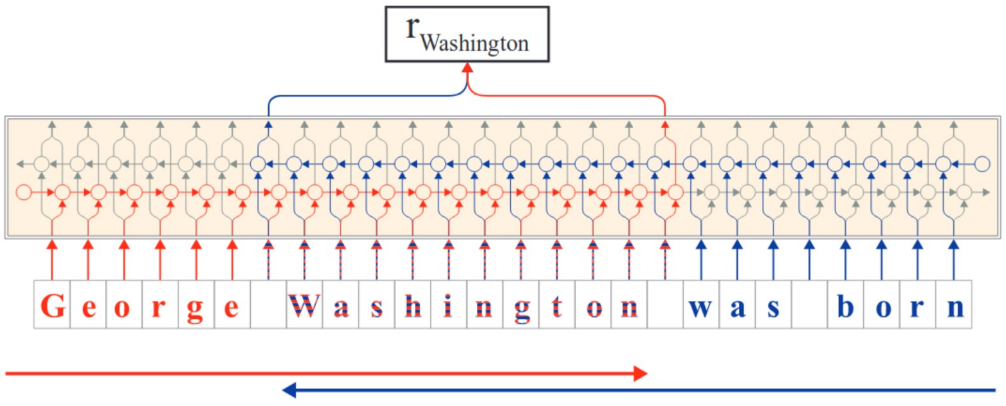


Figure 10: Character LM embeddings (Akbik et al. 2018)

As in ELMo, these word representations are then concatenated $h_t^{LM} = [\vec{h}_t^{LM}; h_t^{\leftarrow LM}]$. These embeddings are computed not just on the characters of a word, but on a window of characters surrounding the word and thus reflect much larger context which captures all of the meanings a word may have (polysemy).

The sequence labeling architecture of the Flair model is similar to other state-of-the-art systems like ELMo, making use of a BiLSTM and CRF decoder. In contrast to ELMo, there is only one LSTM layer, and the CharLM embeddings can thus only be concatenated as input to this layer. For instance, one can stack the CharLM embedding with a pretrained context insensitive embedding such as word2vec.

(21) $$w_i = [w_i^{CharLM}; w_i^{Pretrained}]$$

These word embeddings are next fed to the BiLSTM where forward and backward states for each token are concatenated, representing a longer range sentential context around each token w_t.

(22) $$r_t = \left[\vec{r}_t ; r_t^{\leftarrow}\right]$$

This representation can then be input to a final softmax layer for a prediction, similar to the classical LM described above.

(23) $$P(y_i = j | r_i) = softmax(\mathbf{r_i})[j]$$

Recall from §1.1.1 that in CRFs weight λ_js are associated with feature functions.

(24) $$P(\mathbf{y}|\mathbf{x}) = \frac{exp[\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(y_i, y_{i-1}, x_i)]}{\sum_y exp[\sum_{j=1}^{m} \sum_{i=1}^{n} \lambda_j f_j(y_i, y_{i-1}, x_i)]}$$

Applying this to a neural sequence model, we have

(25) $$P(\mathbf{y_1}, \cdots, \mathbf{y_n} | \mathbf{r_1}, \cdots, \mathbf{r_n}) \propto \prod_{i=1}^{n} \lambda_i(y_{i-1}, y_i, \mathbf{r_i}), \quad \text{where}$$
$$\lambda_i(y', y, \mathbf{r}) = \exp\left(\mathbf{W_{y',y}} \mathbf{r} + \mathbf{b_{y',y}}\right)$$

The best Flair model performance on downstream tasks is achieved when CharLM embeddings are trained and stacked with pretrained word embeddings, and potentially in task character-level embeddings as well, and then fed to a sequence model with a CRF decoder. The Flair language model for all published tasks was trained with the following hyperparameters:

Parameter	Value
Character window size	250
Learning rate before annealing	20
BiLM layers	1
BiLSTM state size	2048
Gradient clipping	.25
Dropout	.5
Batch size	100

Table 6: Flair CharLM

For sequencing tasks, the following hyperparameters were used:

Parameter	Value
Character window size	250
BiLM layers	1
BiLSTM state size	256
Variational dropout	.5
Learning rate model selection	{.01, .05, .1}
Batch size model selection	{8, 16, 32}

Table 7: Flair sequence model

The results for NER and Chunking in English, for the Flair model leveraging pretrained word embeddings (PWE) and character-level embeddings is given below, along with the results for ELMo and TagLM.

ARCHITECTURE	NER	CHUNKING
	F1	F1
FLAIR$_{\text{PWE+CLE}}$	**93.09**	**96.72**
ELMO	92.22	-------
TAGLM	97.42	96.37

Table 8: Results for Flair vis-à-vis Peters et al. (2017, 2018)

These state-of-the-art results are what led me to choose to train a Flair CharLM for the Portuguese sequence labeling experiments which are explored in the next chapter.

CHAPTER 3

Models

In this section, I describe the experimental utilized for the chosen sequence labeling tasks (PoS tagging, NER, and VMWE identification) outlined in chapters 4, 5, and 6.

3.1 Character Language Models (CharLMs)

To obtain effective character language models (CharLMs) it is necessary to train for around two or more weeks with sufficient data. I chose to train Flair language models because of the phenomenal results reported by Akbik et al. (2018) for several sequence labeling tasks (§2,4), combined with the relative ease of training, both in terms of computational resources and time. I was interested in training LMs for Portuguese because of my passion for and knowledge of this language for which resources are lacking. The authors of Flair also indicated to me that contributions for this language were in demand and welcomed[3]. In the next section, I describe the training of these CharLMs, as well as the sequence model I implemented in detail.

3.1.1 Training data

CharLMs were trained on .9B words of Portuguese CommonCrawl text[4]. Chelba et al. (2014) describes the preparation of benchmark training data for language models (One Billion Word Benchmark[5]). The standards established in this work were followed whenever possible.

(26) Preprocessing of CommonCrawl data

3.Models contributed here: https://github.com/zalandoresearch/flair/blob/master/resources/docs
4. https://www.statmt.org/ngrams/
5. http://www.statmt.org/lm-benchmark/

Normalization and tokenization was performed

Duplication was performed[6]

Data was split into 100 disjoint partitions (shards)

1% of the data was chosen as heldout

Heldout was split equally into heldout and test sets[7]

In addition, since much of the CommonCrawl data were messy, containing short or incomplete sentences, sentences less than 3 words or 20 characters and greater than 100 words were removed, the latter because of concerns of required memory for training. Tokenization was performed with a Portuguese regular expression tokenizer which I implemented in Python. However, this step was probably unnecessary because the CharLM treats the text as a string of tokens.

3.1.2 Training

Both forward and backward CharLMs were trained for a little over two weeks, reaching development perplexity of 2.78 and 2.81 respectively. The following hyperparameters, recommended in Akbik et *al.* (2018), were used:

Parameter	Value
Character window size	250
Learning rate before annealing	20
BiLM layers	1
BiLSTM state size	2048
Gradient clipping	.25
Dropout	.5
Batch size	100

Table 9: Hyperparameters for CharLMs

6.The CommonCrawl data used was deduped.

7. This differs from the specified split of heldout into 50 partitions, taking one for test (2%) of total data.

3.2 Sequence Model

Although Flair includes an out-of-the-box sequence tagger implemented in PyTorch[8], I chose to implement my own in TensorFlow[9]. This decision was based in part on my already having implemented an—albeit simple—sequence tagger in TensorFlow. Furthermore, I wanted to extend the features of the Flair tagger, a task which I deemed would be easier for me in TensorFlow than in PyTorch. The following features either not present or not readily adaptable (character embeddings) in Flair were added in my implementation:

(27) Features implemented in addition to those present in Flair

1. Trainable in task word embeddings
2. Trainable in task lemma embeddings
3. Trainable in task PoS embeddings
4. Trainable in task character-level embeddings
5. Several types of dropout applicable in different layers
6. Batch normalization applicable in different layers
7. Verbal Multiword Expression (VMWE) specific evaluation using BIOSE tags

Often it is useful to train word embeddings specific to the task at hand. This is particularly useful when the overlap of pretrained vocabulary and in task vocabulary is far from perfect. Trainable lemmas and PoS embeddings are useful when one has such training data at one's disposable. Lemmas help to capture semantic meaning often lost in highly inflected languages where the total meaning of a word is often distributed over dozens of related forms. PoS features encapsulate morphological meaning which for some tasks like NER or VMWE identification are extremely useful. Like any other embedding, all of these in task embeddings are updated during training and can be concatenated with other embeddings such as pretrained word and CharLM embeddings. In the next section, the experimental setups are described.

8. https://github.com/zalandoresearch/flair
9. https://github.com/ericlief/sequence-tagger.git

3.3 Experiments

In order to assess the performance of the variety of embeddings at one's disposal for a particular sequence labeling task, I conducted a series of task-specific experiments. It was also my aim to gain a better understanding of the inherent semantics of such embeddings.

3.3.1 Tasks

I evaluated three classic sequence labeling tasks using available Portuguese datasets, namely Part of Speech (PoS) tagging using the MacMorpho corpus and CoNLL-2003 evaluation scripts , Named Entity Recognition (NER) using the HAREM corpus and CoNLL-2003 evaluation scripts, and Verbal Multiword Expression (VMWE) identification using the Portuguese PARSEME data and evaluation scripts. PoS tagging was chosen in order to evaluate the effectiveness of the embeddings for shallow syntactic tasks, and NER and VMWE in order to assess the embeddings for shallow semantic tasks (Akibik et al. 2018).

3.3.2 Experimental Setup

For baselines, I chose to evaluate setups utilizing only pretrained word embeddings[10]. These are essentially reimplementations of earlier state of the art approaches within a BiLSTM-CRF sequence labeling architecture (Akibik et al. 2018). Experimental evaluation of these setups will determine how well the CharLM and other in task embeddings perform over earlier approaches.

The baseline setups are:

HUANG: A classic BiLSTM-CRF setup with pretrained word embedding —a reimplementation of Huang et al. (2015)

10. 300-dim fastText embeddings were used for all experiments https://s3-us-west-1.amazonaws.com/fasttext-vectors/word-vectors-v2/cc.pt.300.vec.gz

LAMPLE: A hierarchical BiLSTM-CRF setup with pretrained word embeddings and in task character-level embeddings—a reimplementation of Lample et al. (2016)

The experimental setups are:

FLAIR: A BiLSTM-CRF setup with pretrained character language model (CharLM) embeddings—a reimplementation of Akbik et al. (2018)

FLAIR$_{+PWE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and pretrained word embeddings (PWE)

FLAIR$_{+CLE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and in task character-level embeddings (CLE)

FLAIR$_{+WE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and in task word embeddings (WE)

FLAIR$_{+PWE+CLE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and pretrained word embeddings (PWE) and in task character-level embeddings (CLE)

FLAIR$_{+PWE+WE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and both pretrained and in task word embeddings

FLAIR$_{+WE+CLE}$: A BiLSTM-CRF setup with with pretrained character language model (CharLM) and both in task word (WE) and in task character-level embeddings (CLE)

FLAIR$_{+ALL}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and both in task word (WE) and in task character character-level embeddings (CLE)

There are three additional experimental setups for the task of Verbal Multiword Expression (VMWE) identification:

FLAIR$_{+ALL+LEMMA}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and in task word (WE), character character-level (CLE), and lemma (LEMMA) embeddings

FLAIR$_{+ALL+TAG}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and in task word (WE), character character-level (CLE), and tag (TAG) embeddings

FLAIR$_{+ALL+LEMMA+TAG}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and in task word (WE), character-level (CLE), lemma (LEMMA), and tag (TAG) embeddings.

In the next chapter, we explore the task of PoS tagging.

CHAPTER 4

Part of Speech (PoS) Tagging

Part of Speech tagging (PoS) consists in classifying a token with respect to a morphosyntactic category such as Noun, Verb, Adposition, etc. (i.e. part of speech). Although not so difficult and interesting a task in itself, PoS tagging is an important lower-level task necessary for higher-ones such as chunking and parsing. Although not required for other tasks such as Named Entity Recognition (NER) and MultiWord Expressions (MWEs), as we will see in the next section, training with PoS features usually improves performance. Though there are universal grammatical categories such as Noun, Verb, or Adjective, most corpora are annotated with finer-grained language-specific grammatical categories. As a consequence, there is quite a degree of divergence in both the types of tags found and the size of the tagsets. For instance, the Prague Dependency Treebank (PDT) tagset for Czech, a highly inflected East Slavic language, contains 1547 distinct tags, while the Penn Treebank English tagset contains only 36.

Portuguese possesses a rich morphology, but like all Romance languages its verbs inflect more than its nouns. The most popular tagsets for Portuguese only identify Verb or Verbal Auxiliary—and the handful of nominal tags, unlike the PDT tagset, do not reflect gender or number. We are thus left with a diminished and somewhat biased set of tags which seems to be modeled on lesser inflected English.

4.1 Data

For the PoS task, although there are several available corpora annotated with PoS tags (e.g. Bosque, see Afonso et *al.* 2002), I chose Mac-Morpho (Aluísio 2003)[11], which is currently the largest one with the most reported task results. Mac-Morpho is around one million words and is composed of 109 files from the newspaper *Folha de São Paulo*, divided into 10 sections such as politics, sports, agriculture, etc. There are two three versions of Mac-Morpho and it appears that most of the recent

11. http://nilc.icmc.usp.br/macmorpho/

published results use the first version (e.g. dos Santos et *al*. 2014). For comparability, I chose to use this version. Below are some corpus statistics

	SENTENCES	TOKENS
TRAIN	42,022	957,439
DEV	2,211	50,232
TEST	9,141	213,794
	53,374	**1,221,465**

Table 10: Mac-Morpho v.1

At somewhat over 1.2 million words, the corpus is one of the largest for PoS tagging. Its format is simple, providing only form and tag and no other features such as lemma for training. Its tagset contains 41 tags, 22 PoS tags and 19 punctuation tags. In later versions of the corpus, all punctuation is subsumed under the PUN tag. Below is an sample from the data set.

(28) Mac-Morpho (v. 1) format
"_" Ao_PREP lançar_V as_ART sementes_N em_PREP a_ART terra_N o_ART produtor_N já_ADV deve_VAUX ter_V em_PREP mente_N a_ART etapa_N de_PREP a_ART colheita_N ._.

4.2 Evaluation

Following dos Santos et al. (2014), the CONLL-2003 evaluation script was used to evaluate the NER task. Accuracy is the most used metric for PoS tagging. Since every token must receive a tag and all tags are included in the calculation, evaluation is thus per token, rather than per entity, as we will see for the tasks of NER and MWE identification. Note that true positive (tp) are correctly identified tags, true negative (tn) are correctly not identified tags, false positive (fp) are misidentified tags, and (fn) tags missed that should have been identified.

(29) Evaluation metric

Accuracy is the number of correct tags (tp and tn) divided by the total tagged tokens (tp + tn + fp + fn):

$$Acc = tp + tn \div (tp + tn + fp + fn)$$

$$Acc = correct\, tags \div total\, tags$$

4.3 Recent Neural models for Portuguese PoS tagging

In this section, we review the model which advanced the state of the art in PoS tagging for Portuguese (dos Santos and Zadrozny 2014) using the Mac-Morpho corpus.

4.3.1 CharWNN, dos Santos and Zadrozny (2014)

The deep neural network of dos Santos and Zadrozny (2014) is based on Collobert et *al.*'s (2011) Window Approach Network. This neural architecture was one of the first of its kind to automate the feature engineering process, instead of relying entirely on hand-crafted features. In each layer of the model, a different set of features is extracted[12]. One of the novelties of this type of network is that it is one of the earlier approaches to use the concept of window for sequence labeling. The context of a word is represented by concatenating surrounding words to the input. Tag prediction thus takes into account not just the input token, but also the words around it, its context. This is summarized below.

12. Some variants of this network allow the manual introduction of features such as capitalization and suffix information.

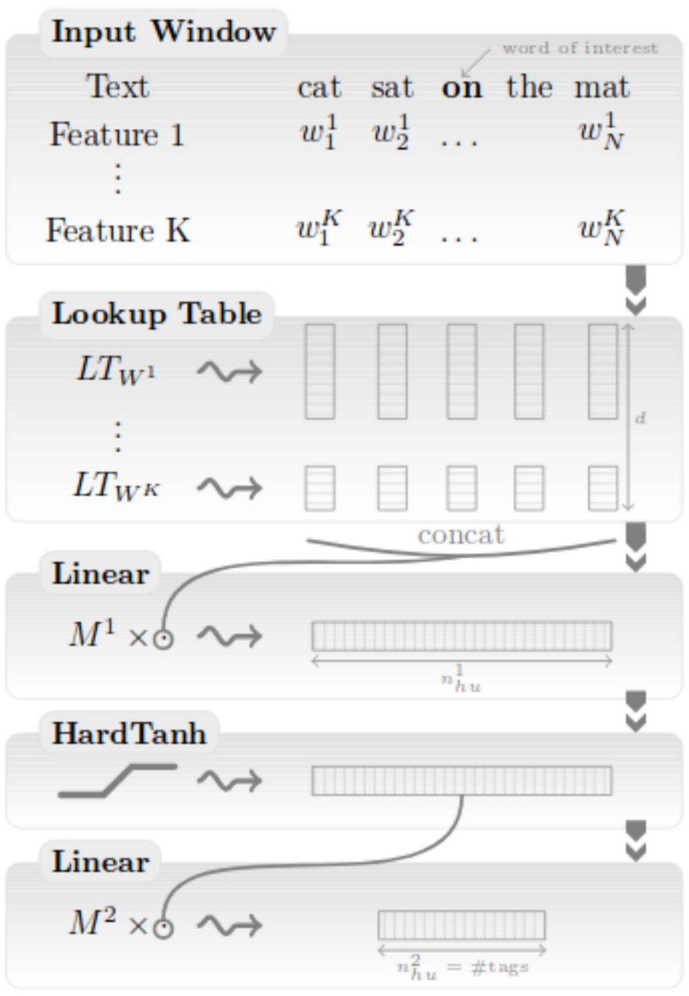


Figure 11: Window Approach Network, from Collobert et al. 2011

As figure 12 illustrates, the network consists of several layers: embedding layer (lookup table), classification layer (two linear layers with a HardTanh sandwiched in between, and a decoding layer (viterbi). A lookup table is used to embed the words in the sentence, with the central word in the window being the one we wish to classify (the word of interest). These words are concatenated into one long vector, which is then passed to the remaining layers for classification.

Dos Santos et *al*. (2014) modifies this architecture slightly, adding a character-level representation. Character-level embeddings are obtained with a convolutional neural network (CNN). CNNs (LeCun 1998) are a special type of network for processing data with a grid-like topology such as images, time-series data, etc. (Goodfellow et al. 2016). Like RNNs, CNNs are especially good at contextualizing data, as convolution is a linear operation like matrix multiplication that maps surrounding input features to an output or feature map. Usually, in another layer, a pooling function replaces output at one location with a summary statistic of nearby output. Max pooling and average pooling are the most frequent functions.

In NLP, since sequences are normally 1-D, or 2-D when dealing with embeddings, 1-D convolution is applied over each input. We can visualize this below in Figure 13 with the sentence *I like this movie very much!* The convolution will be applied over the 2-D sentence matrix using 3 filter (or kernel) sizes, 2 x 2, 2 x 3, and 2 x 4, with 2 of each filter, yielding 2 feature maps for each size. The way this works is that the filter slides over the sentence from start to finish, and an element-wise matrix multiplication (convolution) is applied in each region, across all channels or dimensions, yielding a scalar output or map. Next, for each map size, max pooling takes the maximum value, thus normalizing the dimensionality of outputs (here size two or the number of filters per size). Finally, the outputs are concatenated and this vector is passed to the next layer, here a softmax layer with which the sentence is classified, for instance as being of positive or negative sentiment (polarity).

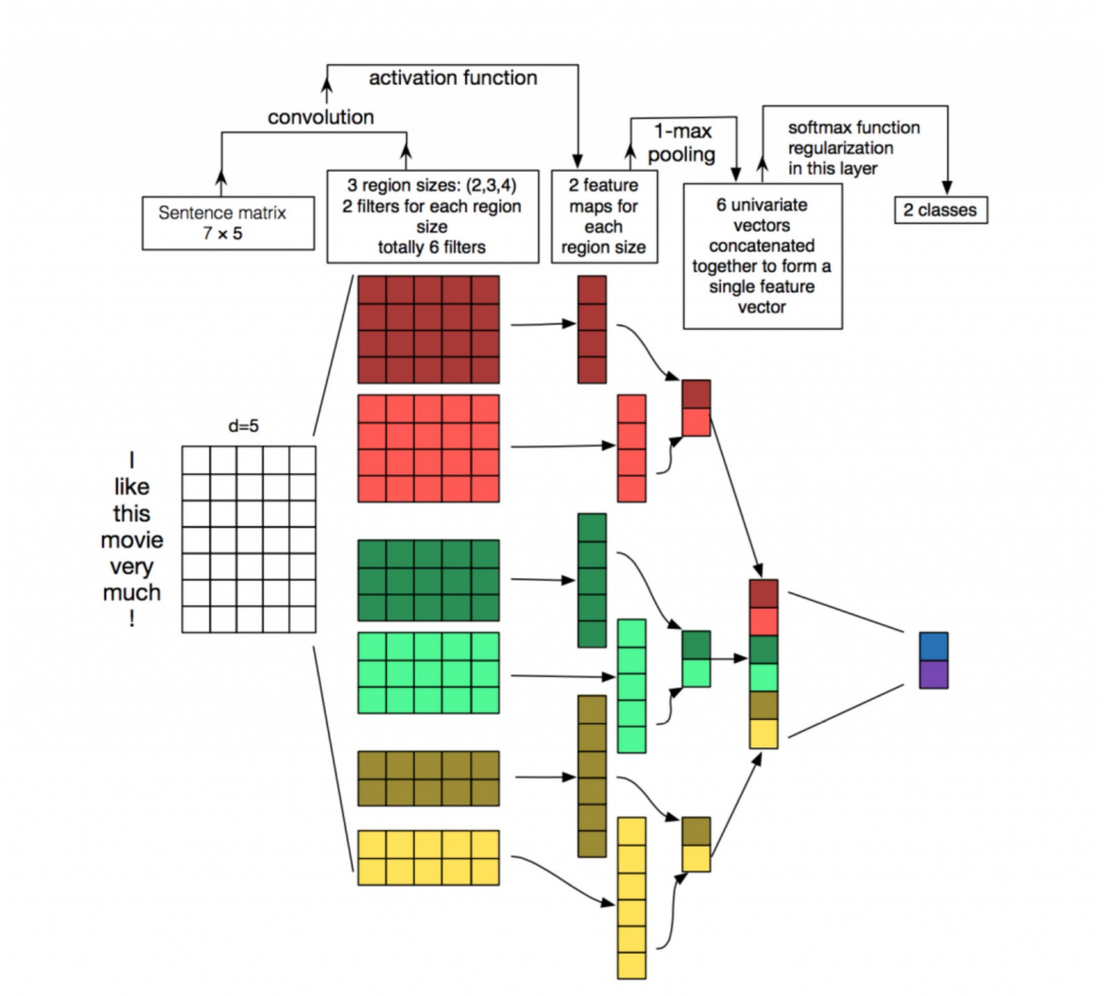


Figure 12: A CNN, from Zhang et al. (2015)

CNNs are quite effective for forming context-sensitive character-level embeddings, in an analogous fashion as that depicted above, but over characters rather than words. After getting a representation for all unique words in a mini-batch, these features can be used as a lookup table that can be indexed by the ids of words in the batch sentences, yielding character embeddings.

For the task of PoS tagging, dos Santos et *al*. (2014) uses the Mac-Morpho corpus (v. 1). Below are the hyperparameters of the network.

Parameter	Value
Word embedding dimension	100
Character embedding dimension	10
Word and character content window size	5
CNN filters	50
Hidden units	300
Learning rate	.0075

Table 11: CharWNN sequence tagging model, dos Santos et *al*. (2014)

Dos Santos et *al*. (2014) achieved state of the art results for Portuguese, slightly more than earlier results which did not utilize neural networks and which relied on feature engineering. Below are the results of CharWNN and some of the different architectures also experimented with which utilize only word embeddings (WNN) in combination with manually added features such as capitalization and suffixes.

ARCHITECTURE	FEATURES	ACCURACY
CHARWNN	–	**97.47**
WNN	–	96.19
WNN	CAPS+SUF3	97.42
WNN	CAPS	97.27
WNN	SUF3	86.35

Table 12: Results (dos Santos et al. 2014)

The addition of character embeddings improves performance. Dos Santos et *al*. (2014) reports an error reduction of 58% in out of vocabulary (OOV) words. This is because character-level representations can be constructed for words not seen in the training data. The addition of capitalization and suffix features, however, shows mixed results[13].

We have seen how successful deep neural models perform in sequence tagging tasks such as PoS tagging. Incorporating word embeddings in combination with character-level word embeddings such as dos Santos et *al*. (2014) leads to state of the art results. This suggests that character features reduce the OOV error rate are better at representing morphological or subword features than word embeddings

13. It is not clear to me why the addition of suffix features by themselves worsens performance, while the combination of both features slightly improves accuracy.

alone are. In the next section, we will see how the use of a character LM (CharLM) leads to state-of-the-art results for PoS tagging.

4.4 Model

The sequence labeling model introduced in §3.2, leveraging character language models (CharLMs), was utilized for the PoS tagging task. Below I describe the tuning of the hyperparameters and several distinct combinations of input representations (features) that were evaluated.

4.4.1 Setups

The baseline and experimental setups described in §3.3.2 (presented below for convenience) were applied to the PoS tagging task.

The baseline setups are:

HUANG: A classic BiLSTM-CRF setup with pretrained word embedding—a reimplementation of Huang et al. (2015)

LAMPLE: A hierarchical BiLSTM-CRF setup with pretrained word embeddings and in task character-level embeddings—a reimplementation of Lample et al. (2016)

The experimental setups are:

FLAIR: A BiLSTM-CRF setup with pretrained character language model (CharLM) embeddings—a reimplementation of Akbik et al. (2018)

FLAIR$_{+PWE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and pretrained word embeddings (PWE)

FLAIR$_{+CLE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and in task character-level embeddings (CLE)

FLAIR$_{+WE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and in task word embeddings (WE)

FLAIR$_{+PWE+CLE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and pretrained word embeddings (PWE) and in task character-level embeddings (CLE)

FLAIR$_{+PWE+WE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and both pretrained and in task word embeddings

FLAIR$_{+WE+CLE}$: A BiLSTM-CRF setup with with pretrained character language model (CharLM) and both in task word (WE) and in task character-level embeddings (CLE)

FLAIR$_{+ALL}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and both in task word (WE) and in task character character-level embeddings (CLE)

The two baseline setups HUANG and LAMPLE are essentially reimplementations of Huang and Yu (2015) and Lample et al. (2016) respectively. These two setups do not utilize a language model and serve to establish the usefulness of adding character LM embeddings. The remaining setups are all combinations of different embeddings with the basic FLAIR model and in a similar fashion will give us insight into the gain associated with adding each of these features to the basic configuration.

4.4.2 Hyperparameters

I experimented with character embeddings extracted from an RNN (BiLSTM) as well as from a CNN, like that utilized by dos Santos et *al*. (2014). A variety of different setups with varying number of filters and kernel sizes were experimented with. As it turns out, the CNN performs on a par with the RNN for character-level embeddings. Furthermore, batch normalization versus dropout was tested with several different configurations for the CNN:

(30) CNN block configurations
1. Conv [no activation] – Norm – Pooling
2. Conv [+ ReLU] – Pooling – Norm
3. Conv – Norm – ReLU – Pooling

Configuration (1) consistently outperformed the others and was thus selected as the default for the the setups involving character embeddings for the tasks carried out for this thesis. Table 13 below show the results for these parameters with the FLAIR$_{+ALL}$ setup, vis-à-vis the same model with RNN character embeddings, with hidden state size of 256. Note that the notation used is *#filters-max_kernel_size,* e.g. 300-3 denotes 300 filters, with kernels of size 2 and 3. All of the results are quite similar and it appears that minimally 200 filters with kernel sizes 2, 3, and possibly 4 are adequate to extract features at the character level for words.

MODEL	**ACCURACY**
200-3	97.46
200-4	**97.49**
300-3	**97.49**
300-4	97.48
400-3	97.46
400-4	**97.49**
500-3	97.46
500-4	97.44
RNN char. emb.	**97.49**

Table 13: CNN versus RNN for Character Embeddings (PoS Tagging)

Given the equal performance, I selected the RNN (BiLSTM) character embeddings for this task. 256 hidden units were found to be optimal for the embeddings. A number of experiments were next run with differing batch normalization and dropout schemes, and the best results were obtained with only dropout before the character-level LSTM and dropout using a different mask (non-variational) at each time step of the sentence-level LSTM[14]. Dropping out at other layers (input, output) yielded worse results. Batch normalization was also found to be most beneficial when applied before the sentence-level LSTM. The reason dropout was advantageous for the in task character embeddings was most likely because character embeddings in combination with the CharLM were somewhat redundant and applying dropout hindered overfitting and allowed the model to generalize more within the input representation. Varying RNN sizes were tested, and 512 hidden units was found to be optimal. These hyperparameters are summarized in Table 14 below.

PARAMETER	VALUE
Pretrained fastText word embedding dimension	300
In task word embedding dimension	512
Character BiLSTM state size	256
BiLSTM state size	512
Optimizer	SGD
Gradient clipping	.25
Dropout before character BiLSTM	.5
Locked dropout BiLSTMs	.5
batch normalization before sentence-level BiLSTM	True
Initial learning rate	.1
Annealing rate	.5
Patience	5
Batch size	32
Epochs	50

Table 14: Hyperparameters of PoS sequence tagger

4.5 Results

The results for the baselines, setups, and best published result for the PoS task are summarized below in Table 14.

14. In TensorFlow DropoutWrapper was used, with variational set to false. Using the recurrent variational options consistently degraded performance.

MODEL	**ACCURACY**
baselines	
HUANG	85.88
LAMPLE	89.15
FLAIR	97.20
FLAIR$_{+PWE}$	97.24
FLAIR$_{+CLE}$	97.32
FLAIR$_{+WE}$	97.44
FLAIR$_{+PWE+CLE}$	97.08
FLAIR$_{+PWE+WE}$	97.30
FLAIR$_{+WE+CLE}$	97.10
FLAIR$_{+ALL}$	**97.49**
best published	
dos Santos et al. (2014)	97.47

Table 15: Model Results (PoS Tagging)

The best model is FLAIR$_{+ALL}$, which surpasses dos Santos et al. (2014) by .02 pp. (.8% error reduction). It is not clear to the author why higher gains were not observed here. As mentioned above, numerous hyperparameters were tested and all led to results capped at 97.49. In addition to CNN in task character-level embeddings, the window-based model of dos Santos et al. (2014) utilized pretrained embeddings and in task word and character windows, perhaps approximating a context-sensitive CharLM with added in task character embeddings, i.e. FLAIR$_{+PWE+CLE}$.

In terms of the other setups, it is interesting that adding in task word embeddings (FLAIR$_{+WE}$) shows performance almost as good as the best full ensemble. When either pretrained word or in task character embeddings are added to the base setup (FLAIR$_{+PWE,}$ FLAIR$_{+CLE}$), slight gains are observed. Of all embeddings, the in task word embeddings fair the best. This is because we have plenty of training data and learning task-specific semantic features is not an issue. For the opposite trend, see the NER results (§5.6.3) However, performance is worse than FLAIR$_{+WE}$ when either one is added in combination with in task word embeddings (FLAIR$_{+PWE+WE}$, FLAIR$_{+WE+CLE}$).

There may be some degree of redundancy here, as the pretrained CharLM and in task word embeddings may effectively represent the word and character-level features necessary for this task. In other words, when more information is added, these extraneous features confound learning and lead to slight overfitting. To remedy this dropout was applied before the character BiLSTM.

The performance of the baseline setups underscores the gains achieved by adding CharLM embeddings. This is most likely due to the shallow morphosyntactic features (i.e. suffixes), as well as orthographic features such as capitalization, important for PoS tagging, which are encoded in the character-based LM. The high margin in the performance of FLAIR, with context-sensitive embeddings versus the LAMPLE reimplementation, with context-insensitive in task character-level embeddings (74% error reduction), also demonstrates the importance of context in this task, i.e. the same word form may have differing PoS tags in different contexts, cf. English *miss* (Noun) vs. vs. *miss* (Verb).

4.6 Conclusions

In this chapter, we have explored the task of PoS tagging and seen how the use of a character language model (CharLM) greatly improves the results for this task. When a pretrained CharLM is used in conjunction with in task word embeddings more gains are observed. It is believed that this is because the size of the data is sufficient for training these task-specific semantic representations. When all of the embeddings are combined in the full setup, even greater gains are achieved (F-score of 97.49), surpassing the published state of the art. This underscores the importance of utilizing both precomputed contextualized and non-contextualized word representations in an ensemble with task-specific word features.

CHAPTER 5

Named Entity Recognition (NER)

The task of named entity recognition (NER) is concerned with finding named entity mentions in unstructured text and classifying them into predefined categories such as person, location, or organization. NER has many applications in machine translation, information retrieval, question answering, etc. Traditionally rule-based approaches were quite popular. Like most rule-based approaches the process was time consuming. In the last ten to fifteen years, machine learning approaches have been quite successful and for the most part supplanted rule-based approaches. Early ML NER models included Hidden Markov Models (Zhao 2004; Todorovic et *al* 2008). In the last few years deep learning models have been extremely successful (Akbik et *al*. 2018, Peters et *al*. 2018).

Since the MUC-7 (Chinchor 1998) and CoNLL-2002 (Sang & De Meulder, 2003) shared tasks and the 2004 HAREM contest for Portuguese (dos Santos et al. 2006), NER has received much attention. In this chapter, we will first examine the Portuguese HAREM data set. Next, the most prominent neural approach to this task (dos Santos and Guimarães 2015). Finally, I present experimental setups and hyperparameters used for this task.

5.1 Toward a Definition of Named Entity

The term named entity was first introduced in the MUC-6 conference, whose goal it was to define PERSON, ORGANIZATION and LOCATION textual mentions (Grishman et al. 1996). In addition to these three classes for named entity expressions (ENAMEX), there were also numerical expressions, e.g time, money and dates (NUMEX). The CoNLL-2003 and CoNLL-2003 shared tasks brought further recognition to NER. These tasks identify four classes: PERSON, ORGANIZATION, LOCATION, MISCELLANEOUS.

Named entities are sometimes loosely referred to as proper nouns, which in philosophical circles are often defined as *rigid designators* (Kripke 1971). In Kripke’s theory, a proper noun refers to the same referent in every possible world, whereas a description in another world or reality could potentially refer to some other object. For instance, the description *the oil company founded by John D. Rockefeller in 1870* originally referred to Standard Oil. Whereas the description could plausibly refer to another object in another world—and even in this world to any one of the 34 entities that the company was broken up into by the U.S. Supreme Court (ExxonMobil, Chevron, etc.) —the proper noun Standard Oil can refer only to the original organization and is thereby rigid. In some cases, however, some entities are not strictly rigid. For instance, the President of the United States could refer to more than one entity (Domingues-Fernandes 2018).

The task of NER is sometimes not as easy as it may seem. Consider the following example.

(31) *[Paris Hilton]$_{PERS}$ flew to [Paris]$_{LOC}$ on the [25th of August]$_{DATE}$.*

The noun Paris can refer to an individual or a place. If we wish to automate this task, it is clear that the whole context as well as perhaps other features such as PoS may be necessary to disambiguate these named entities. However, just as we saw with PoS tagging, the tagsets for NER are also often task or language-specific. Dates and numbers, which are not usually considered proper nouns, are sometimes considered entities because they may in fact be rigid, e.g. the 25th of August of a given year. However, the year is above implied.

Although there is much task-specific variation as regards tagsets, the annotation scheme used to identify the start and end of a named entity sequence is usually BIO (Beginning, Inside, Outside) and less commonly BIOSE (Beginning, Inside, Outside, Single, Ending). Using the BIO scheme, again for the previous sentence, we have:

(32)	*Paris*	*Hilton*	*flew*	*to*	*Paris*	*on*	*the*
	B-PERS	I-PERS	O	O	B-LOC	O	O
	25th	*of*	*August*		*.*		
	B-DATE	*I-DATE*	*I-DATE*		*O*		

The outside (O) tag guarantees that all input words have a label, allowing the entire sentence to be processed by a neural network. The classification task in an abstract sense here is thus one of first identifying entity ~ non-entity (O) and then classifying the entity.

5.2 Data

There are few Portuguese NER datasets. The largest corpus, produced originally for the HAREM contest (dos Santos et al. 2006), is quite small, containing a little more than 150K words, nearly 13% of the Mac-Morpho[15]. Below are some corpus statistics.

15. There are two versions. The first HAREM was used: https://www.linguateca.pt/HAREM

	SENTENCES	TOKENS
TRAIN	3,480	87,643
DEV	202	4,589
TEST	2,590	62,440
	6,272	**154,672**

Table 16: HAREM V. 1 Corpus Statistics

The HAREM corpus is actually comprised of two datasets, HAREM and miniHAREM, containing 10 named entity categories: Person (PESSOA), organization (ORGANIZACAO), Location (LOCAL), Value (VALOR), Time (TEMPO), Abstraction (ABSTRACCAO), Title/Work (OBRA), Event (ACONTECIMENTO), Thing (COISA), and Other (OUTRO).

For the data used in my experiments, I followed the standard used by (dos Santos and Guimarães 2015), which splits HAREM-I into training (95%) and development (5%) sets, and uses miniHAREM for the test set. Because splitting was done randomly, however, tag distribution of my data is somewhat different from this author's. Below in Table 17 is the breakdown for tags. Note that there are 8594 total entities and that Location followed by Person are the most frequent tags.

	TRAIN	DEV	TEST	
PESSOA	918 (.20)	115 (.28)	830 (.23)	**1863**
ORGANIZACAO	858 (.19)	65 (.16)	599 (.17)	**1522**
LOCAL	1151 (.25)	71 (.17)	875 (.24)	**2097**
VALOR	404 (.09)	59 (.14)	325 (.09)	**788**
TEMPO	401 (.09)	34 (.08)	360 (.10)	**795**
ABSTRACCAO	375 (.08)	25 (.06)	203 (.06)	**603**
OBRA	189 (.04)	7 (.02)	191 (.05)	**387**
ACONTECIMENTO	94 (.02)	34 (.08)	57 (.02)	**185**
COISA	130 (.03)	1 (0)	170 (.05)	**301**
OUTRO	36 (0)	3 (.01)	14 (0)	**53**
	4556	**414**	**3624**	**8594**

Table 17: HAREM Tag Distribution

The HAREM data are in XML format. I used scripts prepared by Domingues-Fernandes (2018) to convert this format into CoNLL column format[16]. Unlike the CoNLL datasets with available PoS tags, HAREM only contains word form and NER tag column features. For instance, for the sentence *Benvindos à página Web do Aeroclube de Torres Vedras* 'Welcome to the webpage of the flight club Torres Vedras' we have:

(33)	*Benvindos*	O
	à	O
	página	O
	Web	B-LOCAL
	do	O
	Aeroclube	B-ORGANIZACAO
	de	I-ORGANIZACAO
	Torres	I-ORGANIZACAO
	Vedras	I-ORGANIZACAO
	.	O

In addition, to the standard HAREM with ten classes, there is another available transformed dataset (selective HAREM) with only four categories similar to the CoNLL standard, with Person (PER), Organization (ORG), Abstraction (MISC), Location (LOC) tags[17]. The tag distribution is given below.

	TRAIN	DEV	TEST	
PERSON	1024 (.30)	9 (.08)	830 (.33)	**1863**
ORG	896 (.26)	27 (.23)	599 (.24)	**1522**
LOC	1165 (.34)	57 (.48)	875 (.35)	**2097**
MISC	373 (.10)	27 (.23)	203 (.08)	**603**
	3458	**120**	**2507**	**6085**

Table 18: Selective HAREM Tag Distribution

16. https://github.com/ivoadf/PT_NER_DL

17. https://www.linguateca.pt/aval_conjunta/HAREM/CDSegundoHAREM.xml

Although the HAREM dataset is small in size, it is nevertheless important, since it is one of the few available for NER in Portuguese. In the next section, CRFs are briefly discussed.

5.3 CRFs for NER

Unlike for PoS tagging, where often the neural model perform fairly well merely using hidden states as features to make tagging decisions independently of surrounding context, NER is a task where even stronger dependencies between tags hold, leading to quite poor performance when using softmax decoding (Lample et al. 2016). CRF's have proven quite useful for this task. The nature of the BIO or BIOSE annotation commonly used leads to constraints against I-Y following B-X, which are quite general, a generalization missed in the absence of a CRF.

5.4 Evaluation

Following dos Santos et al. (2015), the CONLL-2003 evaluation script was used to evaluate the NER task. Although this same script script was used to evaluate the PoS task, the computation used is different. For the PoS task, a raw score can be used, since there are no outside (O) tags. In the case of NER, however, not every token is inside the span. In other words, only named entity spans (in the predicted and gold tags) are part of the computation, everything else is outside. Furthermore, in addition to accuracy, precision, recall and F1 metrics are also reported here. Note that accuracy is not reported since it included non-entities in its calculation and therefore is not a reliable indicator of performance.

(34) Evaluation metrics

Precision is the number of correctly identified named entities (tp) divided by the total retrieved entities (tp + fp):

$$P = tp \div (tp + fp)$$

Recall is the number of correctly identified named entities (tp) divided by the total true entities (tp + fn):

$$R = tp \div (tp + fn)$$

F1 is the harmonic mean precision and recall:

$$F1 = 2 \cdot R \cdot P \div (R + P)$$

Only complete matches are included in the count and not partial matches. Thus if the system misses one or more items in the entity span, this does not constitute correct identification.

5.5 State of the Art for NER in Portuguese

In this section, we discuss two neural models (Dos Santos and Guimarães 2015; Domingues-Fernandes 2018) with state-of-the-art results for NER using the HAREM corpus.

5.5.1 CharWNN, dos Santos and Guimarães (2015)

Dos Santos and Guimarães (2015) uses the same architecture as that of dos Santos and Zadrozny (2014), referred to as the CharWNN model. CharWNN is a feedforward model which includes pretrained word and in task character embeddings. For more details refer to section §4.21. Dos Santos et al. (2015) in this article focuses on the NER task, using the CoNLL-2002 corpus for Spanish and the HAREM-I corpus for Portuguese. The same hyperparameters are used for both datasets and are almost identical to those of dos Santos et al. (2014), with the exception of more convolutional units.

Parameter	Value
Word embedding dimension	100
Character embedding dimension	10
Word and character content window size	5
CNN filters	200
Hidden units	300
Learning rate	.0075

Table 19: CharWNN NER tagging model, dos Santos et *al*. (2014)

In the same vein of Santos et al. (2014), dos Santos et al. (2015) also compare the effect of using different embedding setups, all combinations of word and character, plus the use of the handcrafted features of capitalization and suffixation with the word embeddings. The results in Table 20 below were obtained using the CoNLL 2003 evaluation script[18].

MODEL	**FEATURES**	**TOTAL SCENARIO**			**SELECTIVE SCENARIO**		
		PR	**REC**	**F1**	**PR**	**REC**	**F1**
CHARWNN	word, char	67.16	**63.74**	65.41	73.93	**68.68**	71.23
WNN	word, cap, suffix	**68.52**	63.16	**65.73**	**75.05**	68.35	**71.54**
WNN	word, cap, suffix	63.32	52.23	57.84	68.91	58.77	63.44
WNN	word, cap, suffix	57.10	50.65	53.68	66.30	54.54	59.85

Table 20: Results (dos Santos et *al*. 2015)[19]

Except for recall, the CharWNN setup does not completely outperform the WNN system which lacks character embeddings and utilizes suffix and capitalization features. Dos Santos et al. (2015) hypothesizes that this may be due to the small size of the dataset[20]. See §5.4 below for more regarding this. Dos Santos et al. (2015) also used a different selective dataset than that described above containing five rather than ten categories: Person, Organization, Location, Date, Value. This selective scenario

18. These results were evaluated using the CoNLL-2002 evaluation script https://www.clips.uantwerpen.be/conll2002/ner/bin/conlleval.txt
19. *Selective Scenario* refers to the HAREM selective dataset.
20. The evaluation script from HAREM-I yielded higher scores for dos Santos (2015) et. al.'s model, but this evaluation was based on criteria other than purely complete span matches

yielded more favorable results due to the collapsing of problematic categories into O. We will discuss this more in §5.6 below.

5.5.2 Domingues-Fernandes (2018)

Domingues-Fernandes' (2018) master's thesis deals with NER in Portuguese. He implements several architectures for the task: the window-based approach of dos Santos et al. (2015), including extra feature engineering, and several bidirectional RNN models (BiLSTM with and without in-task character embeddings.

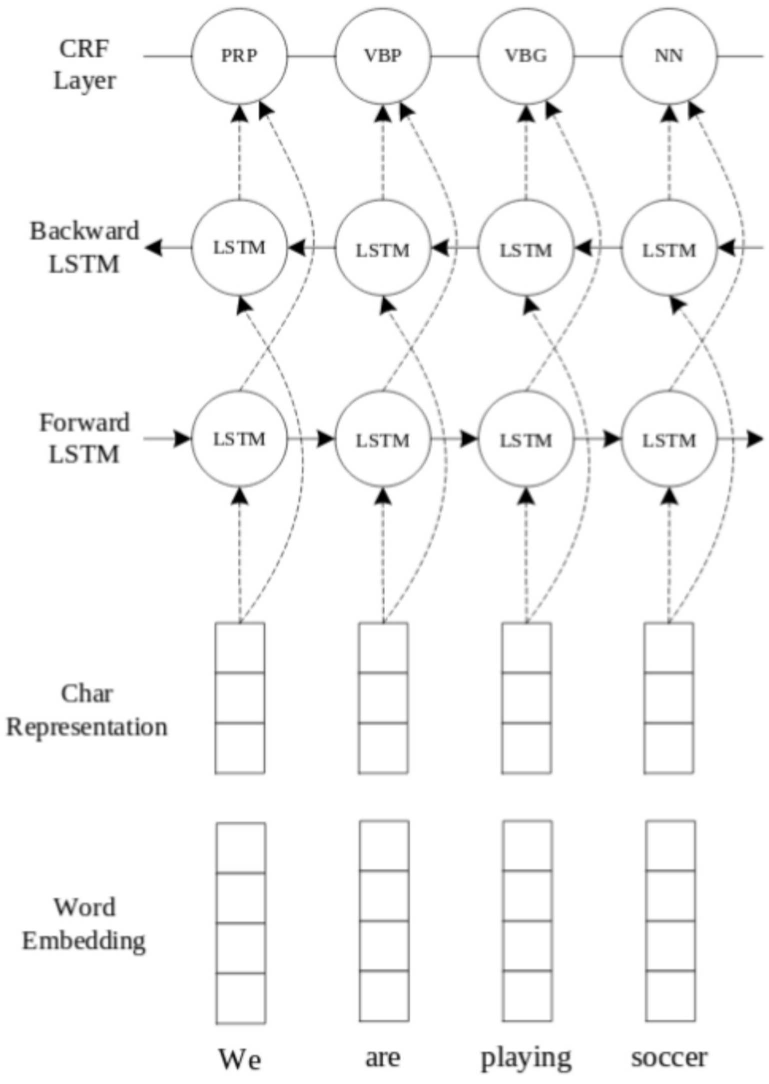


Figure 13: BiLSTM with Convolutional Character Level Embeddings (Ma et al. 2016)

As we have seen in our discussion of PoS tagging in the previous chapter, RNNs are especially adept for sequence labeling. As a consequence, both PoS and NER tasks can share the same architecture. Domingues-Fernandes (2018) achieves his best results for the HAREM total dataset using a BiLSTM with character CNN embeddings (BiLSTM_CNN), based on *(Ma & Hovy, 2016)*. Figure 14 above illustrates NER with a BiLSTM with character CNN embeddings. Once a word representation is obtained with the character level embeddings, these features are concatenated with pretrained word embeddings such as word2vec and fed to the BiLSTM, which at each timestep outputs a hidden state $\mathbf{h}_t$ representing the best tag class given the history. These outputs are then fed to a CRF which outputs a probability distribution for all output classes for the input sentence.

For the selective scenario, Domingues-Fernandes (2018) achieves his best results using Lample et al. (2016)'s Theano implementation of a BiLSTM[21], referred to as BiLSTMChar. This model is similar to that of Ma et al. (2016), but uses in-task character embeddings from a BiLSTM rather than a CNN. Dropout is applied to concatenated word and character-level representations before the BiLSTM layer. The forward and backward outputs are concatenated and fed to another *tanh* dense layer the same size of the original dimension, thus downsizing the concatenated outputs of the BiLSTM. The final later is decoded with a CRF. The hyperparameters as well as results of the model are given below in Tables 21-22.

21. https://github.com/glample/tagger

Parameter	Value
Character CNN kernel size	3
Number of CNN filters	30
Word embedding dimension	64
Character embedding dimension	25
BiLSTM state size	256
Learning rate	.1
Decay rate	.05
Dropout	.5
Batch size	16

Table 21: Hyperparameters of BiLSTM with Character CNN [BiLSTM_CNN] (Domingues-Fernandes 2018)

Parameter	Value
Word embedding dimension	100
Character embedding dimension	25
Character BiLSTM state size	25
BiLSTM state size	100
Learning rate	25
Dropout	.5
Batch size	

Table 22: Hyperparameters of BiLSTM with BiRNN Character Embeddings [BiLSTMChar] (Domingues-Fernandes 2018)

The results for both of these models obtained using the CoNLL-2003 script are given below:

MODEL	TOTAL SCENARIO			SELECTIVE SCENARIO		
	PR	REC	F1	PR	REC	F1
WNN (dos Santos et al 2015)	68.52	63.16	65.73	**75.05**	68.35	**71.54**
BILSTMCHAR	68.94	65.84	67.35	71.90	**68.49**	70.15
BILSTMCNN	**72.64**	**67.50**	**69.97**	70.67	66.35	68.44

Table 23: Best Results for Domingues-Fernandes (2018)

For the total scenario (10 tags) the BiLSTM with CNN-based character embeddings outperforms the window-based network with capitalization and suffix features. For the sake of comparison, dos Santos et al (2015)'s selective results are given here as well, but it should be noted that the datasets used differ in the tagsets used. In any case, the windows-based approach performs better in this task for the selective dataset, perhaps suggesting that the usefulness of character-level features are not as important for NER in the selective scenario. However, the results in §5.5.2 speak in favor of character-level features here, and it seems more likely that what we are observing here is the impact of the window-based model with its feature engineering. There, however, is a clear performance gain for the model leveraging CNN character embeddings in the total scenario. Why the CNN performs better than the BilSTM for extracting character features is not clear. In the next section, I present the model and results for the setups I experimented with for this task.

5.6 Model

The sequence labeling model introduced in §3.2, leveraging character language model (CharLMs) embeddings, which was used in the PoS task (§4.3), was utilized for the NER tagging task with only fine tuning of the hyperparameters. Below I describe the hyperparameters found to be optimal and several distinct combinations of input representations (features) that were evaluated.

5.6.1 Setups

The baseline and experimental setups described in §3.3.2 (presented below for convenience) were applied to the NER tagging task.

The baseline setups are:

HUANG: A classic BiLSTM-CRF setup with pretrained word embedding —a reimplementation of Huang et al. (2015)

LAMPLE: A hierarchical BiLSTM-CRF setup with pretrained word embeddings and in task character-level embeddings—a reimplementation of Lample et al. (2016)

The experimental setups are:

FLAIR: A BiLSTM-CRF setup with pretrained character language model (CharLM) embeddings—a reimplementation of Akbik et al. (2018)

FLAIR$_{+PWE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and pretrained word embeddings (PWE)

FLAIR$_{+CLE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and in task character-level embeddings (CLE)

FLAIR$_{+WE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and in task word embeddings (WE)

FLAIR$_{+PWE+CLE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and pretrained word embeddings (PWE) and in task character-level embeddings (CLE)

FLAIR$_{+PWE+WE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and both pretrained and in task word embeddings

FLAIR$_{+WE+CLE}$: A BiLSTM-CRF setup with with pretrained character language model (CharLM) and both in task word (WE) and in task character-level embeddings (CLE)

FLAIR$_{+ALL}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and both in task word (WE) and in task character character-level embeddings (CLE)

5.6.2 Hyperparameters

A handful of experiments were carried out with differing batch normalization and dropout schemes, and it was found that neither normal nor locked dropout before either the character-level or sentence-level BiLSTM was favored. Dropping out at consistently yielded worse results. This contrasts with the PoS tagging task, in which dropout before LSTMs improved performance. As with PoS tagging, however, batch normalization before the sentence-level LSTMs helped with overfitting. The reason dropout did not help with overfitting is not clear.

As mentioned, the HAREM dataset was small, and there was a huge margin in the performance of training and development sets indicative of overfitting. The dimensionality of the CharLM embeddings alone were 4096, and in each BiLSTM there are 8 weight matrices, leading to a huge amount of parameters in the setup. During experimentation a linear layer was added, projecting this high dimension space to a lower dimension space in the range of 96 to 2048. It turned out that this worsened performance. Another hyperbolic tangent (tanh) layer was also added between the BiLSTM and output layer similar to Lample et al. (2016)'s implementation, but this was also not favorable. The optimal state size for the BiLSTM tagger was found to be 256, half the size of that of the PoS tagger. This was most likely due to the small size of the HAREM dataset.

I carried out another experiment with character embeddings from a CNN rather than an RNN (BiLSTM), similarly to was done for PoS tagging, using the basic FLAIR setup. The motivation for this was that Domingues-Fernandes (2018) achieved his best results for the HAREM total dataset using a BiLSTM with character CNN embeddings (BiLSTM_CNN). The results of this setup based on Ma and Hovy (2016) are given in Table 24, along with RNN character-level embeddings (CLE). The notation is *#filters-max_kernel_size*, e.g. 300-3 denotes 300 filters, with kernels of size 2 and 3. Despite the number of filters and kernel sizes experimented with, however, RNN CLE consistently outperformed CNN CLE for both the total and selective data sets.

MODEL-	TOTAL SCENARIO			SELECTIVE SCENARIO		
	PR	REC	F1	PR	REC	F1
200-3	65.63	62.39	63.97	69.32	64.34	66.74
200-4	65.41	61.89	63.60	67.96	64.74	66.31
300-3	65.17	61.59	63.33	69.12	64.18	66.56
300-4	64.00	62.00	62.99	67.97	65.10	66.50
400-3	65.93	62.75	64.30	68.23	65.78	66.98
400-4	65.04	61.81	63.38	66.01	63.90	64.94
500-3	65.45	63.08	64.24	67.85	65.34	66.57
500-4	64.79	61.48	63.09	68.16	**66.02**	67.07
RNN CLE	**67.13**	**63.16**	**65.08**	**69.75**	65.86	**67.75**

Table 24: Results NER task comparing RNN versus CNN character embeddings

These hyperparameters are summarized in Table 25 below.

PARAMETER	VALUE
Pretrained fastText word embedding dimension	300
In task word embedding dimension	256
Character BiLSTM state size	256
BiLSTM state size	256
Optimizer	SGD
Gradient clipping	.25
Batch normalization before sentence-level BiLSTM	True
Initial learning rate	.1
Annealing rate	.5
Patience	5
Batch size	32
Epochs	35

Table 25: Hyperparameters for sequence tagger (NER)

5.7 Results

In this section I present the models used in experiments with the HAREM dataset. For the sake of comparability, I used similar datasets as those of dos Santos et al. (2015) and Domingues-Fernandes (2018), with the only difference being that I used my own split of training and development sets, following dos Santos et al (2015) and like both of these authors took the test data from the miniHAREM. For the selective scenario, I used the same data as Domingues-Fernandes (2018).

In Table 26 below, the results of these baselines and setups are presented, vis-à-vis the best published results to my knowledge, dos Santos et al. (2015) and DOMINGUES-FERNANDES (2018).

MODEL	**TOTAL SCENARIO**			**SELECTIVE SCENARIO**		
	PR	**REC**	**F1**	**PR**	**REC**	**F1**
baselines						
HUANG	63.11	55.24	58.92	68.20	58.68	63.08
LAMPLE	58.51	55.77	57.11	66.94	60.83	63.74
models						
FLAIR	60.41	56.93	58.62	70.05	65.86	67.89
FLAIR$_{+PWE}$	67.17	62.83	64.93	70.52	65.38	67.85
FLAIR$_{+CLE}$	65.62	61.84	63.67	71.15	65.19	68.42
FLAIR$_{+WE}$	65.45	60.95	63.12	69.66	65.94	67.75
FLAIR$_{+PWE+CLE}$	66.43	62.75	64.54	70.16	65.18	67.58
FLAIR$_{+PWE+WE}$	65.21	61.09	63.09	70.05	66.53	68.25
FLAIR$_{+WE+CLE}$	65.42	62.53	63.94	68.53	64.62	66.52
FLAIR$_{+ALL}$	67.13	63.16	65.08	69.75	65.86	67.75
best published						
dos Santos et al. (2015)	68.52	63.16	65.73	**75.05**	68.35	**71.54**
Domingues-Fernandes (2018)	**72.64**	**67.50**	**69.97**	71.90	**68.49**	70.15

Table 26: Results NER task

In the total scenario, it is surprising that the basic FLAIR configuration performs slightly worse than the one of the baselines, HUANG, with only pretrained embeddings, and only somewhat better than LAMPLE, with added character embeddings. Recall that for PoS tagging, the addition of the CharLM to the setup led to significant error reduction. Addition of pretrained word embeddings to this configuration leads to a sharp rise in performance, slightly more than that of in task character or word embeddings. This is the case because of the paucity of training data. Pretrained word embeddings outperform in task ones when there is not enough training data for task-specific semantic features to be learned. We have seen the opposite trend with PoS tagging (§4.4.3), where data was sufficient to learn task-specific word representations.

When two embeddings of any type are combined, we observe a surprising drop in performance. This was the case for PoS tagging, such that the performance of FLAIR$_{+PWE}$ is better. This suggests that this may be overkill for such datasets of small to medium sizes. However, if this is the case, it is then quite surprising that for the total scenario combination of all of the embeddings (FLAIR$_{+ALL}$) leads to the best result among the setups. More expected would be preference for a simpler model. In light of the decent performance of baseline HUANG, I hypothesize that although decent performance is possible with just pretrained word embeddings, the addition of the LM allows the model to learn shallow semantic structure more than just the word embeddings by themselves. Similarly, adding character embeddings to this mix, leads to more favorable results because these in task context-insensitive character embeddings combined with both pretrained and in task word embeddings lead to better word semantics than either the context-sensitive LM or context-insensitive character embeddings in isolation.

In the selective scenario, we observe somewhat different results. Among the baselines, character-enhanced LAMPLE performs that best, and similarly among the experimental setups, FLAIR$_{+CLE}$ is the winner. As mentioned in 5.4, the capitalization and suffix features of dos Santos et al. (2015) seem to be favored here. It is not clear to me why the CNN-based character embeddings of Domingues-Fernandes (2018) outperforms the LM setups. Perhaps the smaller number of parameters of this simpler model avoided the mentioned overfitting problem encountered by these experimental setups.

5.7.1 Errors

In both the total and selective dataset system results, there were some confusions of Person and Organization, e.g. *alunos do 1^o Ciclo do Ensino Básico da Escola Codeçoso* (Person) 'students from the first Cycle of the Elementary School Codeçoso'. In the total dataset there is a combination of Value, Event, and Abstraction tags for this entity, which should be all Person. For the selective dataset, the system labeled the first part of the entity as O (rather than wrong entity classes) and the last part (*Escola Codeçoso*) as an Organization, the same as in the total dataset. Thus it appears that having less tags may have led to less confusion but still resulted in a missed correct entity. Similarly, *feira do Soajo* is correctly a Location in the total dataset, yet labeled as a Person in the selective dataset.

There are some apparent inconsistencies in the annotation that led to very frequent errors. Titles such as *senhor* 'Mr.' and *professor* 'Professor', which in Portuguese are not capitalized, are not part of the entity, yet grandmother (*avó*) and grandfather (*avô*) before a noun is included as the entity. In both datasets, often the title was labeled as Person, and less frequently it was correctly marked as O. 'Grandmother/grandfather' was always O. Titles (Obra) were problematic in the total dataset, and their elimination in the selective dataset seems to have helped. Abstraction (Misc in the selective dataset) was often missed, e.g. HISTÓRIA 'history' or labeled as Organization, e.g. *(ministro da) Agricultura* (both datasets). Numerals are often confused, particularly when they are ordinals (i.e. number + ^{o}) in an entity. However, the selective dataset has changed this notation, somewhat inconsistently, to the fully spelled (non-numeric) ordinal, cf. *1^o Ciclo* (selective *Primeiro Ciclo)* for the above entity, and *1^o Cabo* 'first head' (Person), the same in both datasets, in which the number is labeled as Value in the total dataset and O in the selective. In cases like these, both changing spelling and removing the VALUE tag also has led to better results.

5.8 Conclusions

In this chapter, we have focused on the task of NER. In contrast to what we observed for PoS tagging in Chapter 4, the addition of a character language model (CharLM) did not lead to immediate improvement over the baselines for this task. It was speculated that this was because of the paucity of training data. This hypothesis seems to have been confirmed, because when pretrained word embeddings, independent of the task at hand, were added to the base CharLM, significant gains were achieved. Thus the precomputed fastText embeddings allowed the model to learn word representations that were not possible with task-specific word and character-level embeddings alone. When all of the embeddings are combined in the full setup, slightly more gains were achieved, again demonstrating the usefulness of using both precomputed contextualized and non-contextualized word representations in an ensemble with task-specific word features.

CHAPTER 6

Verbal Multiword Expression (VMWE) Identification

The task of verbal multiword expression (VMWE) identification is similar to NER, in that it involves detecting a specific type of linguistic construction in unstructured text and then classifying it into a fine grained category such as verbal idiom (*take a nap, fly off one's rocker)* or light verb construction (*make a decision, do some thinking*). Multiword expressions are idiosyncratic and typically non-compositional in meaning and are long considered a 'pain in the neck' for NLP (Sag et al. 2012).

MWE identification is often a necessary step in machine translation and parsing. The international research community PARSEME is devoted to the task and organizes regular workshops (the first MWE workshop in 2008) and multilingual shared tasks since 2017 (Savary et al. 2018). In this chapter, we will first define MWEs and then examine some of the published results for the PARSEME shared task. Finally, I present experimental setups and hyperparameters I used for this task.

6.1 Toward a Definition of MWE

Consider the following examples:

(35)	*Benvindos*	O
	à	O
	página	O
	Web	B-LOCAL
	do	O
	Aeroclube	B-ORGANIZACAO
	de	I-ORGANIZACAO
	Torres	I-ORGANIZACAO
	Vedras	I-ORGANIZACAO

(36) *After much frustration at work, John called it a day.*

(37)	*Ela*	*tocou*	*no*	*assunto.*
	She	touched.3.SG	on-the	matter.
	She discussed the matter.			

(38)	*Gabriela*	*quer*	*se*		*fazer*	*médica*
	Gabriela	want-3.SG	REFL.3.SG		make	doctor
	Gabriela wants to be a doctor.					
(39)	*Estende-se*	*por*	*uma*	*área*	*de*	*29,34 km^2.*
	Extends.REFL.3.SG	through	an	area	of	29,34 km^2.
	It extends for an area of 29,34 km^2.					

The meaning of the idiom *call it a day* ‘finish’ in (36) is not derivable from its parts and is thus non-compositional. While the Portuguese idiom *tocar no assunto* ‘discuss, comment’ in (37) may be partially compositional, since *to touch something* is metaphorically *to elaborate on it*, to a nonnative speaker, this construction would most likely need to be learned by special rule. The reflexive constructions in (38-39) are also problematic. In (38) *fazer-se* is not strictly *to make oneself*, and *extender-se* in (39) is not really to *extend oneself*. The reflexive clitic pronoun here intransitivizes the verb, adding a nuanced meaning in (38) but not in (39), which is thus not a VMWE. Typically VMWEs such as (35-38) are fully to partially non-compositional on the scale of compositionality.

The following properties are among the most typical of VMWEs (Savary et al. 2018):

1. Semantic non-compositionality: The meaning of VMWEs cannot be inferred in a grammatically regular way from their parts.

2. Lexical and grammatical rigidity: VMWEs are subject to lexico-grammatical constraints. When words in an idiom are replaced by related words, the expression loses its idiomatic meaning, e.g. *call it a day* → **denominate it a day, tocar no assunto* → **apalpar o assunto*. If the expression is modified syntactically, it may lose its meaning in the case of an idiom—e.g. **call it a long day.*

3. Grammatical variability: In spite of (2), VMWEs still inflect and undergo passivization and those that are light verbs can accept modifiers, e.g. *John takes (took, etc.) a **long** nap.*

4. Discontinuity: The variability in (3) often leads to discontinuous components of the VMWE., e.g. ***John takes*** *a long* ***nap, Naps*** *are often* ***taken*** *after heavy meals.*

These properties make task of VMWE identification challenging, more so than NER, since named entities by their vary definition are rigid designators and quite easy to detect. In the next section, we explore the VMWE categories of the PARSEME shared task.

6.2 PARSEME Shared Task

The PARSEME (Savary et al. 2018) multilingual shared task on VMWE identification, organized by the European PARSEME community, is the largest of its kind. There have been open and closed track competitions in 2017 and 2018. The 2017 event (Savary et al. 2017) included 18 languages. The 2018 (Savary et al. 2018) event involved some changes in the categories, data format, and languages. I chose to work only with the Portuguese data since I have only trained language models for this language and it was my goal to utilize experimental setups similar to those presented in §4.3.2 and §5.5.2. I also chose to work with the 2017 PARSEME data in order to compare my results to those of previous neural implementations.

6.2.1 VMWE typology

The 2017 PARSEME shared task identifies both language-agnostic (universal) and language-specific VMWE types. Among language-agnostic types, which are shared by all languages in the task, are:

1. Light Verb Constructions (LVCs) in which the verb has lost its meaning (i.e is bleached), taking on the meaning of its nominal complement

Eu	*dei*	*uma*	*caminada.*
I	gave	a	walk.

I took a walk.

Isso	*me*	*dá*	*dor*	*de*	*cabeça.*
That	me	give.3.SG	pain	of	head.

That gives me a headache.

2. Idioms (ID):

Ele	*sempre*	*faz*	*das*	*suas.*
He	always	do.3.SG	of	his own.

He always does stupid things.

Among the language-specific types, which characterize some but not all languages, are:

3. Inherently Reflexive Verbs (IReflVs): Frequent in Romance and languages, where the pronoun changes the meaning or subcategorization frame of the verb

O	*professor*	*se*	*enganou.*
The	professor	3.SG.REFL	deceived.

The professor made a mistake.

4. Verb Particle Constructions (VPCs): Pervasive in Germanic and Hungarian, rare in Romance and Slavic. The particle completely alters the meaning of the verb, e.g. *to do in* ‘kill’ or adds a predictable non spatial meaning to the verb (Ramisch et al. 2018), e.g. *to eat up*

5. Other: Not belonging to (1-4), e.g. *to drink and drive, to short-circuit*

No category is present in all languages, but ID and LVC are used in almost languages, while the most frequent category in all the corpora is IReflV due to its pervasiveness in Slavic and Romance (Savary et al. 2017).

6.2.2 Corpora

The entire PARSEME corpora contain 230,062 sentences (4,5 M tokens) for training and 44,314 sentences (900K tokens) for testing. There are 3947 MWEs in the Portuguese corpus. The distribution of tags is given in Table 25. Note that the most frequent category for Portuguese in LVC, followed by ID, and IReflV. There are no VPCs, and the other category is extremely rare and does not appear in the test dataset.

	LVC	ID	IREFLV	OTHER
TRAIN	2110 (.61)	820 (.24)	515 (.15)	2 (0)
TEST	329 (.66)	90 (.18)	81 (.16)	0 (0)
	2439 (.62)	910 (.23)	596 (.15)	2 (0)

Table 27: PARSEME Tag Distribution (Portuguese)

The Portuguese training data was split (95% ~ 5%) into training and development sets respectively. Below are some corpus statistics.

	SENTENCES	TOKENS
TRAIN	18,559	341,349
DEV	1081	17,996
TEST	2600	54,675

Table 28: PARSEME Corpus Statistics (Portuguese)

Each dataset contains two files. The first file is in CONLL-U column format, containing 10 columns with morphosyntactic annotations: index, form, lemma, universal PoS (upos), language-specific PoS (xpos),

(40) CONLL-U format

```
1   Você    você    PRON    PRON    Case=Nom|...|PronType=Prs                 2  nsubj  _ _
2   sabia   sabia   VERB    VERB    Mood=Ind|...|Tense=Imp|VerbForm=Fin       0  root   _ _
3   ?       ?       PUNCT   .       _                                         2  punct  _ _
```

The second file is in PARSEMETSV format, containing index, form, no_space (for printing in which punctuation should not be preceded by a space) and the target MWE label, which are enumerated in the following form *ID:MWE-TYPE.* More than one MWE associated with a word are separated by semicolons, such as *ter* below in (40) which heads three separate light verb constructions (LVCs). Every MWE instance after the first is annotated with the ID rather than MWE type.

(41) PARSEMETSV format

1	*É*	_	_
2	*necessário*	_	_
3	*ter*	_	1:LVC;2:LVC;3:LVC
4	*ensino*	_	1
5	*médio*	ns	1
6	*,*	_	_
7	*experiência*	_	2
8	*em*	_	_
9	*vendas*	_	_
10	*e*	_	_
11	*disponibilidade*	_	3
12	*de*	_	_
13	*horário*	nsp	_
14	*.*	_	_

As this annotation scheme is not amenable to neural classification, preprocessing was performed in such as way as to allow classification of subsequent occurrences of the MWE. A variant of the BIOSE (Beginning Inside Outside Single End) scheme with an additional – label instead of O for all outside tags was chosen since it easily allowed the identification of discontinuous MWEs. Thus all IDS were replaced by the full MWE-TYPE. However, multiple tags on single tokens as well as embedded MWEs such as 1:LVC and 2:LVC above are problematic for such a system. These cases were dealt with by eliminating all tags other then the first and erasing any ID chains. After preprocessing (40) appears as (41) below:

(42) PARSEMETSV format

1	*É*	_	_
2	*necessário*	_	_
3	*ter*	_	B-LVC
4	*ensino*	_	I-LVC
5	*médio*	ns	F-LVC
6	*,*	_	_
7	*experiência*	_	_
8	*em*	_	_
9	*vendas*	_	_
10	*e*	_	_
11	*disponibilidade*	_	_
12	*de*	_	_
13	*horário*	nsp	_
14	*.*	_	_

These two files were joined into one input file consisting of 11 columns in a modified CoNLL-U format identical to that used by the 2018 PARSEME task[22] . The script used for this is included in the attached source code.

6.2.3 Evaluation

As we saw for NER tagging §5.4, precision, recall, and F1 score were used for the MWE task. The 2017 PARSEME scripts were used, providing per MWE (full match) as well as per token (partial match) results.

6.3 Neural Models for MWE (PARSEME Task)

In this section, we review one neural model MUMULS, which participated in the 2017 PARSEME shared task (Klyueva, Doucet, and Straka 2017). Although not scoring the highest in the 2017 PARSEME task, MUMULS, achieved quite good results for the majority of languages without using feature engineering and a shift-reduce parsing setup. Next I present a later modification of this model (Variš and Klyueva 2017), which, although it did not compete in the task, has improved on MUMULS and published results using the PARSEME corpora.

6.3.1 MUMULS, Klyueva, Doucet, and Straka (2017)

The MUMULS system (Klyueva, Doucet, and Straka 2017) was the only neural model to participate in the 2017 PARSEME task. It is a bidirectional GRU

22. The NO_SPACE column was eliminated since it was not used for evaluation purposes.

(BiGRU), which is quite similar to the BiLSTM model used in this thesis[23]. Although the gating is distinct to that of the LSTM, the recurrent connections allow the GRU to encode long range dependencies (history) in a similar fashion. The BiGRU was also implemented in TensorFlow and, as is discussed below, is a variant of my model and is one of the setups which I have reimplemented. Below are the hyperparameters.

PARAMETER	VALUE
Word, lemma, PoS embedding dim	100
BiGRU state size	100
Optimizer	Adam
Decoder	Softmax
Learning rate	.001
Batch size	64
Epochs	14

Table 29: Hyperparameters (MUMULS)

MUMULS uses in task word, lemma, and PoS embeddings, which are concatenated and fed to the RNN. The outputs are then concatenated and passed to a final dense layer upon which softmax classification is finally performed.

Data preprocessing although not identical to that used by me for this task, was similar, using a CONT for all subsequent MWE tags rather than the BIOSE scheme. Multiple tags for a tag other than the first were also thrown away. The PARSEME evaluation script was used, giving the following results for Portuguese:

MODEL	**PER MWE**			**PER TOKEN**		
	PR	**REC**	**F1**	**PR**	**REC**	**F1**
MUMULS	53.58	37.40	44.05	82.47	47.17	60.01
Al Saied et al. (2017)	**75.43**	**60.80**	**67.33**	**80.05**	**63.70**	**70.94**

Table 30: Results for Portuguese (MUMULS)

23. https://github.com/natalink/mwe_sharedtask/blob/master/mwe_tagger.py

The results for Portuguese were were somewhat lower than those of other languages. In §6.8 I comment on this more.

6.3.2 Variš and Klyueva (2017)

Variš and Klyueva (2017) improved upon MUMULS. I will refer to those version as MUMULS+. In task character-level embeddings (CLE) are added to MUMULS. Experimental setups are run with CNN as well as RNN-based CLE. Similarly to my experiments, the CNN-based CLE feared the better of the two. Several encoder architectures for the classification task are also experimented with, a BiLSTM (like MUMULS), a deep convolutional encoder, and a self-attentive multihead encoder. The BiLSTM encoder is identical to what was used in MUMULS. The deep convolutional encoder (Gehring et al. 2017) works like the CNN architectures which have been discussed, with the exception that positional embeddings which encode the index of inputs, residual connections, and a gating mechanism are used here. Self-attention (Vaswani et al. 2017) is used instead of a BiLSTM to encode long-distance dependencies. As all pairs of words in an input are mapped, this reduces the path length that gradients must propagate to one. If the sentence is long, self-attention is more effective than a BiLSTM, which has trouble such long range dependencies. In addition to these different encoders, two decoder setups are also used: softmax (like in MUMULS) and a CRF (like in my system).

Experiments were carried out with the Czech PARSEME dataset. The best embedding results were obtained via the CNN. With regard to decoders, self-attention and BiLSTM yielded similar results followed by the deep convolutional decoder. As is expected for this task, the CRF fared better than softmax. The self-attention encoder with CNN CLE and CRF was chosen for the PARSEME task. The hyperparameters of this setup are summarized below.

PARAMETER	VALUE
Character CNN filters	?
Character CNN max kernel size	6
Word, lemma, PoS embedding dim	100
Self-attentive layers	3
Self-attentive encoder heads per layer	10
Fully connected layer hidden units	450
Dropout	.8
Decoder	CRF
Optimizer	?
Learning rate	?
Batch size	?
Epochs	14

Table 31: Hyperparameters (MUMULS+)

The results for the Portuguese VMWE identification task are given below:

MODEL	PER MWE	PER TOKEN
	F1	**F1**
MUMULS	44.05	60.01
MUMULS+	40.00	52.00
Al Saied et al. (2017)	**67.33**	**70.94**

Table 32: Results for Portuguese (MUMULS)

For most of the languages other than Portuguese, the MUMULS+ setup performed better than its predecessor. Both MUMULs and MUMULS+ had some difficulties with Portuguese. We will discuss this more in §6.8.

6.4 Model

The model used for the task of VMWE identification is identical to that used for the tasks of PoS tagging and NER, with only minor changes in the hyperparameters, described below.

6.4.1 Setups

The setups are identical to those used for PoS tagging and NER, with the exception that the three additional setups with in task lemma, PoS, and both lemma and PoS embeddings were also used. For this task, I slightly altered the baselines from the those used for NER, since for this task the setups involve lemma and tag embeddings, in addition to those for the other two tasks. The baselines are MUMULS and MUMULS+ (the improved version presented in §6.3.2 with a BiLSTM in lieu of self-attention).

The baseline setups are:

MUMULS: a BiLSTM-SOFTMAX setup with in task word, lemma, and tag embeddings—a reimplementation of (Klyueva, Doucet, and Straka 2017)

MUMULS+: A BiLSTM-CRF setup with in task word, lemma, and tag embeddings, as well as character-level embeddings—a reimplementation of Variš and Klyueva (2017)

The experimental setups are:

FLAIR: A BiLSTM-CRF setup with pretrained character language model (CharLM) embeddings—a reimplementation of Akbik et al. (2018)

FLAIR$_{+PWE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and pretrained word embeddings (PWE)

FLAIR$_{+CLE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and in task character-level embeddings (CLE)

FLAIR$_{+WE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and in task word embeddings (WE)

FLAIR$_{+PWE+CLE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and pretrained word embeddings (PWE) and in task character-level embeddings (CLE)

FLAIR$_{+PWE+WE}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and both pretrained and in task word embeddings

FLAIR$_{+WE+CLE}$: A BiLSTM-CRF setup with with pretrained character language model (CharLM) and both in task word (WE) and in task character-level embeddings (CLE)

FLAIR$_{+ALL}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and both in task word (WE) and in task character character-level embeddings (CLE)

There are three additional experimental setups for the task of Verbal Multiword Expression (VMWE) identification:

FLAIR$_{+ALL+LEMMA}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and in task word (WE), character character-level (CLE), and lemma (LEMMA) embeddings

FLAIR$_{+ALL+TAG}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and in task word (WE), character character-level (CLE), and tag (TAG) embeddings

FLAIR$_{+ALL+LEMMA+TAG}$: A BiLSTM-CRF setup with pretrained character language model (CharLM) and with pretrained word embeddings (PWE) and in task word (WE), character-level (CLE), lemma (LEMMA), and tag (TAG) embeddings.

6.4.2 Hyperparameters

A number of experiments were carried out to tune the hyperparameters for the model. First, I tested several setups with CNN versus RNN character embeddings for the FLAIR$_{+ALL+LEMMA+TAG}$ full setup. In contrast to what was observed for PoS tagging and NER. it was found that CNN character embeddings outperformed RNN character embeddings. In Table 33 below these results are summarized. Note that the setup notation is *#filters-max_kernel_size*, e.g. 300-3 denotes 300 filters, with kernels of size 2 and 3.

MODEL	PER MWE			PER TOKEN		
	PR	REC	F1	PR	REC	F1
200-3	66.88	62.60	64.67	78.72	67.32	72.57
200-4	69.81	51.80	59.47	85.55	56.64	68.16
300-3	66.23	61.20	63.62	80.90	66.48	72.99
300-4	68.79	**64.80**	**66.74**	80.50	**68.25**	**73.87**
400-3	68.25	54.60	60.67	84.60	60.17	70.32
400-4	**73.52**	42.20	53.62	**90.79**	46.70	61.68
500-3	67.83	54.40	60.38	84.18	59.80	69.92
500-4	66.45	60.20	63.17	82.03	65.27	72.70
RNN CLE						

Table 33: Different CNN Character Hyperparameter Results

Subsequently I varied the size of the RNN, from 128 up to 1024 hidden units. The best results were obtained with 512. These hyperparameters are summarized in Table 34 below.

PARAMETER	VALUE
Pretrained fastText word embedding dimension	300
In task word/lemma/PoS embedding dimension	512
Character CNN units	300
Character CNN max filter size	4
BiLSTM state size	512
Optimizer	SGD
Gradient clipping	.25
Batch normalization before sentence-level BiLSTM	True
Batch normalization Character CNN	True
Initial learning rate	.1
Annealing rate	.5
Patience	5
Batch size	32
Epochs	20

Table 34: Hyperparameters for sequence tagger (MWE)

6.5 Results

In Table 6.5, I present the results of the setups used with the PARSEME dataset. The best results were obtained with the full setup leveraging all embeddings FLAIR$_{+\text{ALL}+\text{LEMMA}+\text{TAG}}$, with an F-score of 66.74, slightly under that of the winner of the majority of the PARSEME languages, the transition-based system ATILF-LLF, with an F-score of 67.33. However, since only closed results (with no allowed pretraining) were reported for this event, in reality it is not fair to compare any of the setups other than the baselines to this system, since all of the other setups make use of pretrained character language model embeddings. Thus the best 'open' system is really MUMULS+, with identical hyperparameters to the other setups, with an F-score of 57.72. Although this result is significantly lower than ATILF-LLF, it does require the feature-engineering (feature tuning phase) of the transition system. It should also be pointed out that this baseline reimplementation performs somewhat better than Variš and Klyueva (2017), with MWE F-score of .40, slightly lower than original MUMULS (.44). It is not clear why the baselines deviate from the original systems, unless it has to do with the way the data were preprocessed and perhaps the hyperparameters and normalization used.

MODEL	PER MWE			PER TOKEN		
	PR	REC	F1	PR	REC	F1
baselines						
MUMULS	63.69	45.60	53.15	87.23	52.65	65.66
MUMULS+	66.15	51.20	57.72	83.65	56.55	67.48
models						
FLAIR	67.16	45.00	53.89	84.54	49.77	62.65
FLAIR$_{+PWE}$	64.32	47.60	54.71	82.16	53.02	64.45
FLAIR$_{+CLE}$	57.40	62.80	59.98	72.29	69.27	70.74
FLAIR$_{+WE}$	62.91	54.00	59.81	77.62	62.49	69.24
FLAIR$_{+PWE+CLE}$	56.62	62.40	59.37	72.56	68.99	70.73
FLAIR$_{+PWE+WE}$	64.86	50.20	56.60	83.54	56.55	67.44
FLAIR$_{+WE+CLE}$	64.95	55.60	59.91	81.52	60.63	69.54
FLAIR$_{+ALL}$	66.67	46.00	54.44	84.62	51.07	63.69
FLAIR$_{+ALL+LEMMA}$	61.20	59.00	60.08	78.01	65.55	71.24
FLAIR$_{+ALL+TAG}$	68.13	46.60	55.34	86.00	51.90	64.74
FLAIR$_{+ALL+LEMMA+TAG}$	68.79	**64.80**	66.74	80.50	**68.25**	**73.87**
Al Saied et al. (2017)	**75.43**	60.80	**67.33**	80.05	63.70	70.94

Table 35: Results VMWE task

Several generalizations emerge from the results of the experimental setups. It is surprising that addition of the character language model (CharLM) only leads to slight improvements over the MUMULS baseline. Recall that this setup used only in task word, lemma, and PoS embeddings, without a CRF. However, we cannot attribute this to paucity of training data as was the case for the NER task. Instead it appears that this task depends on more than precomputed contextualized character-level features. Adding either word or character embeddings improves the result of the base FLAIR model, but the CLE yield the best results, followed by the in task word embeddings. Why there is such a marked difference between the pretrained and in task word embeddings is most likely due to the fact that pretrained embeddings are most useful when there is little training data available. This is the case because, as we saw in §1.3.1 , they are context-insensitive and thus reflect generic semantic features. When there is sufficient data available, as is the case here, in task word embeddings can be learned which are better representatives of the task in hand. The PARSEME data is magnitudes larger than the NER dataset, and it is for this reason that the results here are different from those observed for NER (§5.6.3), where the opposite trend was observed, namely that pretrained word embeddings outperformed the rest. The reason the CLE embeddings fair the best is principally due to two factors. First, these embeddings represent subword features (i.e. character n-grams) which generalize better across related words (e.g. paradigms). These embeddings are capable of representing idiosyncratic aspects of training data such as spelling and capitalization (Liang and Zhao 2017). These features are important for MWE identification. Second, because of this, CLE help to combat the data sparsity problem and reduce the out of vocabulary (OOV) rate.

Combining pretrained word embeddings and CLE lead to lower results than FLAIR$_{+CLE}$. A similar result was observed for PoS tagging. It appears that the pretrained embeddings are not so favorable when non-generic in task embeddings are available. All the other embeddings seem to combine well. The FLAIR$_{+ALL}$ setup is also unusually low, perhaps because it also contains both of these two seemingly incompatible embeddings. Adding tag and lemma embeddings improves this result, however, and the best result is obtained with the full setup with all embeddings. In an additional experiment, I ran the full setup without each of the character and pretrained word embeddings together in the full setup in order to see if better results were obtained than with either of these together, and worse results were obtained. Thus it seems that lemma, followed by tag, embeddings greatly improve the lower gains of the FLAIR$_{+ALL}$ setup. Lemma embeddings contribute an in task semantic genericness shared across morphologically related forms which aids the task of MWE identification. Tag embeddings contribute shallow morphosyntactic features which help in this task, yet which are not as important as lemma semantic features.

6.5.1 Errors

Table 36 presents the results for the best setup per tag. In the detailed analysis section of Klyueva et al. (2017), several patterns were discerned with regard to mistakes made by the MUMULS tagger for Portuguese. These observations as well as some of my own are given in the sections to follow.

	LVC	ID	IREFLV	OTHER
PR	66.6	64.3	**67.1**	-
REC	67.2	60.0	**68.1**	-
F1	66.9	64.3	**67.6**	-

Table 36: Results per Tag (FLAIR$_{\text{+ALL+LEMMA+TAG}}$)

6.5.1.1 LVC

LVCs were the most frequent tag in Portuguese. Savary et al. (2018) identify LVC candidates as those constructions containing a verb and nominal complement, possibly preceded by a preposition. These candidates must pass several tests to be considered bonafide LVCs:

(43) LVC Tests

1. Does the Noun denote a state or event?
2. Does the Noun maintain one of its original senses?
3. Does the Verb only provide morphological features?
4. Can a nominal periphrasis be used to denote the same meaning as that with the verb, e.g. *John had a long walk* → *John's long walk*?

It is tricky to identify LVCs and correct identification was confounded by several additional factors.

(44) Difficulties (LVCs)
1. Discontinuity
2. Inverted syntax (V...N - > N..V)

As was the case with MUMULS (Klyueva, Doucet, and Straka 2017), my system had some trouble with discontinuous LVCs in which several elements intervene between the verb and noun, e.g. ***fazer*** *hoje uma* ***comemoração*** '(lit.) to make today a celebration' When a noun was fronted, my model also missed some LVCs, particularly when they were discontinuous, e.g. ***concurso*** *da Mega - Sena que , será* ***realizado***... 'contest, which will be carried out...'. There were a few cases of false positives such as ***dá*** *um* ***beijo*** 'give a kiss', which does not pass tests 1, 3 and 4 above. At least two false positives ***faz*** *um escândalo* 'give a kiss', *tem deficiência mental* 'have a mental deficiency', which seem to pass all of the tests, seem to be an annotation error, cf. *teve orgasmo* 'had an orgasm' which is a LVC.

6.5.1.2 IReflV

There are eight litmus tests to determine if we are dealing with an IReflV, but the following three are usually sufficient (Savary et al. 2018) :

(45) IReflV Tests
1. Does the verb always occur with the clitic?
2. Does its sense change?
3. Does the subcategorization frame change?

If one can answer yes to any of these tests, it appears we're dealing with IReflV. It should be noted that non-inherent reflexive-like constructions such as non-idiosyncratic reflexives (e.g. *se transforma* 'transform oneself → turn into'), impersonal constructions (e.g. *aqui se fala Português* 'One speaks/Portuguese is spoken here'), passive constructions (e.g. *se construiu uma casa* 'a house was built', and reciprocal constructions (*se beijam* ‘they kiss each other’) have identical forms as the IReflVs, yet do not pass the above tests. It is unclear how a neural network learns to distinguish all of these formally identical constructions.

Affecting the identification of IReflVs is clitic syntax in Portuguese. Brazilian Portuguese utilizes two different positions, proclisis (before the verb) and enclisis (after the verb), the latter to a much lesser extent than in European Portuguese. The PARSEME corpus appears to be mostly Brazilian. However, I can find no information regarding the source texts[24]. Unfortunately, the corpus is not consistent in its annotation and some cases of enclitic pronouns attached with a hyphen are separated while others are not.

(46) Examples of proclisis (no hyphen)

1	S*e*	_	1:IReflV
2	*meterão*	_	1

(47) Examples of enclisis (hyphen)

1	Corresponder-	_	1:IReflV
2	se	_	1

but

1	refiro-me	_	1:IReflV

Like MUMULS (Klyueva, Doucet, and Straka 2017), my system had trouble with unbroken cases like *refiro-me* ‘I am referring to’, *vingar-se* ‘to take revenge’, while the separated clitic gave an unambiguous indication of the reflexive construction. In some cases, however, the broken construction was sometimes not identified, cf.. *casa- se* ‘is getting married’, which may have been confused with *casa* ‘house’.

24. The PARSEME 2018 literature reveals that this corpus is Brazilian (Ramisch et al. 2018).

6.5.1.3 ID

One would expect idioms to be problematic for any ML system. Unlike the case of LVCs and IReflVs, we do not want the system to generalize with IDs (Klyueva, Doucet, and Straka 2017). The performance of my system for IDs was not so much lower than the other categories. The most typical were false negatives, e.g. *segue o baile* ‘the dance continues’, *chamando atenção* ‘call attention’, which were simply not in the training data and thus not learned. There were some cases of false positives, such as literal *estender a mão* ‘(lit.) extend one’s hand’, in which the system undesirably has overgeneralized. The only way to prevent overgeneralization is controlling for overfitting with batch normalization or dropout as has been done.

6.6 Conclusions

In this chapter, we have focused on the task of VMWE identification. Similarly to what was observed for NER in Chapter 5, the addition of a character language model (CharLM) did not lead to immediate improvement over the baselines for this task. However, in contrast to NER, this was not due to a lack of training data, since when pretrained word embeddings, independent of the task at hand, were added to the base CharLM, results only slightly improved, indicating that task-specific word features are needed. Gains are observed when in task word and character embeddings are used, ideally separately or together. When lemma and tag embeddings are combined in the full setup, significant gains were observed (F-score 66.74), underscoring the usefulness of lemmas for capturing general word meaning and part-of-speech (PoS) tags for representing shallow syntactic features.

CHAPTER 7

Conclusions

The goal of this thesis was to examine the effect of using different embedding setups in various sequence labeling tasks. The results of these experiments were quite favorable and advanced the state of the art for Portuguese part of speech (PoS) tagging and achieved near state of the art results for the tasks of Portuguese NER and VMWE identification.

In Chapter 1, the reader was introduced to deep learning methods and architectures, as well as word embeddings and language models. In Chapter 2, we explored the use of character language models (CharLMs) in state-of-the-art sequence taggers. In Chapter 3, the deep learning architecture of my sequence tagger model was described, along with the experiments embedding setups to be used in the sequence labeling tasks. Chapter 4 presented the task of part of speech (PoS) tagging, the Portuguese data, the state of the art, the parameters of the tagging model, and the results of the experimental setups. We likewise did the same for the task of NER in Chapter 5 and VMWE identification in Chapter 6.

In all three tasks we have observed similar as well as distinct effects of using different embedding setups. For PoS tagging (Chapter 4), the use of a character language model (CharLM) by itself led to significant improvements over the baselines. This seems to be motivated by the ease of learning these shallow syntactic features, greatly aided by the presence of contextualized word representations extracted via a CharLM. When pretrained CharLM embeddings were used in conjunction with task-specific word embeddings state-of-the-art results were achieved for PoS tagging (F-score of 97.49).

The task of NER (Chapter 5) posed significant challenges, particularly due to the lack of training data. In contrast to what was observed for PoS tagging, leveraging a character language model (CharLM) by itself did not lead to immediate improvement over the baselines for NER. Here more was needed, i.e. pretrained word embeddings, which when combined with contextualized embeddings led to significant gains. It was hypothesized that using pretrained embeddings allowed the model to learn word representations not possible with task-specific word and character-level embeddings by themselves.

Similarly to in NER, in the task of VMWE identification, the use of a character language model (CharLM) also did not lead to immediate improvement over the baselines. In contrast to NER, however, this cause of this was not lack of training data. Here sufficient data allowed task-specific word and character embeddings to be learned. The presence of lemma and tag features for this dataset also made it possible to train task-specific lemma and tag embeddings, whose use led to close to state-of-the-art results for Portuguese (F-score 66.74). This underscores the importance of lemmas for capturing general word semantics and part-of-speech (PoS) tags for representing shallow syntactic features.

To conclude, the major contribution of this thesis is its thorough exploration of the use of different experimental setups with embeddings applied to sequence labeling tasks. Secondary contributions are in the advancement of the state of the art in PoS tagging for Portuguese and near state of the art results without the use of handcrafted features for Portuguese NER and VMWE identification.

Bibliography

Afonso, S., Bick, E., Haber, R., & Santos, D. (2002). Floresta sinta(c)tica: a treebank for Portuguese. *Proceedings of the Third International Conference on Language Resources and Evaluation* (LREC 2002), 1698–1703.

Akbik, A., Blythe, D.,Vollgraf, R. (2018). Contextual String Embeddings for Sequence Labeling. *Proceedings of the 27th International Conference on Computational Linguistics*, 1638-1649. Retrieved from http://aclweb.org/anthology/C18-1139

Aluísio, S., Pelizzoni, J., Marchi, A. R., de Oliveira, L., Manenti, R., & Marquiafável, V. *(2003). An account of the challenge of tagging a reference corpus for brazilian portuguese. Proceedings of the 6th international conference on Computational processing of the Portuguese language, PROPOR'03, 110–117,* Berlin, Heidelberg: Springer-Verlag.

Bengio, Y., Ducharme, R., Vincent, P., & Janvin, C. (2003). A neural probabilistic language model. *Journal of Machine Learning Research*, *3*, 1137–1155.

Camacho-Collados, J., & Pilehvar, M. T. (2018). From Word to Sense Embeddings: A Survey on Vector Representations of Meaning, 1–46. Retrieved from https://arxiv.org/abs/1805.04032

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., Koehn, P., & Robinson, T. (2014). One billion word benchmark for measuring progress in statistical language modeling. *Proceedings of the Annual Conference of the International Speech Communication Association, INTERSPEECH*, 2635–2639.

Chen, G. (2016). A Gentle Tutorial of Recurrent Neural Network with Error Backpropagation, 1–9. Retrieved from http://arxiv.org/abs/1610.02583

Chelba, C., Mikolov, T., Schuster, M., Ge, Q., Brants, T., & Koehn, P. (2014). One Billion Word Benchmark for Measuring Progress in Statistical Language Modeling. Retrieved from https://arxiv.org/abs/1312.3005

Chinchor, N. A. (1998), Proceedings of the Seventh Message Understanding Conference (MUC-7) Named Entity Task Definition. Fairfax, VA

Chiu, J. & Nichols, E. (2016). Named entity recognition with bidirectional LSTM-CNNs. *TACL*, 4, 357-370.

Collobert, R., Weston, J., Bottou, L., Karlen, M., Kavukcuoglu, K., and Kuksa, P.P. (2011). Natural language processing (almost) from scratch. *JMLR*, 12, 2493-2537.

Deng, L., & Yu, D. (2014). *Foundations and Trends in Signal Processing* (*Deep Learning Methods and Applications*), 7 (3-4).

dos Santos, C. N., Seco, N., Cardoso, N., & Vilela, R. (2006). HAREM: An Advanced NER Evaluation Contest for Portuguese. *Proceedings of the 5th International Conference on Language Resources and Evaluation, LREC 2006*, 1986–1991. Retrieved from https://pdfs.semanticscholar.org/d07c/03050ce795989123c7ec24a0ff11f039dbcf.pdf

dos Santos, C. N., & Zadrozny, B. (2014). Learning Character-level Representations for Part-of-Speech Tagging. *Proceedings of the 31st Annual International Conference on Machine Learning (ICML'14), ICML-14*(2011), 1818–1826. Retrieved from http://proceedings.mlr.press/v32/santos14.pdf

dos Santos, C. N., & Guimarães, V. (2015). Boosting Named Entity Recognition with Neural Character Embeddings, 25–33. Retrieved from https://arxiv.org/abs/1505.05008 Gehring, J., Auli, M., Grangier, D., Yarats, D., and Dauphin, Y. (2017). Convolutional sequence to sequence learning. Retrieved from https://arxiv.org/abs/1705.03122

Domingues-Fernandes, I. A. A Deep Learning Approach to Named Entity Recognition in Portuguese Texts (Master's thesis). Retrieved from https://sigarra.up.pt/feup/pt/pub_geral.show_file?pi_gdoc_id=1225188

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. Boston, MA: MIT Press.

Graves, A. (2014). Generating Sequences With Recurrent Neural Networks, 1–43. Retrieved from https://arxiv.org/abs/1308.0850

Grishman, R. & Sundheim, B. (1996). *Message Understanding Conference 6: A Brief History*. Proceedings of the 16th Conference on Computational Linguistics, 1, 466-471. San Mateo, CA: Morgan Kaufmann.

Huang, Z., Xu, W., & Yu, K. (2015). Bidirectional LSTM-CRF Models for Sequence Tagging. Retrieved from https://arxiv.org/abs/1508.01991

Jozefowicz, R., Schuster, M., Wu, Y., Com, Y. G., & Brain, G. (2015). Exploring the Limits of Language Modeling. Retrieved from https://arxiv.org/abs/1602.02410

Józefowicz, R., Vinyals, O., Schuster, M., Shazeer, N., & Wu, Y. (2016). Exploring the limits of language modeling. Retrieved from https://arxiv.org/abs/1602.02410

Klyueva, N., Doucet, A., Straka, M. (2017). Neural Networks for Multi-Word Expression Detection. Proceedings of the 13th Workshop on Multiword Expressions (MWE 2017), 60–65. ACL.

Kripke, Saul (1971). *Identity and Necessity*. New York: New York University Press.

Lafferty, J. D., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. ICML '01, 282-289.

Lample, G., Ballesteros, M., Subramanian, S., Kawakami, K.m & Dyera, C. (2016). Neural Architectures for Named Entity Recognition. Retrieved from https://arxiv.org/abs/1603.01360

LeCun, Y. (1989). Generalization and network design strategies. Technical Report CRG-TR-89-4, University of Toronto.

Liang, D., Xu, W., & Zhao, Y. (2017). Combining Word-Level and Character-Level Representations for Relation Classification of Informal Text. *Proceedings of the second conference on representation learning for NLP*, 43–47. Retrieved from http://www.aclweb.org/anthology/W17-2606

Ma, X., & Hovy, E. (2016). End-to-end Sequence Labeling via Bi-directional LSTM-CNNs-CRF. Retrieved from https://arxiv.org/abs/1603.01354

Melamud, O., Goldberger, J., & Dagan, I. (2016). context2vec: Learning Generic Context Embedding with Bidirectional LSTM. *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, 51–61. Association for Computational Linguistics Retrieved from http://www.aclweb.org/anthology/K16-1006

Mikolov, T., Sutskever, I., Chen, K., Corrado, G. S., & and Dean, J. (2013c). Distributed representations of words and phrases and their compositionality. *NIPS*, 1-9.

Peters, M. E., Ammar, W., Bhagavatula, C., & Power, R. (2017). Semi-supervised sequence tagging with bidirectional language models. Retrieved from https://arxiv.org/abs/1705.00108

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., Zettlemoyer, L. (2018). Deep contextualized word representations. Retrieved from https://arxiv.org/abs/1802.05365

Ramisch, C. et al. (2018) Edition 1.1 of the PARSEME Shared Task on Automatic Identification of Verbal Multiword Expressions. Proceedings of the Joint Workshop on Linguistic Annotation, Multiword Expressions and Constructions (LAW-MWE-CxG-2018), 222–24. Retrieved from http://aclweb.org/anthology/W18-4925

Rocha-Fonseca, E. & Rosa, J.L.G (2013). Mac-Morpho Revisited: Towards Robust Part-of-SpeechTagging. *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*, 98–107. Sociedade Brasileira de Computação.

Sag, I. A., Baldwin, T., Bond, F., Copestake, A. & Flickinger, D. (2002). Multiword Expressions: A Pain in the Neck for NLP. *Proceedings of the 3rd International Conference on Intelligent Text Processing and Computational Linguistics* (CICLing-2002), Berlin, DE, Springer.

Sang, E. F. T. K., & De Meulder, F. (2003). Introduction to the CoNLL-2003 Shared Task: Language-Independent Named Entity Recognition, 1–4. Retrieved from https://arxiv.org/abs/cs/0306050

Savary A., et al. (2018). PARSEME multilingual corpus of verbal multiword expressions. In Stella Markantonatou, Carlos Ramisch, Agata Savary & Veronika Vincze (Eds.), *Multiword expressions at length and in depth: Extended papers from the MWE 2017 workshop*, 87-147. Berlin, DE.: Language Science Press.

Schütze, H. (1998). Automatic Word Sense Discrimination. *Computational Linguistics*, *24*(1), 97-123.

Straková, J. (2017). Neural Network Based Named Entity Recognition (Doctoral dissertation). Retrieved from http://ufal.mff.cuni.cz/~strakova/doctoral_thesis.pdf

Todorovic et *al* (2008). Named entity recognition and classification using context Hidden Markov Model. *Neural Network Applications in Electrical Engineering*, 43–46. IEEE. Retrieved at https://ieeexplore.ieee.org/document/4685557

Variš, D. & Klyeva, N. (2017). Improving a Neural-based Tagger for Multiword Expression Identification. *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018).* Paris, FR: European Language Resource Association.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2017). Attention is all you need. Retrieved from https://arxiv.org/abs/1706.03762

Zhang Y., & Wallace, B. (2016). A Sensitivity Analysis of (and Practitioners' Guide to) Convolutional Neural Networks for Sentence Classification. Retrieved from arXiv:1510.03820

Zhao S. (2004). Named entity recognition in biomedical texts using an HMM model. Proceedings of the International Joint Workshop on Natural Language Processing in Biomedicine and its Applications, 84-87. Stroudsburg, PA: ACL.

Zheng, Y., Shi, Y., Guo, K., Li, W., & Zhu, L. (2017). Enhanced word embedding with multiple prototypes. *4th International Conference on Industrial Economics System and Industrial Security Engineering, IEIS 2017. Piscataway, NJ: IEEE.*

List of Figures

List of Tables

List of Abbreviations

CharLM	Character Language Model
CLE	Character Level Embedding
CNN	Convolutional Neural Network
CoNLL	Conference on Natural Language Learning
CRF	Conditional Random Field
BiGRU	Bidirectional Gated Recurrent Unit
biLM	Bidirectional Language Model
BiLSTM	Bidirectional Long Short-Term Memory
BiRNN	Bidirectional Recurrent Neural Network
BIO	Beginning (B), Inside (I) and outside (O)
BIOSE	Beginning (B), Inside (I) and Outside (O), Single (S), End (E)
GPU	Graphical Processing Unit
GRU	Gated Recurrent Unit
HMM	Hidden Markov Model
LM	Language Model
LSTM	Long Short-Term Memory
MLP	Multilayer Perceptron
MUC	Message Understanding Conference
MWE	Multiword Expression
NER	Named Entity Recognition
NLM	Neural Language Model
NLP	Natural Language Processing
NN	Neural Network
PoS	Part-of-Speech (Tagging)
ReLU	Rectified Linear Unit
RNN	Recurrent Neural Network
SGD	Stochastic Gradient Descent
MWE	Verbal Multiword Expression