A Bidirectional LSTM neural network for Named Entity Recognition

Liubov Karpova
lkarpova@uni-potsdam.de

Karina Hensel
karina.hensel@uni-potsdam.de

Abstract

This research paper closely follows the method and ideas of (Lample et al., 2016a). We adopt his innovation of the usage of no traditional features such as language-specific knowledge or resources such as gazetteers. One of their architectures is based on bidirectional LSTMs and conditional random fields. We aim to repeat their experiment, but without usage of CRF layer. We had to not use the character-based embeddings, but nevertheless achieved comparable results.

1 Introduction

Named Entities Recognition (further NER) involves locating and classifying the names in text. Those names can be attributed to one of the following pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities and similar. Recognition is often difficult partly due to the possible ambiguity of segmentation: we need to decide what's an entity and what is not. (Speech, 2019) That problem is further touched upon in the section on tagging systems.

Sequence classifiers (e.g. Maximum Entropy Markov Models (MEMM) with Conditional Random Field (CRF) or bidirectional Long Short Term Memory (bi-LSTM) with CRF)) as well as transformers are trained to label the tokens in a corpus with named entities. (Speech, 2019)

The task of NER belongs to the broad circle of problems of information extraction. This one can be solved by traditional means of computer linguistics (like hand-written regular expressions or cresting lists), by applying the methods of supervised or unsupervised learning or, finally, by combining some of those methods. Table 1 below shows how a small part of a gazetteer list labeled for location looks like.

NER type label	Entity
LOC	Berlin
LOC	Lisboa
LOC	Zurique

Table 1: An example of a gazetteer for NER classification with type specified from the corpus of (Florian et al., 2003)

Two most important features for traditional method would be word shape features and lists. Word shape features are used for representing the abstract letter pattern of the word (via regular expressions). Typical lists in the NER task include a gazetteer (a list of place names) and name-list (first names and surnames). The obvious drawback of this method is that it is very tedious to create lists for topologically rich languages. The (half-) automated creation of lists is not always possible. As a result often one does not have enough lists for some corpus and has to create them manually, which is a very tedious and long process. (Speech, 2019)

In the figure on the page below you see how the automatic process of traditional method functions. The supervised classifier uses grammatical features in order to decide about the tag. The method presented in the scheme requires a lot of training material, mostly hand-labeled.

We find that nowadays most of the authors of research papers on the topic of Named Entity Recognition are using either traditional features or some mixture of traditional and unsupervised features. We can explain the popularity of traditional features usage by the straightforwardness of using orthographic structure of the extracted words (patterns of orthographic similarity are easily captured by regular expressions or nested group of regular expressions). It is also usually relatively straightforward to find or create lists of names (for example,

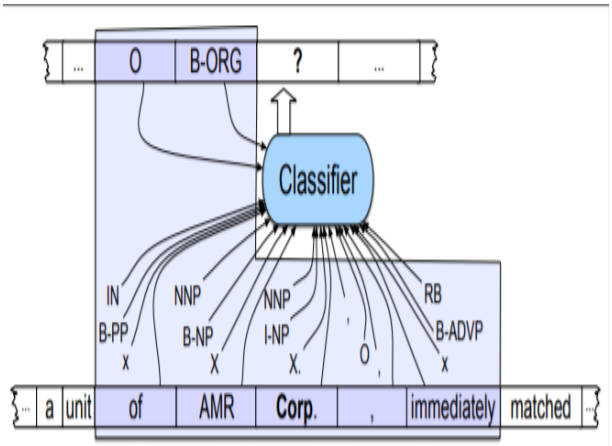

Figure 1: From (Speech, 2019) Named entity recognition as sequence labeling. The features available to the classifier during training and classification are those in the boxed area.

a list of all street names). We can hypothesize that the current popularity of the newly emerged unsupervised as opposed to supervised learning is connected with the fact that locating and labeling even small amounts of linguistic is a redundant and a very time-consuming work. This is also a reason why a mixture of traditional features that should not be labeled in the text to that extent and unsupervised features is the most popular strategy.

As the usage of traditional features is compatible with unsupervised learning methods traditional features stay popular tools even within unsupervised learning methods as they can often fine-tune the result of the training..

Mostly the systems that have extensively used unsupervised features (Collobert et al., 2011) (**?**) (Wang et al., 2009) (Ando and Zhang, 2005) have have done so in order to combine hand-engineered features (e.g., knowledge about capitalization patterns and character classes in a particular language) and specialized knowledge resources (e.g., gazetteers). See figure 2 for an example of such a studys structure. Though neural methods without usage of gazetteers come very close to the results of those (neural) methods that use gazeteers, the latter are mostly better (Luo et al., 2015)(see table 2 on the right). "Soft gazetteers there attempt to solve the problem of unavailabily of gazetteers lists in low-resource languages. "Soft gazetteers" consist of available information from English knowledge bases "into neural named entity recognition models through cross-lingual entity linking". This approach adds 4.0 points to the F1 score of the model. (Rijhwani et al., 2020) The usage of the character embeddings in (Lample et al., 2016a) adds only 0.74 points.

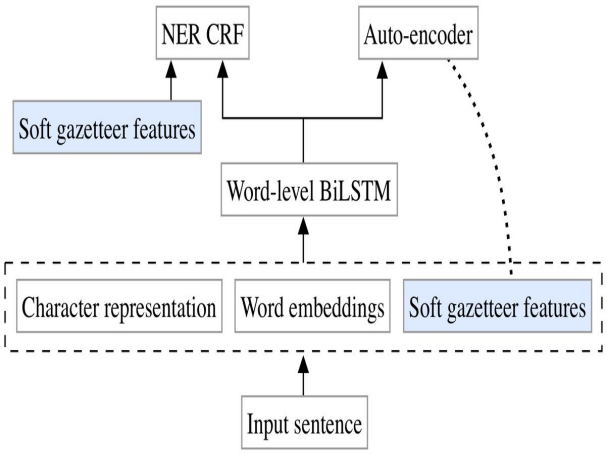

Figure 2: An example from (Rijhwani et al., 2020)

The work of paper of our focus (Lample et al., 2016b) was one of the first attempts to create a system that would rely on no specialised knowledge like gazetteers and other name lists.

In the current paper we will try to reduplicate the results of Lample using the invented by them algorithm (but with LSTM system without CRF laye)r and the same corpus of texts. The details of their and our system will be discussed in detail in the section Related work and in the section LSTM Model.

While completing this research paper we had in mind the following research questions:

- Does an additional CRF layer improve performance?
- Does character-level information increase F1-score more than pretrained word embeddings?
- Do different pretrained embeddings influence model performance?

The following section will elaborate more on the context of our research paper.

2 Related work

The result by (Florian et al., 2003) was in 2009 improved by (Luo et al., 2015) with a neural network by performing unsupervised learning on a massive unmarked corpus.

Another architecture proposed by (Collobert et al., 2011) uses a CNN on a sequence of word

overlays with a CRF layer above it. It is a bit similar to the model without character embeddings and with CNN instead of LSTM. Some studies used some hand-crafted spelling rules. (Lample et al., 2016a). Many traditional methods that use the hand-crafted rules and lists get worse (Lin and Wu, 2009) (using phrase cluster features and spelling features) or similar results (Passos et al., 2014) (using spelling features and gazetteers) when compared topure neural methods. An above-mentioned neural method implementing gazetteers(Luo et al., 2015) got the best results on the CoNll-2003 data. A neural methods using Language independent NER models also existed in the past.

To our knowledge the first system using context features was the one in the paper of Cucerzan and Yarowsky by training character-level and token-level features.(Cucerzan and Yarowsky, 2002) We plan to use the context information by using of character-based word representation combined with distributional representation. No CRF layer can make our results regarding context worse.

Also there is currently a lot of interest in models for NER that use letter-based representations. (Gillick et al., 2015) models sequence to sequence learning problem with character-based representations. Interestingly, they get low results in English (see table 2), but much better results for morphological rich languages. That might mean that the model manages to get longer and richer word forms, which is not important in English

Sometimes the researches use CNN instead of character-level embeddings. Chiu and Nichols (Chiu and Nichols, 2016) successfully learn character-level features this way. They get the resuls very similar to (Lample et al., 2016a)

To sum up, the main models (most of them were discussed above) existing in the literature and their F1 scores are presented in the table below.

Some even more recent studies that emerged after (Lample et al., 2016a) often use Transformer-based Neural Models and reach even better scores of up to **94.07 percent** F1 on Vietnamese (The et al., 2020) and for English CONLL-2003 around the one for Bi-LSTM, for example 91.10 percent like in (Liu et al., 2019) up to 91.45 on English conllu-2003 dataset. (Yan et al., 2019)

We are not going to use CRF Layer in our research, but we should mention that CRF Layer is not always used in such tasks. Discrete classifiers predict a class or label at a word. Conditional Random Fields are nor discrete classifiers, they predict labels based on not just the word, but also the context of the word. (Liu et al., 2017)

Model	**F1**
Collobert et al. (2011)*	89.59
Lin and Wu (2009)	83.78
Lin and Wu (2009)*	90.90
Huang et al. (2015)*	90.10
Passos et al. (2014)	90.05
Passos et al. (2014)*	90.90
Luo et al. (2015)* + gaz	89.9
Luo et al. (2015)* + gaz	**91.2**
Chiu and Nichols (2015)	90.69
Chiu and Nichols (2015)*	90.77
LSTM-CRF (no char)	*90.20*
LSTM-CRF	**90.94**
S-LSTM (no char)	87.96
S-LSTM	90.33

Table 2: "English NER results (CoNLL-2003 test set). * indicates models trained with the use of external labeled data" from (Lample et al., 2016b)The best results are in bold, the one that are compatible with ours in italics

Often Viterbi decoding is used after applying of Conditional Random Field. As CRFs are not predicting the best label for a word, but rather the right label sequence for a word sequence. Viterbi Decoding finds the most optimal tag sequence from the scores computed by a Conditional Random Field. (Liu et al., 2017)

Some of the studies wished to also analyse which types (tags) of the names are analyzed better and why. Here is an example of possible tags:

It results in a more complicated schema as in example below. We do not address that question specifically in the current paper, but discuss its implications for advantages and disadvantages of the system for different tag types as opposed to only one in the section on Results.

3 The task

The paper of our main interest presents neural architectures for NER that do not use traditional features or resources(except for a small amount of supervised training data and unlabeled corpora).

The authors concentrated on creating an unsupervised system that will capture two intuitions on how names differ from all other words in natural language texts. Firstly, they would differ in ortho-

Type (in bold) and Sample Categories (in cursive)	Tag
People *people, characters*	PER
Organization *companies, sports teams*	ORG
Location *regions, mountains, seas*	LOC
Geo-Political Entity *countries, states, provinces*	GPE
Facility *bridges, buidings, airports*	FAC
Vechicles *planes, trains, automobiles*	VEH

Table 3: A list of generic named entity types after (Speech, 2019)

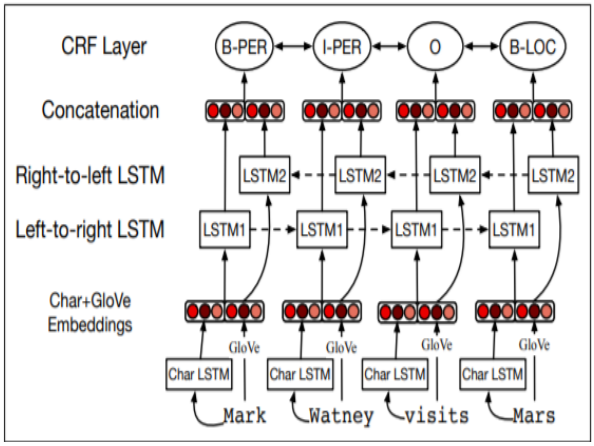

Figure 3: Bi-LTSM dealing with type tags (Speech, 2019)

graphic features (capitalization, characters distribution of names). Secondly, word-level distribution should be important in detecting the entities. (Lample et al., 2016a)

For acquiring orthographic sensitivity character-based word representation model (**?**) was adopted. For acquiring distributional sensitivity character-based word representation model was combined with distributional representations (Mikolov et al., 2013). We will concentrate more on the structure of the model in the following section after explaining the labeling scheme in the subsection called tagging system.

Below we provide the table taken from the paper of our main interest. It shows all the model variants and the results that they received. Components of the variants are deciphered in the following way: . "pretrain" stands for models that include pre-trained word embeddings, "char" represents models that include character-based modeling of words, the mark "dropout" means that the models that have it include dropout rate. Dropout rate is used here to encourage the model to learn to trust both sources of evidence: not only pretrained, but also character-based modelings. The authors (Lample et al., 2016a) observed a significant improvement in their model's performance after using dropout. They got the best results in a model version which used all the three conponents (pretrained embeddings, character embeddings and dropout rate). Using of the dropout rate also improves the pretrained only model.

4 Tagging system

The standard algorithm in NER is a labeling task which is carried out word-by-word. In the process the assigned tags capture the boundaries.One has to come up with the optimal tagging system for the tagging problem.

In information extraction practice there exists several tagging systems which help us to label the location of the information. The corpus we planned to use exists in two versions: IOB and IOBES.

IOB (short for inside, outside, beginning) and IOBES (short for inside, outside, beginning,end of entity, singleton entity) are tagging systems for NER locating. B stands for the beginning of a chunk, I - tag is inside of the chunk, 0 - belongs to a chunk, S - singleton entity, E - end of the entity. The latter IOBES system was made in order to better locate singleton entities and end points of the chunk, which are not tagged in the IOB system.

If you look at the table 5, you will see that each row corresponds to one token. For each token, there is a word, a part-of-speech tag, a "shallow syntax" label, and the BIO-coded named entity label, separated by whitespace. The sentences are separated by empty lines. Above in the table 5 you see an example of the sentence labeled in the IOBES and IOB tagging schemes. The sentence itself is demonstrated in the ConLL 2003 format and is directly taken from the corpus we use for training..

One should consider several important points in order to choose the most suitable tagging system. To begin with, as names often consist of several tokens, reasoning jointly over tagging decisions for each token is important. Also, the compatibility

Model	Variant	F1
LSTM (our baseline)	char+dropout+pretrain	89.15 (no CRF layer)
LSTM-CRF	char + dropout	83.63
LSTM-CRF	pretrain	88.39
LSTM-CRF	pretrain +char	89.77
LSTM-CRF	pretrain + dropout	90.20
LSTM-CRFr	pretrain + dropout + char	**90.94**

Table 4: Performance of the models from (Lample et al., 2016a)

IOBES	IOB
EU S-ORG	EU I-ORG
rejects O	rejects O
German S-MISC	German I-MISC
call O	call O
to O	to O
boycott O	boycott O
British S-MISC	British I-MISC
lamb O	lamb O
. O	. O

Table 5: Tagging systems. Tagging and labelin in CoNLL/2003

with other studies decides which tagging system is used.

The researches in the (Lample et al., 2016a) paper chose IOBES system. Some researchers (for example, (Ratinov and Roth, 2009)) showed that using a scheme like IOBES improves model performance because it provides more information that turns out to be significant. The authors of the paper that we try to reduplicate did not receive a significant improvement over the traditional IOB system, but we decided to still use it thus assuring more compatibility between our studies.

5 Bidirectional LSTM model

Despite it is possible to train a RNN (Recurrent Neural Network) to perform this task, it is simultaneously difficult, because such a task requires a network to use information that is located distantly in the text. One can get a confirmation of that fact looking at the results of applying the bidirectional RNN to the CoNLL-2003 corpus. The F1 score remains lower than while using other methods: **85.14**. (Johansson)

Long Short-term Memory Networks (LSTMs) have been designed as a gated variant of RNN to overcome one of the most significant drawbacks of the RNNs (Recurrent Neural Networks). The latter are too sequential in the choice of their predictions (see the figure 4 below for the schema of the difference between sequential and non-sequential predictions, the former predicts the next element in the sequence, the latter predicts the tag of the current word, which makes them ideal for different tasks) , and tend to handle long-term dependencies not well. It was desired and implemented in a way that the RNN could decide if the update is optional, or if the update happens, by how much etc.(Hochreiter and Schmidhuber, 1997)

LSTM enables not only conditional updates, but also forgetting of the values in the previous hidden state. The forgetting is the product of the previous hidden state value with a gating function (the function produces the value between 0 and 1 depending on its input).(Hochreiter and Schmidhuber, 1997)

LSTMs incorporates a memory-cell and can handle long-range dependencies better than RNN. In the paper of our main interest it was implemented after (Hochreiter and Schmidhuber, 1997). Several gates were used in order to control the proportion of the input to give to the memory cell, and also the proportion from the previous state to forget.

A deep neural network with gradient based learning and backpropagation requires finding partial derivatives, and for that we have to traverse the network from the the final layer to the initial layer. By the chain rule deeper layers go through continuous matrix multiplications for computing their derivatives. If there is n layers, there will be n derivatives to be multiplied together. This results in too small or too large gradients. (Pykes)

Vanishing (too small) and exploding (too large) gradient problems in RNNs also motivate the emergence of LSTMs. Vanishing or exploding gradients are gradients that can get out of control (it depends whether the direction of the absolute values of the gradients are shrinking or growing respectively). Both great absolute value of the gradient or a tiny one (less than 1) value is able to make the optimiza-

tion procedure unstable. (Hochreiter and Schmidhuber, 1997)

Large derivatives will increase the gradient exponentially while propagating down the model until they eventually explode. Small derivatives will decrease the gradient exponentially as we propagate through the model until it eventually vanishes. (Pykes)

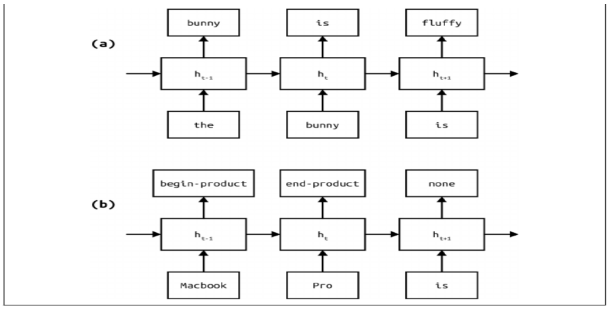

Figure 4: Taken from (Rao and McMahan, 2019), a is sequential language modeling, "b" is NER

Bidirectional mode of LSTM is in fact 2 LSTMs starting at different text points. In the bidirectional LSTM the learning algorithm is fed with the original data once from beginning to the end and once from end to beginning of the text. In the paper word and character embeddings are computed for the input word before being directed to the bi-LSTM.

In their "Neural Arcitectures for Named Entity Recognition" (Lample et al., 2016a) Lample and colleagues implemented two neural networks to perform the task of named entity recognition: a stacked Long Short Term Memory (LSTM) network, which uses an approach similar to transition-based dependency parsing, and a bidirectional LSTM (BiLSTM) with an additional conditional random field (CRF) layer which is shown in Figure 7. In this project we will also implement a variation of the latter but without the CRF layer.

As one can see from both 5th and 6th figures the main model takes character embeddings (produced using yet another BiLSTM in a preceding step) and pre-trained word embeddings, which are concatenated together of each token in the input sequence. Concatenated embeddings are given as final embedding to the main bi-LSTM.

Similarly, the final conditional random field mechanism following the LSTM layer can model the single, token-based tagging decisions for the entire sequence jointly. The authors hypothesize that this is particularly useful because there are hard constraints when labelling named entities ac-

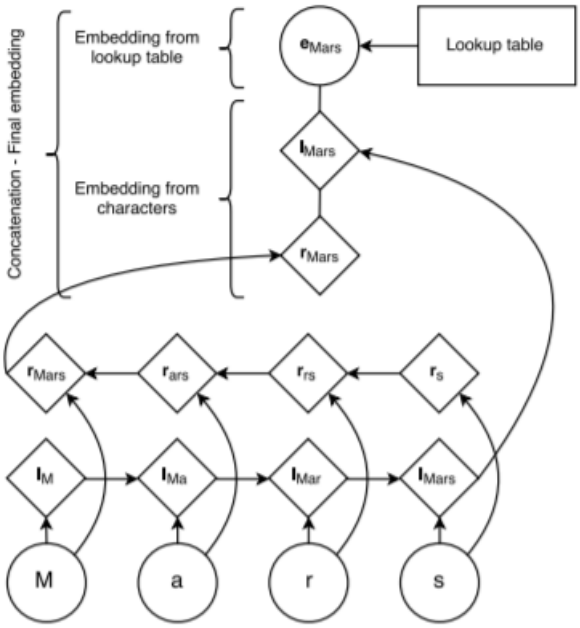

Figure 5: Architecture to generate a word embedding for a word from its characters and to train it via LSTM as proposed by (Lample et al., 2016a) "The character embeddings of the word "Mars" are given to a bidirectional LSTMs. We concatenate their last outputs to an embedding from a lookup table to obtain a representation for this word."

cording to the IOBES schema. For instance, in a multi-word name a word labelled *I-X* can only be followed by a word with the label *I-X* or *E-X*, not *I-Y* or *O-X*. The output of the neural network is the tag sequence with the highest probability given the input sentence.

As already mentioned, in this project we have implemented the bidirectional LSTM neural network as proposed by (Lample et al., 2016a).Theoretically the LSTM output layer can be directly passed to the softmax that creates a probability distribution over all NER tags, after what the most likely tag is chosen. That is the alternative that we have implemented instead of giving the output to the CRF layer.

Mathematic description where x is the input vector of the softmax function looks the following way:

$$Softmax(x_i) = \frac{\exp(x_i)}{\sum_j \exp(x_j)}$$

The softmax function is a function that turns a vector of K real values into a vector of K real values that sum to 1, those real values stand for probabilities of the tag being the real tag. As we use the

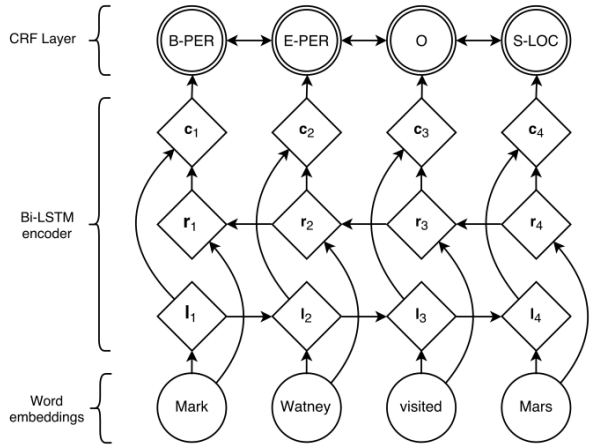

Figure 6: Main architecture as proposed by (Lample et al., 2016a)*"Word embeddings are given to a bidirectional LSTM. l_i represents the word i and its left context, r_i represents the word i and its right context.Concatenating these two vectors yields a representation of the word i in its context, c_i."*

softmax function in a machine learning model, we had in mind that it has a tendency to produce values very close to 0 or 1 while interpreting them as probabilities. The term on the bottom of the sostmax formula is a normalization term, it ensures that all the output values of the function will sum to 1 and each be in the range (0, 1). The figure 7 below shows how the softmax works graphically. (Wood) After that one calculates the best tag. Pytorch makes both steps for us.

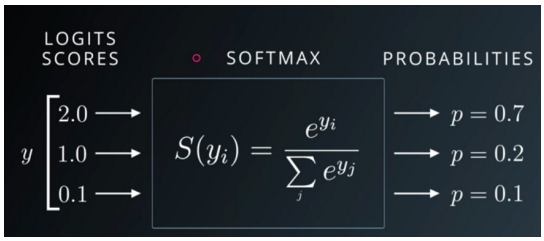

Figure 7: Softmax function takes vector y as output and turns each element to probability score from explanatory blogpost (uniqtech)

In our implementation after pytorch specifications the input sentence $w_i \in W$ belongs to our vocabulary. T is the tag set, and y_i is the tag of word w_i.

For the prediction, an LSTM is passed over the sentence. The hidden state at timestep i is h_i. Also, each tag has a unique index. Finally we come to the prediction rule for y_i is

$$y_i = argmax_j(logSoftmax(Ah_i + b))_j$$

The log softmax of the affine map of the hidden state is taken, and the predicted tag y is the best fit according to the argmax function, it is the tag that has the maximum value in this vector. (Speech, 2019) Pytorch implements both last formulas with one command.

The last step and the only planned significant differnece between our system and the system of (Lample et al., 2016a) is the absense of CRF layer in our implementation. It must be noted that this implementation differs slightly from the sequence-wise prediction in the original model. There the probability of a label sequence assigned to a complete sentence, not the single token is maximized. CRF is the factor that ensures that: it takes into account neighboring tags, yielding the final predictions for every word, which assures more contextual information.

The motivation behind the modifications mentioned above is to observe the effect of character-level information as well as conditional random fields on the quality of the model output.

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

Figure 8: Torch implementation of LSTM from Pytorch website

$$\begin{aligned}
\mathbf{i}_t &= \sigma(\mathbf{W}_{xi}\mathbf{x}_t + \mathbf{W}_{hi}\mathbf{h}_{t-1} + \mathbf{W}_{ci}\mathbf{c}_{t-1} + \mathbf{b}_i) \\
\mathbf{c}_t &= (1 - \mathbf{i}_t) \odot \mathbf{c}_{t-1} + \\
&\qquad \mathbf{i}_t \odot \tanh(\mathbf{W}_{xc}\mathbf{x}_t + \mathbf{W}_{hc}\mathbf{h}_{t-1} + \mathbf{b}_c) \\
\mathbf{o}_t &= \sigma(\mathbf{W}_{xo}\mathbf{x}_t + \mathbf{W}_{ho}\mathbf{h}_{t-1} + \mathbf{W}_{co}\mathbf{c}_t + \mathbf{b}_o) \\
\mathbf{h}_t &= \mathbf{o}_t \odot \tanh(\mathbf{c}_t),
\end{aligned}$$

Figure 9: Authors (Lample et al., 2016a)implementation of LSTM after (Hochreiter and Schmidhuber, 1997)

The main feature of the LSTMs is gates. Gates can regulate the flow of information. As the cell state goes on its journey, information get's added or removed to the cell state via gates. The gates are different neural networks that decide which information is allowed on the cell state.The name gates is chosen because the sigmoid function squashes the values of these vectors between 0 and 1. When we multiply the gates elementwise with another vector we define how much of that other vector we want to "let through". The input gate defines how much of the newly computed state for the current input you want to let through. The forget gate defines how much of the previous state you want to let through. Finally, The output gate defines how much of the internal state you want to expose to the external network (higher layers and the next time step). All the gates have the same dimensions , the size of your hidden state.(Phi) (Hochreiter and Schmidhuber, 1997)

Next we are going to look at the mathematical representations of LSTMs in the paper of our interest and in our implementation (pytorch implementation). Below you can see the mathematical presentation of the LSTM implementations that are implemented in Pytorch and in the paper. W is weight matrix, b is bias, σ is the element-wise sigmoid function, and $\odot$ is the element-wise product. h_t is the hidden state at time t, in the case of bidirectional LSTM $h_t = [\overrightarrow{h_t}; \overleftarrow{h_t}]$.

As for other parameteres c_t is the cell state at time t, x_t is the input at time t, t-1 is also time at point 0, i_t,f_t, g_t,o_t are the input, forget, cell, and output gates, respectively. The tanh activation is used to help regulate the values flowing through the network. The tanh function squishes values to always be between -1 and 1.

One immediately sees that the presentations (mostly the definitions) differ slightly.

As one sees the formal presentation of the papers lacks the equation for forget gate and cell gate. Cell gate in the paper is incorporated in the cell state because cell gate is only used in the cell state and probably in order to make the equation system and running time smaller. The cell gate (in Pytorch implemented as a separate gate) just takes the tahn function of the input. But there are differences between two realisations of this process. Instead of input gate and hidden state of cell gate in Pytorch implementation there are input cell and hidden state of cell state. Authors define bias in one, not two bias vectors. Though of course the idea stay the same: g is a "candidate" hidden state that is computed based on the current input and the previous hidden state. Again, input gate in the papers implementation uses more weight matrices, but less bias vectors. Output gate also has there more matrices and one bias vector (which does not directly count for the input and hidden states, but ony for the current one).The forget function as already mentioned is not formally defined, but nested in the cell state. It is described not via weight matrices and vectors like input and output gates, but as $1 - i_t$.

The pseudocode for the bidirectional LSTM after (Hochreiter and Schmidhuber, 1997) (Graves and Schmidhuber, 2005) would be the following:

"Forward pass

- Feed all input data for the sequence into the BLSTM and determine all predicted outputs.
  - Do forward pass just for forward states (from time t_0 to t_1) and backward states (from time t_1 to t_0).
  - Do forward pass for output layer.

Backward pass

- Calculate the error function derivative for the sequence used in the forward pass.
  - Do backward pass for output neurons.
  - Do backward pass just for forward states (from time t_1 to t_0) and backward states (from time t_0 to t_1).
- Update weights."

5.1 Corpus

The shared task of CoNLL-2003 concerns language-independent named entity recognition. Its data files contain four columns separated by a single space. Each word is on its separate line. The first item on each line is a word, the second a part-of-speech (POS) tag, the third a syntactic chunk tag and the fourth is the named entity tag. The example of the IOB and IOBES versions of the corpus were already given below in the section Tagging system, see table 5 for the sentence example.

Below you see the labeled sentence from the IOBES CoNLL-2003 corpus, which is also a sentence from news:

Only RB B-NP O France NNP I-NP B-LOC and CC I-NP O Britain NNP I-NP B-LOC backed VBD B-VP O Fischler NNP B-NP B-PER 's POS B-NP O proposal NN I-NP O . . O O

The CoNLL 2003 data that we worked with is a NER benchmark data set based on Reuters news data. It provides both training, validation and testing sets. Let us look at the studies that used the same (or sometimes fuller) data in order to underline how few data is used for the system that we try to reduplicate with some minor changes.

One of the best systems that was presented by (Florian et al., 2003) at the NER CoNLL 2003 challenge achieved 88.76 percent F1 score. A combination of various machine-learning classifiers using features such as words, POS tags, CHUNK tags, prefixes and suffixes, a large gazetteer (not provided by the challenge), as well as the output of two other NER classifiers trained on richer data sets.

5.2 Embeddings

As already mentioned a difference to the original network is that we do not use character embeddings but only word vectors as input to our model. These are 100-dimensional GloVe embeddings pretrained on the English Wikipedia 2014 and Gigaword 5th Edition corpora (6 billion tokens, 400,000 words)(Pennington et al., 2014)[1]. GloVe is an unsupervised learning algorithm which obtains vector representations for words. They are trained on aggregated global word-word co-occurrence statistics from a corpus.

We chose to use those embeddings because of the availability and performance reasons. The authors (Lample et al., 2016a) pre-trained their word embeddings themselves on the English Gigaword version 4. We used the pre-trained embeddings train on the next version of the data, so they our embeddings should be comparable to theirs, probably slightly better due to a fresher version.

The vocabulary of our pretrained GloVe embeddings contains millions of words, but there are always some new words, so for the case of encountering an out-of-vocabulary word it is assigned in GloVe with some random vector values. That is sometimes criticised as a process that can confuse a model. (Antonio) Using character embeddings is a good way to solve the problem. Another methods include using translation vector or using prediction (weighted average of the predicted words' embedding).

[1] https://www.kaggle.com/rtatman/glove-global-vectors-for-word-representation?select=glove.6B.100d.txt

5.3 Character embeddings

We stress again that we did not create character embeddings for the current study, but we would like to briefly explain them in order to contrast the differences in our paper and the paper of (Lample et al., 2016a) and to discuss the benefit of using character embeddings alongside with the word embeddings.

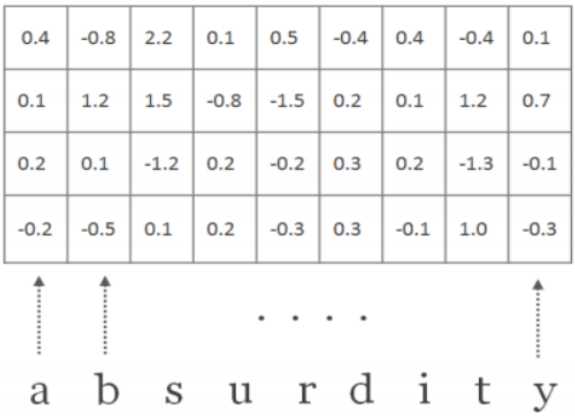

Figure 10: Character embeddings in a matrix from (Antonio)

Above in the figure 9 you can see the character embeddings in a matrix form where the whole matrix is a word representation and each vector is a character representation.

As already mentioned, out-of-vocabulary words are well-handled by using character embeddings . This way all the missspelled words, new words, emoticons, symbols are better handeled by the model.

Character-level features were used while training instead of hand-engineering prefix and suffix information about words. They have been found useful for morphologically rich languages and made a significant change there. English might be not the best language to show the effect of the character embeddings innovation. Still as the languages with a greater affixes variation seem to perform much worse than English we would like to see the results of character based models application to different languages, as the existence of both right-to-left or left-to-right LSTM is more beneficial for agglutinative suffixing or prefixing languages or languages with poor morphology(like English which is examined in current paper), and German, Dutch and

Spanish are more fusional.

Authors (Lample et al., 2016a) expect the final representation of the forward LSTM to be" an accurate representation of the suffix of the word", and the final state of the backward LSTM to be "a better representation of its prefix". It is interesting why the application of the model to the more morphologically rich languages like Spanish and German does not yield greater difference in model variants with and without character embeddings. It stays unclear whether it has something to do with the corpus data, with the fact that character do not bring much new information except for inclusion of OOV words or with the fact of the above-mentioned languages being fusional.

Figure 5 shows how character embeddings (word embedding from words character, one for each character in a word). A character lookup table with an embedding for every character is initialized at random. Those character embeddings are given to the bidirectional LSTM (forward embedding to the forward LSTM, backward emmbedding to the backward LSTM). As forward and backward representations are concatenated we get the embedding for a word . This character-level representation is then concatenated with a word-level representation from a word lookup-table. If words do not have an embedding in the lookup table they are mapped to a UNK embedding. (Lample et al., 2016a)

Character embeddings are mostly created in the following manner:a list of characters including numbers, special characters and unknown character (UNK) is created. They are transferred as 1-hot encoding. (Ma)

Word and character embeddings are not the only embeddings used in the NLP on the word level. Facebook AI Research team use subwords to train their models. So if we take the word "linguistics", "l", "li", "lin", "ling" and so on are used. (Ma)

5.4 Architecture

The model contains of two main parts: creating of character embeddings by one-hot vector creation and biLSTM usage, and subsequent training of concatenated word and character embeddings where biLSTM is again used. Dropout is applied in the final Bi-LSTM in order to depend on both charactetr and word representations, but it is also technically used while running the network with only one embedding type for evaluation purposes. (Lample et al., 2016a) It is possible to calculate the final loss

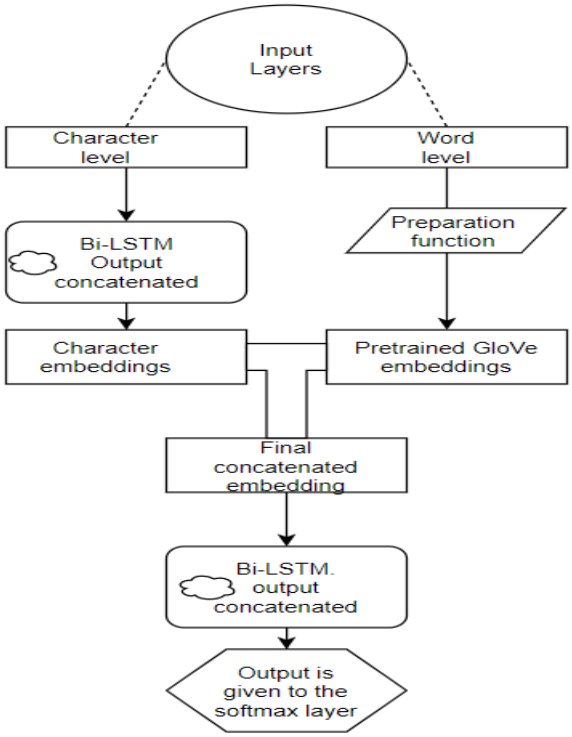

Figure 11: Algorithm for the current paper without CRF layer

of the whole NER labeling. The authors do not explicitly mention how and if they calculate the loss, but it is said (sgrvinod) that the total loss in the process of multi-task learning is a linear combination of the losses on the individual tasks. The parameters of the combination can be fixed or learned as "updateable" weights. So theoretically we could compute 5 losses: one for each LSTM in both Bi-LSTMs and one total loss (e.g. by summing four existing losses).

A character lookup table initialized at random contains an embedding for every character. The embedding for a word derived from its characters is the concatenation of its forward and backward representations from the bidirectional LSTM. This character-level representation is then concatenated with a word-level representation from a word lookup-table. (Lample et al., 2016a)

The dropout mask to the final embedding layer is applied just before the input to the bidirectional LSTM. Final concatenated representations are linearly projected onto a layer whose size is equal to the number of distinct tags, and softmax function is applied to that output. (Lample et al., 2016a)

The model is build in Python using the Pytorch (Paszke et al., 2019). The initial embedding layer

is a matrix of the dimensions 400,000 x 100. It is followed by a BiLSTM layer with 100 hidden units which maps its input to the 9 possible named entity classes (based on the labels in the English training data of the CoNLL 2003 shared task (Tjong Kim Sang and De Meulder, 2003)).
Additionally, there is a dropout layer after the embedding layer.

All in all we tried to keep the parameters as close to the original setting as possible to ensure optimal comparability. The only difference is that we do not use a character-level representation of the input and do not add a CRF layer after the LSTM. Due to the missing CRF layer our prediction method is slightly different to the original approach, not maximizing the probability of a tag sequence predicted for an entire sentence but only for single words.

6 Training

As for the training procedure the parameters remained unchanged compared to the setting used by (Lample et al., 2016a). We run training for 50 epochs using the back-propagation algorithm. Parameters are updated on every training sentence using stochastic gradient descent (SGD). To measure the error we use the negative log likelihood loss as the loss function. A complete list of hyperparameter values can be found in table 6.

Hyperparameter	Value
Epochs	50
Dropout	0.5
Learning rate	0.01
Gradient clipping	5
Momentum	0.6

Table 6: Hyperparameter values

7 Results

To ensure optimal comparability we trained and evaluated our neural network on the same data set as (Lample et al., 2016a), namely the English test data of the CoNLL-2003 shared task (Tjong Kim Sang and De Meulder, 2003). Moreover, the same metric (f1 score as implemented in the sklearn.metrics package (Pedregosa et al., 2011)) is used. Initially, we were planning to use the same baseline model as (Lample et al., 2016a) (character and word embeddings, dropout, no CRF layer). However, as it was not possible due to time constraints to also train character-level embeddings the basic model in this project only uses pre-trained word embeddings without applying dropout. Table 7 presents the results of the original model and our implementation.
A table (8) with the accuracy on the test set can be found in the appendix of this paper.

8 Evaluation and Discussion

As one can see from the scores in table 7 our models achieved a performance similar to the original neural network without CRF (LSTM(our baseline) in the table 4). Even though their model without CRF has both character and word embeddings (89.15 original vs. 89.42 ours). This shows that using pre-trained word embeddings but no character embeddings does not change much in the results significantly. That means that character embeddings do not change much, as they are the only different parameter in the papers and ours models. That is consistent with the papers F1 scores: adding character embeddings only change the F1 performance from 90.20 to 90.94. Interestingly we did not need character embeddings for dealing with out-of-character words.

One could suggest that we have similar results due to the new corpus version. That does not sound like a plausible version to us, but that might be the case. Theoretically it is possible that differences in LSTM implementation discussed in the fifth section add performance to the pytorch model.

Moreover, we were not able to train the neural network with different pre-trained embeddings as planned initially. It would also have been interesting to observe whether embeddings obtained from corpora of various genres or of different dimensions influence the final output. In the future one could also compare the differences of corpus versions used for creating of word vectors. That will also close the debate whether our full mode model benefits from newer corpus.

It would be interesting to see whether the adding of CRF layer would change much in the performance of a model. We suggest it would change a lot, because in (Lample et al., 2016a) it changed the result from 89.15 to 90.94.

Also pre-trained embeddings with dropout perform slightly better in their version with CRF than in our version without CRF layer. That also means that CRF layer alters a lot. Previous research (Lam-

Input features	(Lample et al., 2016a)	Our model
pretrain	88.39 (with CRF layer)	88.23
pretrain + dropout	90.20 (with CRF layer)	89.42
char + dropout + pretrain	89.15 (no CRF layer)	-
char + dropout	83.63 (with CRF layer)	-
pretrain + char	89.77 (with CRF layer)	-
pretrain + dropout + char	**90.94** (with CRF layer)	-

Table 7: F1 scores of the original and our models on the English test set data of the CoNLL-2003 shared task

ple et al., 2016a) use a CRF to take into account neighboring tags. We suggest that this accounts for the change to the better performance.

Moreover, the effect of dropout is negligible as well. This is not surprising because as the purpose of dropout is to prevent the model from relying too much on one feature as a source of information but with the usage of solemnly word embeddings without character embeddings only one type of information was provided anyway. It is convenient to use dropout effect with separately either pretrained or character embeddings in order to distinguish between main model types, but as mentioned before our model situation did not allow for that.

9 Conclusion

- Although we did not manage to pursue all of our initial research question, we have build a working model which achieved almost the same results
- context plays a role for NER and it remains a question whether and how much word character embeddings add to the big picture. One can hypothesize whether some languages or some concrete types of the names (for example street names as a place entity or names that depend on the context the least) benefit from using them more than others.

Now we would like to directly answer our research questions.

- Does an additional CRF layer improve performance? We suggest that it does, we can say that based on the results from the paper of our interest and based on the different
- Does character-level information increase F1-score more than pretrained word embeddings? We suggest that it does not.
- Do different pretrained embeddings influence model performance?We cannot be sure, but it is possible that different pretrained embeddings influenced model performance in our case.

As further suggestions of the changes to the model we would follow Facebook AI research team and build a system which has character ngram embeddings. Also, training a CNN instead of character embeddings gives good results. Moreover, one could apply character embeddings only for the tag types (say, post index or street names) and languages (morphologically rich) when it is needed.

References

Rie Kubota Ando and Tong Zhang. 2005. A framework for learning predictive structures from multiple tasks and unlabeled data. *Journal of Machine Learning Research*, 6(Nov):1817–1853.

Meraldo Antonio. Word embedding, character embedding and contextual embedding in bidaf — an illustrated guide.

Jason PC Chiu and Eric Nichols. 2016. Named entity recognition with bidirectional lstm-cnns. *Transactions of the Association for Computational Linguistics*, 4:357–370.

Ronan Collobert, Jason Weston, Léon Bottou, Michael Karlen, Koray Kavukcuoglu, and Pavel Kuksa. 2011. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(ARTICLE):2493–2537.

Silviu Cucerzan and David Yarowsky. 2002. Language independent NER using a unified model of internal and contextual evidence. In *COLING-02: The 6th Conference on Natural Language Learning 2002 (CoNLL-2002)*.

Radu Florian, Abe Ittycheriah, Hongyan Jing, and Tong Zhang. 2003. Named entity recognition through classifier combination. In *Proceedings of the seventh conference on Natural language learning at HLT-NAACL 2003*, pages 168–171.

Dan Gillick, Cliff Brunk, Oriol Vinyals, and Amarnag Subramanya. 2015. Multilingual language processing from bytes. *arXiv preprint arXiv:1512.00103*.

Alex Graves and Jürgen Schmidhuber. 2005. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural networks*, 18(5-6):602–610.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. *Neural computation*, 9(8):1735–1780.

Richard Johansson. Sequence tagging example, from machine learning for natural language processing (phd-level course; september – december, 2019).

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016a. Neural architectures for named entity recognition. In *Proceedings of the 2016 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 260–270, San Diego, California. Association for Computational Linguistics.

Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. 2016b. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*.

Dekang Lin and Xiaoyun Wu. 2009. Phrase clustering for discriminative learning. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP*, pages 1030–1038.

Liyuan Liu, Jingbo Shang, Frank F Xu, Xiang Ren, Huan Gui, Jian Peng, and Jiawei Han. 2017. Empower sequence labeling with task-aware neural language model. *arXiv preprint arXiv:1709.04109*.

Pengfei Liu, Shuaichen Chang, Xuanjing Huang, Jian Tang, and Jackie Chi Kit Cheung. 2019. Contextualized non-local neural networks for sequence learning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, pages 6762–6769.

Gang Luo, Xiaojiang Huang, Chin-Yew Lin, and Zaiqing Nie. 2015. Joint entity recognition and disambiguation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, pages 879–888.

Edward Ma. Besides word embedding, why you need to know character embedding?

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119.

Alexandre Passos, Vineet Kumar, and Andrew McCallum. 2014. Lexicon infused phrase embeddings for named entity resolution. *arXiv preprint arXiv:1404.5367*.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. dAlché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc.

F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Michael Phi. Illustrated guide to lstm's and gru's: A step by step explanation.

Kurtis Pykes. The vanishing/exploding gradient problem in deep neural networks.

Delip Rao and Brian McMahan. 2019. *Natural language processing with PyTorch: build intelligent language applications using deep learning*. " O'Reilly Media, Inc.".

Lev Ratinov and Dan Roth. 2009. Design challenges and misconceptions in named entity recognition. In *Proceedings of the Thirteenth Conference on Computational Natural Language Learning (CoNLL-2009)*, pages 147–155.

Shruti Rijhwani, Shuyan Zhou, Graham Neubig, and Jaime Carbonell. 2020. Soft gazetteers for low-resource named entity recognition. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 8118–8123, Online. Association for Computational Linguistics.

sgrvinod. pytorch-sequence-tagger?

D Jurafsky Speech. 2019. Language processing: An introduction to natural language processing. *Computational Linguistics, and Speech Recognition/Daniel Jurafsky, James H. Martin//Prentice Hall PTR Upper Saddle River, NJ,–2000*.

Viet Bui The, Oanh Tran Thi, and Phuong Le-Hong. 2020. Improving sequence tagging for vietnamese text using transformer-based neural models. *arXiv preprint arXiv:2006.15994*.

Erik F. Tjong Kim Sang and Fien De Meulder. 2003. Introduction to the conll-2003 shared task: Language-independent named entity recognition. In *Proceedings of the Seventh Conference on Natural Language Learning at HLT-NAACL 2003 - Volume 4*, CONLL '03, page 142–147, USA. Association for Computational Linguistics.

uniqtech. Understand the softmax function in minutes.

Shu-Ping Wang, Wen-Lung Wang, Yih-Leong Chang, Chen-Tu Wu, Yu-Chih Chao, Shih-Han Kao, Ang Yuan, Chung-Wu Lin, Shuenn-Chen Yang, Wing-Kai Chan, et al. 2009. p53 controls cancer cell invasion by inducing the mdm2-mediated degradation of slug. *Nature cell biology*, 11(6):694–704.

Thomas Wood. What is a softmax function?

Hang Yan, Bocao Deng, Xiaonan Li, and Xipeng Qiu. 2019. Tener: Adapting transformer encoder for name entity recognition. *arXiv preprint arXiv:1911.04474*.

A Appendices

Appendix A. Test set accuracy.

Input features	Accuracy
pretrain	89.22
pretrain + dropout	89.93

Table 8: Accuracy scores of our model on the CoNLL-2003 English test set

B Supplemental Material

Running instructions The code used in this project can be found in the following GitHub repository: https://github.com/karina-hensel/sose20-deep-nlp-Named-Entity-Recognition.

The saved model files can be downloaded from the following Kaggle repository: https://www.kaggle.com/karinahensel/ner-trainedlstms.

To verify the results download the GitHub repository and the model files. Then run the training script using the command:

```
python3 train.py PATH-TO-EMBEDDINGS-
FILE PATH-TO-CONLL-DATASET NUMBER-
EPOCHS DROPOUT MODEL-FILE-NAME
```

Do make sure that NUMBER-EPOCHS=50 and DROPOUT=0.5 to exactly replicate the experiments lined out in this paper. Furthermore, the script only runs with data sets in the CoNLL format(Tjong Kim Sang and De Meulder, 2003) and GloVE word embeddings (Pennington et al., 2014). Also all necessary dependencies need to be installed on your system.

Running the training script will generate the trained models and save them to files.
For evaluation run the provided evaluation script as follows:

```
python3 evaluate.py PATH-TO-
EMBEDDINGS-FILE PATH-TO-CONLL-
DATASET MODEL-FILE-NAME
```

Another way to replicate the results is to run the Jupyter notebook available in the same Kaggle repository as the model files, which is linked above.