Czech Technical University in Prague
Faculty of Electrical Engineering

Department of Cybernetics

BACHELOR PROJECT ASSIGNMENT

Student:	Hoang Long N g u y e n
Study programme:	Open Informatics
Specialisation:	Computer and Information Science
Title of Bachelor Project:	Named Entity Recognition Using Recurrent Neural Networks

Guidelines:

1. Research current state of the art methods for Named Entity Recognition in Czech and English.
2. Propose a suitable recurrent neural network architecture based on the previous research.
3. Implement the neural network in Tensorflow for sequence labelling.
4. Perform experiments on several datasets in both languages and measure results.
5. Discuss your results and compare them with state of the art.

Bibliography/Sources:

[1] CHIU, Jason PC; NICHOLS, Eric. Named entity recognition with bidirectional lstm-cnns. arXiv preprint arXiv:1511.08308, 2015.
[2] STRAKOVÁ, Jana; STRAKA, Milan; HAJIČ, Jan. Neural Networks for Featureless Named Entity Recognition in Czech. In: International Conference on Text, Speech, and Dialogue. Springer International Publishing, p. 173-181, 2016.

Bachelor Project Supervisor: Ing. Jan Pichl

Valid until: the end of the summer semester of academic year 2017/2018

L.S.

prof. Dr. Ing. Jan Kybic
Head of Department

prof. Ing. Pavel Ripka, CSc.
Dean

Prague, January 13, 2017

Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Cybernetics

Bachelor's Thesis

Named Entity Recognition Using Recurrent Neural Networks

Long Hoang Nguyen

Supervisor: Ing. Jan Pichl

Study Programme: Open Informatics

Field of Study: Computer and Information Science

May 25, 2017

Aknowledgements

I would like to thank my great mentor Ing. Jan Pichl for never giving up on me, Bc. Jakub Hajič for always questioning my knowledge and Gareth Dwyer, BA (Hons), for his great support. Finally, I want to thank the whole team at eClub Prague lead by Ing. Jan Šedivý, CSc. for their inspiring work every single day.

Author statement for undergraduate thesis

I declare that the presented work was developed independently and that I have listed all sources of information used within it in accordance with the methodical instruction for observing the ethical principles in the preparation of university theses.

In Prague, on May 24, 2017 ....

Abstract

Named entity recognition is a subtask in information retrieval, where we look for entities and objects that can be denoted by a proper name, such as persons, organizations or locations. The results are often used in a pipeline, such as question answering or entity linking.

The aim of this work is to research and implement a named entity recognition system on the Czech Named Entity Corpus (CNEC 2.0) and CoNLL2003 dataset. We will do several experiments using standard algorithms for sequence labelling (Conditional Random Fields) with linguistic features extracted from literature and several neural network architectures which operate on raw data.

Abstrakt

Rozpoznávání pojmenovaných entit je podúloha ve vyhledávání strukturovaných informací. Jedná se o klasifikaci slov ve větě, které reprezentují entity s vlastním jménem, jako na příklad osoby, organizace nebo lokace. Rozpoznávání entit se často využívá v automatickém odpovídání na otázky.

Cílem této práce je rešerše a implementace systému pro rozpoznávání pojmenovaných entit na českém (Czech Named Entity Corpus 2.0) a následně anglickém (ConLL2003) datasetu. Bude provedeno několik experimentů s běžnými algoritmy na klasifikaci sekvencí (Conditional Random Fields), které vyžadují ruční tvorbu příznaků, a umělými neuronovými sítěmi, které se reprezentaci dat učí samy.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Named entity recognition (NER) is commonly used in information retrieval tasks such as question answering, where we want to detect entities that contribute to the question. The detected entity is afterwards linked to a concept in a knowledge base (a task commonly referred to as entity linking). Alternatively, the class of the entity itself is used as a feature for question classification. With the increase of chatbots, personal assistants and AI conversation agents, the need for good natural language processing (NLP) and natural language understanding (NLU) systems increases. Unlike part-of-speech (POS) tagging, NER is a more difficult task due to the ambiguity of words. For instance, apple might mean the computer company or the fruit, depending on the sentence. *New York* is a named entity while a *new car* is not. Most systems solve this by considering the context of the word as well.

In this work, we will first review the datasets and current NER methods. Afterwards, we will propose several neural network architectures and discuss their advantages and disadvantages. Finally, in the practical part, we will perform several experiments and compare the results.

Systems with expertly picked linguistic features tend to have good performance. The approach of neural networks is different, as it tries to learn a good representation from raw data instead. This had great success in computer vision with convolutional neural networks, due to the way pictures are structured. For natural language processing (NLP), different approaches were required. One of those is the recurrent neural network (RNN), which keeps an inner state, allowing it to keep long term memory and remember relevant parts of the sentence.

Chapter 2

Related work

Many machine learning approaches have been implemented for NER. Both generative and discriminative models have been applied. Some ignored the sequential structure of the task, some tried to use it for their benefit.

For CNEC, the first algorithm was a decision tree [36] in 2007 which used boolean, contextual and categorical features. Contextual features were represented by the presence or absence of certain trigger words that were semimanually extracted from the training data. The approach was to detect one-word named entities and two-word named entities separately. For longer entities, special hand-crafted algorithms were used. They covered only street names as a special case.

There is also a recognizer based on Support Vector Machines (SVM) published in 2009, also known as maximum margin classifier. They also created three separate classifiers for one-word, two-word and three-word named entities, omitting any longer ones. For SVMs, the bag of words approach was omitted to keep the dimensionality low. They used part-of-speech tags, boolean features, gazetteers and the context around the given token. The resulting feature space dimension was 200.

Another approach was using a maximum entropy classifier by [22]. Maximum entropy is equivalent to logistic regression [27]. For features, they used a very similar set of feature functions like for our CRF system, with bag of words and categorical features. In 2013, a two-stage maximum entropy classifier with Viterbi decoding was released [38]. The novel approach was using dynamic, two-stage decoding of the probabilities instead of the argmax in the sequence. The current state-of-art was published near the end of 2016 and uses a RNN with added character level features [39].

For ConLL2003, the algorithms include adaptive boosting (AdaBoost) [6] which uses a linear combination of weak classifiers, in this case fixed depth decision trees. Other approaches include a character based Hidden Markov Model [21], a LSTM neural network [12] and ensemble approaches that combined several classifiers together [10] [47].

Chapter 3

Problem specification

In this chapter, we will describe the task along with its evaluation metrics. Afterwards, we will describe in detail both datasets.

3.1 Description of the task

Named entity recognition is at its core a sequence labelling task, where we take a sentence and annotate each word using tags. It contains several subtasks, such as tokenizing or chunking, but we will disregard those for the purpose of the experiment and consider the text to be already split into tokens. General NER systems extract several linguistic features, such as word case, suffixes, prefixes, parts-of-speech tags or dependency parse trees. Afterwards, they employ a statistical model to predict a tag for the word.

3.2 Evaluation metrics

There are many ways to measure the performance of a NER system. The easiest one is to simply count the number of correct tags and divide it by the total number of tags. We call this accuracy. There are also more sophisticated methods to deal with multi-word entities. Those only count it as correct when the whole entity has been matched. For that purpose, the classes are extended with B(egin), I(nside) and O(outside) tags. We call this the BIO scheme. Some systems go even further and add L(ast) and U(nit), creating the BILOU scheme. The article [38] uses a modified BILOU scheme. Finally, we introduce the terms precision, recall and F1 score.

Precision Precision is the number of correctly predicted entities divided by the number of all predicted entities.

Recall Recall is the number of correctly predicted entities divided by the number of entities located in the data.

F1 F1 is the harmonic mean of precision and recall.

$$F1 = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{3.1}$$

3.3 Datasets

3.3.1 CoNLL-2003

The CoNLL-2003 dataset concerns language-independent named entity recognition. The dataset contains English and German newspaper articles annotated by people from the University of Antwerp. The English dataset is created from the Reuters Corpus. The annotations were publicly available, but the raw data was not. The articles had to be requested from the National Institute of Orders and Technology. The CoNLL-2003 dataset considers 4 types of entities: Person, Organization, Location and Misc. This, together with the BIO scheme, leads to 9 different classes: P-B, P-I, O-B, O-I, L-B, L-I, M-B, M-I, O. In this case, P-B would signify Person-Begin, O-B Organization-Begin, etc. The lone O is the Outside tag. The data includes POS tags.

3.3.2 CNEC 2.0

The Czech Named Entity Corpus is a dataset annotated by the Institute of Formal and Applied Linguistics at the Charles University in Prague. It consists of 8993 sentences and has 46 different tags in a two-level scheme. The upper level contains 8 supertypes and the lower has 46 different types. This allows us to use different granularities for evaluation. 46 types with the BIO scheme leads to 93 different classes spread between 35,220 entities, which makes the task notably harder. Some entities are mentioned very rarely and others are only in multiword named entities. Named entities in CNEC can be nested in each other. For example, `<gu Ustí nad <gh Labem>>` contains two entities.

There are also special tags defined in [36] called containers, which encapsulate multiword named entities. They are written in uppercase letters. For instance `<P <pf Gordon <ps Summer>>` would signify a Person with a given name and surname.

Finally, there are special tags that do not signify entities but rather attributes such as <f> for foreign word, <s> for abbreviation, <cap> for capitalized words.

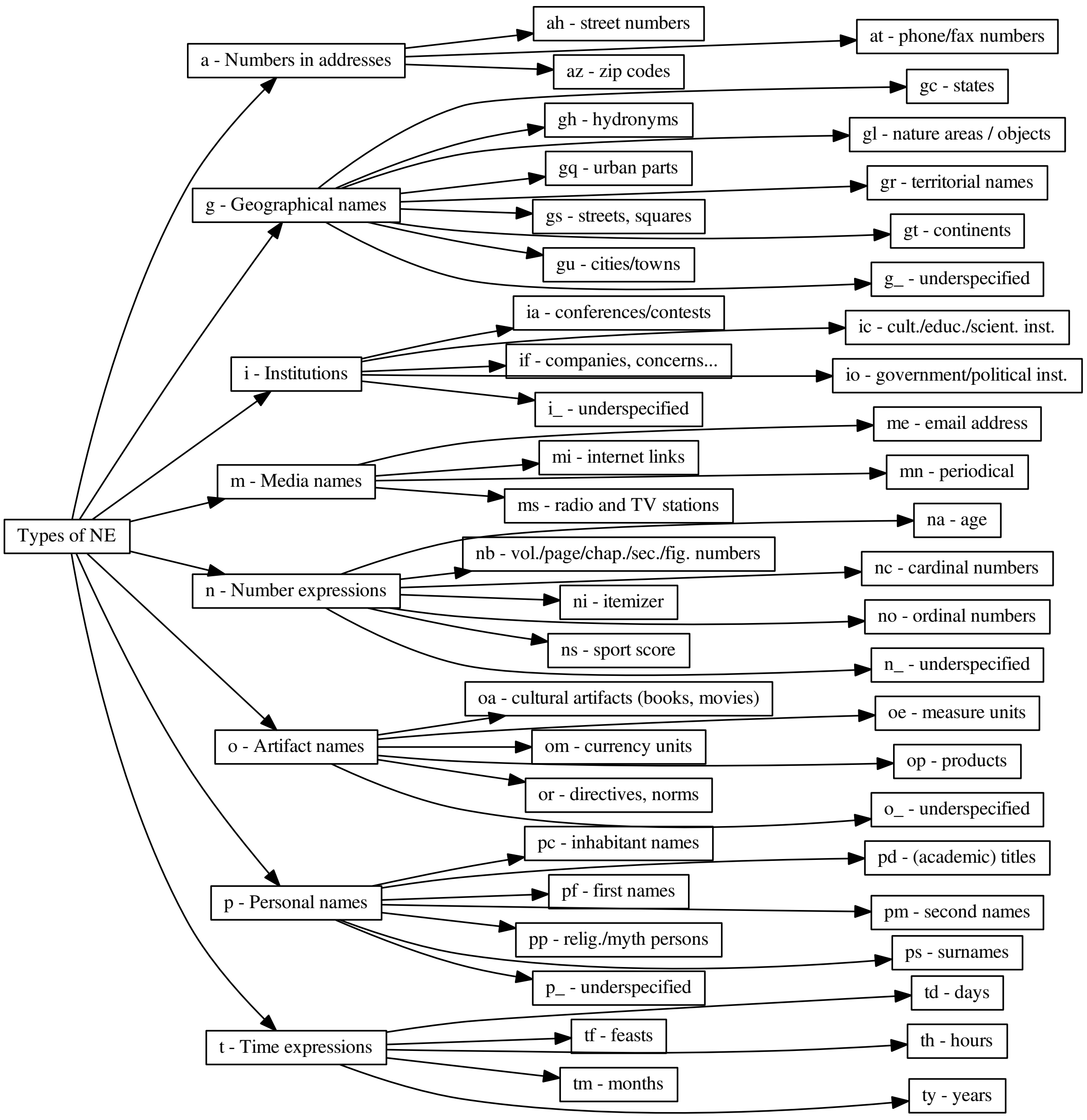

Figure 3.1: A description of the NE tags in the CNEC2.0 dataset taken from [35]. Note the third column is for spacing reasons.

Table 3.1: This table contains the container tags in the CNEC2.0 dataset

Container	Meaning	Example
A	address	<A< KOMO>, < gs Knížecí> <ah 12/173>, <az 709 00> <gu Ostrava-Nová Ves>, tel.: <at 069 6621773>, <at 6621375>, <at 601527588>, fax: <at 069 6621773>>
C	bibliographical information	<C<P<pf G.> <ps Lukács>>: <oa<f< Die Theorie des Romans>>, <gu Berlin>, <ic<f Verlag <P< pf Paul> <ps Cassier>>>> <ty 1920>>
P	person name	<P< pd Doc.> <pd MUDr.> <pf Přemysl> <ps Doberský>, <pd DrSc.>>
T	time value	<T< td 21.> <tm června> <ty 2003> <th 20.00>>

Table 3.2: A table containing non-entity special tags with the explanation and an example

Tag	Meaning	Example
s	An abbreviation	<io<s ODS>>
f	A foreign language word	<if<f Deutsche Bank>>
segm	A word which was capitalised due to a segmentation error	Revoluční zvrat v pohledu na život <segm Základem> života není....
cap	A word written in all caps	A jak <cap TO> vysvětlíte?
lower	A word which was wrongly capitalized	ekonomicky ovládnout <lower Střední> <gt Evropu>
upper	A word which was wrongly written in lower case	dalšim zajímavým <upper britem> je...
?	A unspecified entity which doesn't belong to any of the previously mentioned categories	<? Asmara> se odmítá stáhnout z území...
!	The whole sentence is unannotated	<!> 70: 15 Písní na S. Georga, op.

Chapter 4

Algorithms

4.1 Introduction

In this section, we will mention several commonly used algorithms for sequence labelling. All of them model the conditional probability of a label given the context $P(y_n|x_n, x_{n-1}, ..., x_1)$. The first ones, Hidden Markov Models (HMM), are generative models that assign a joint probability to paired observations and label sequences [23]. A generative model must usually enumerate all possible observation sequences, which makes inference intractable. HMMs apply the Markov Assumption, which states that the probability distribution of future states only depends on the present one and not on the previous ones. This makes training and inference easier, but does not allow us to model long-range relations between words. HMMs were used to align biological sequences or in POS tagging. They were followed by Conditional Random Fields [23] (CRF). CRFs are undirected discriminative graphical models. They model the conditional probability instead of the joint probability. For sequence labelling, a subset called linear chain CRFs is used. They use a directed linear structure, leading to both simpler training using maximum likelihood and easier inference using dynamic programming. CRFs allow us to easily employ a variety of both global and local features. This can lead to computational issues (considering both memory consumption and compute time), so we dedicated section 4.2.6 for feature selection and pruning. Finally, we will describe both general neural network architectures and ones containing internal memory for structured classification of sequences (i.e. recurrent neural networks). We will briefly discuss training and common issues.

4.2 Linear-chain Conditional Random Fields

4.2.1 Definition

Definition 4.2.1. [43] Let Y, X be random vectors, $\mathrm{w} \in R^k$ be a weight vector and $\mathcal{F} = f_k(y, y', \boldsymbol{x})_{k=1}^K$ be a set of real-valued feature functions. Then a linear-chain conditional random field is a distribution $p(\boldsymbol{y}|\boldsymbol{x})$ that takes the form:

$$p(\boldsymbol{y}|\boldsymbol{x}) = \frac{1}{Z(\boldsymbol{x})} \prod_{t=1}^{T} \exp(\sum_{k=1}^{K} w_k f_k(y_t, y_{t-1}, \boldsymbol{x}_t)) \tag{4.1}$$

where T is the number of timesteps and K is the number of feature functions. $Z(\boldsymbol{x})$ is an input-dependent normalization function to turn it into a probability,

$$Z(\boldsymbol{x}) = \sum_{y} \prod_{t=1}^{T} \exp(\sum_{k=1}^{K} w_k f_k(y_t, y_{t-1}, \boldsymbol{x}_t)) \tag{4.2}$$

The feature functions can take both numeric and boolean forms, but as we only train the weights, they are most commonly set to 1 if they are true and 0 otherwise. There is a large variety of possible feature functions which we will discuss in 4.2.5.

In the case of sequence labelling, X would be the input and Y would be the output NE tags.

4.2.2 Discriminative versus generative learning

CRFs can be seen from two separate viewpoints. The first one is as an extension of logistic regression (a discriminative model) to sequences. The second is as a relaxation of Hidden Markov Models (generative sequence models) to conditional probability. There is thus a 4 way relation between Naive Bayes and Logistic Regression, HHMs and CRFs [43].

4.2.3 Training

We are given independent and identically distributed data $D = \boldsymbol{x}^{(i)}, \boldsymbol{y}^{(i)}{}_{i=1}^{N}$ where $\boldsymbol{x}^i = (x_1^i, x_2^i, x_T^i)$ is a sequence of inputs and $\boldsymbol{y}^i = (y_1^i, y_2^i, y_T^i)$ is a sequence of predictions. The weights are trained using maximum likelihood estimation.

$$L(w) = \sum_{i=1}^{N} \log p(\boldsymbol{y}^{(i)} | \boldsymbol{x}^{(i)}) \tag{4.3}$$

We substitute this into the linear chain CRF:

$$L(\boldsymbol{w}) = \sum_{i=1}^{N} \sum_{t=1}^{T} \prod_{k=1}^{K} w_k f_k(y_t, y_{t-1}, \boldsymbol{x}_t)) - \sum_{i=1}^{N} \log Z(\boldsymbol{x}^{(\boldsymbol{i})}) \tag{4.4}$$

The weights we seek are thus:

$$\boldsymbol{w}^* = argmax(L(\boldsymbol{w})) \tag{4.5}$$

This is an optimization task. The problem is concave [43], thus any local optimum is also the global optimum. In machine learning tasks, it is common to introduce regularization. Regularization is a function that penalizes complex models to avoid overfitting on the training data. There is for example L2 regularization, where we add the squared Euclidean norm of the weights to the minimization term. This encourages the weights to be small. Similarly, an L1 norm takes the absolute value of the weights. This induces sparsity, i.e. many weights are set to 0. We used L2 regularization in our CRF module, because we performed feature selection separately. The optimization task is thus:

$$L(w) = \sum_{i=1}^{N} \sum_{t=1}^{T} \sum_{k=1}^{K} w_k f_k(y_t, y_{t-1}, \boldsymbol{x}_t)) - \sum_{i=1}^{N} \log Z(\boldsymbol{x}^{(i)}) - \lambda \cdot \sum_{k=1}^{K} w_k^2 \tag{4.6}$$

where λ is a hyperparameter to determine how much we should penalize large weights. To optimize this, iterative methods are used. The simplest one being gradient descent, which is rather slow [43]. Alternatively, one can use second order methods such as Newton's method for faster iteration, but the resulting Hessian is usually too large to fit into memory, making the approach infeasible. As a compromise, Quasi-Newton methods are used. One can approximate the Hessian using the first derivative (BFGS), or use its limited memory version. (L-BFGS).

4.2.4 Inference

For inference, the linear-chain CRF uses algorithms which are analogous to the Viterbi algorithm for Hidden Markov Models. The Viterbi algorithm [32] is a a dynamic programming algorithm to pick the most likely *sequence* of hidden states in a graphical model with a linear structure. It stores all path probabilities in a 2D matrix instead of taking the most probable tag with every token. At each timestep, it saves the currently most probable path to each of the output tags and calculates the next step using those values. When the algorithm gets to the last token, it chooses the most probable path using backpointers.

4.2.5 Feature functions

There are several distinct categories of feature functions considered. We were inspired by several research papers [44, 35, 10]

- **Word tokens** A feature function outputs 1 iff $\boldsymbol{x_i}$ is equal to a certain word. For the implementation, we made templates that generate the feature functions out of the given text automatically. We encoded it as `w[0]=word`. This is was used as a baseline approach.

- **Neighbouring tokens** We encoded neighbouring words the same way. The feature function outputs 1 iff the previous or next token is a certain word. The encodings are `w[1]=word`, `w[2]=word`, `w[-1]=word`, `w[-2]=word`. For the edge cases, we used special tokens such as `w[-1]=START` and `w[1]=END`

- **Conditional tokens** Additionally, we encoded the previous token with the current token like `previous|current`. The reasons for this is to circumvent the linear nature of linear chain CRFs. We can't use the output of the previous step, so we add a separate feature activation instead.

- **Word lemma** A lemma is also known as the canonical or dictionary form of the word. Since Czech is a morphologically rich language, we employed a lemmatizer to cut down the number of words. However, morphologically relevant information might be lost from the word so we encoded the suffixes and prefixed separately.

- **Word suffix** We considered word suffixes of several lengths (3,4). If the word was shorter than the threshold, we would use the whole word instead. The reasoning was that certain suffixes signify that the word is an adjective and could be part of a multi-word named entity.

- **Word prefix** We maintain similar arguments for prefixes.
- **Boolean features** We also included basic boolean features signalling whether the word is capitalized, contains a number, or an "@" sign.
- **Word shape** The word shape was extracted from [44]. We would substitute consecutive character sequences of the token with a mask: *A* for uppercase letters, *a* for lowercase letters, *N* for numbers and *"."* (a dot sign) for punctuation. Thus, *Jakub* would turn into *Aa*. *7.května* would become *N.a*, *12.4.1996* would turn into *N.N.N.*
- **Part of Speech tags** A key feature for NER systems are part-of-speech (POS) tags. POS tagging is a process where we assign a part of speech to a token based on it's definition and context. POS tags include noun, verb, adjective, adverb, etc. They allow us to generalize and infer entities from the sentence structure. We didn't use the commonly used Stanford CoreNLP pipeline but opted for the newly released, neural network based SyntaxNet by Google instead. We used the local, neighbouring and conditional tags.
- **Brown Clustering** Brown clustering [5] is a hiearchichal clustering of words based on their appearance in similar context. It can be seen as a series of merges which minimize the mutual information gain using a greedy heuristic. The result of Brown clustering is a binary tree of words, where each word has an assigned bit-string. By cutting (taking the substring) the clustering at different positions, we gain several numbers of clusters. Thus 10101010111 can be cut into 10101010 or further to 101010, each would represent a cluster of a different granularity. The greedy approach leads to the most common words being used in each cluster instead of semantically similar words. We used the whole string, and substrings of a range of lengths. The main motivation for Brown clustering was to deal with out of vocabulary words. Different named entities/names would be shown in similar contexts and would thus be part of the same clusters.
- **Gazetteers** Another common NER feature is the gazetteer. A gazetteer was originally a geographical dictionary which contained additional information such as statistics or physical features of the region. In named entity recognition, gazetteers are simply lists of given names, countries and organizations. We extracted given names, surnames and addresses from the Czech Ministry of Interior website. The feature is simply a boolean signalizing whether the token is in the given gazetteer.

4.2.6 Feature Selection

The resulting number of features is in the range of one hundred thousand features. While the model itself is roughly 40MB in size, we still wish to prune unneeded features. We already mentioned L1 regularization which would set many weights to 0 and would enable us to discard them. Another method used before actual training is to count the number of instances. Afterwards, we remove feature functions that occurred less than 10 times in the training data. For example, we would remove `w[0]=rareWord`.

4.3 Neural Networks

Neural networks and by extension deep neural networks are a powerful class of machine learning algorithms which use a nonlinearity projection to stack linear layers on top of each other. They are inspired by the biological brain and try to approximate its function. For example, it has been shown that convolutional neural networks correlate to the animal visual cortex [16].

4.3.1 Definition

The basic computing unit of a neural network is a neuron, which takes several inputs and applies a weighted sum. A single neuron is equivalent to a perceptron.

$$p^{(i)} = \sum_{i=1}^{|x_i|} x_i w_i \tag{4.7}$$

Afterwards we apply a nonlinear function σ (we will discuss those in section 4.3.2). We call the result an activation. Neurons are grouped in layers in an acyclic graph [11]. Note a single layer can be represented using a weight matrix W and the calculation $W^T \cdot x$ can be done in parallel. The acyclic graph assumption leads to a feed forward network [28] (in certain literature known as multilayer perceptrons), where the information travels only one way. There are modifications that allow the outgoing information to be fed back to the neuron which are called recurrent neural networks, which we will discuss in section 4.3.7. We can write the neural network as a series of nested functions, which will later be useful during training:

$$y^{(i)} = \sigma(w_i^T \cdot \sigma(w_{i-1}^T \cdot \sigma(w_{i-2}^T \cdot x))) \tag{4.8}$$

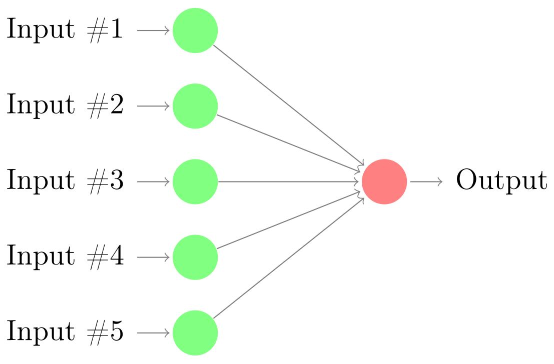

Figure 4.1: A single neuron

4.3.2 Activation functions

One of the first activation functions was the step function, which is simply:

$$f(x) = \begin{cases} 1, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.9}$$

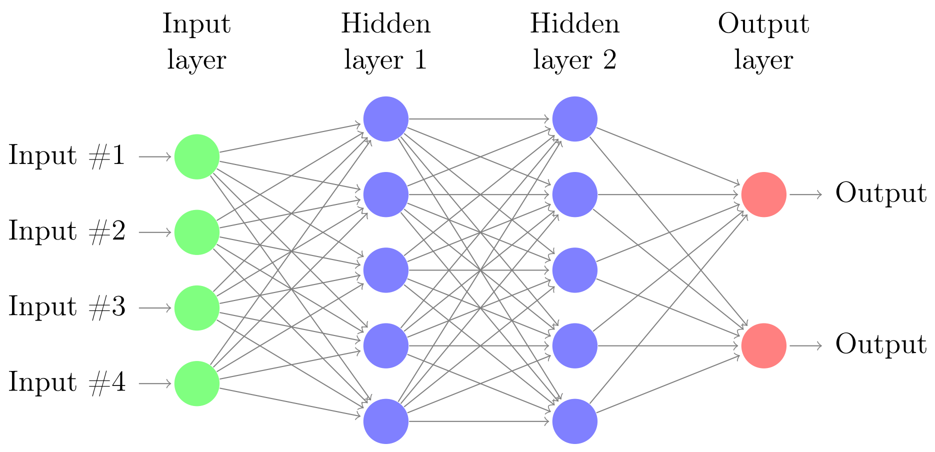

Figure 4.2: A feedforward neural network with 2 hidden layers and 2 outputs.

The step function was used in early perceptron models, but was not continuous and thus was not suited for gradient-based backpropagation. The most common activation function was the logistic sigmoid, which allows an area of uncertainty.

$$f(x) = \frac{1}{1+e^{x}} \tag{4.10}$$

Additionally, its derivative has the following closed form:

$$f'(x) = f(x)(1 - f(x)) \tag{4.11}$$

There is ongoing research on activation functions and how they influence training. There are issues in training such as the vanishing gradient problem [13], where the error gradient turns too close to zero and does not influence the deeper layers. For example with the logistic sigmoid function, very high values would lead to the gradient being 0. This was partly addressed by introducing the Rectified Linear Unit (ReLU) with the following form:

$$f(x) = \begin{cases} x, & \text{if } x \geq 0 \\ 0, & \text{otherwise} \end{cases} \tag{4.12}$$

For large x, the derivative is simply 1 and the error signal doesn't vanish. It was also mentioned in literature that ReLU often introduces sparsity [48] and improves generalization. There are many modifications of ReLU, such as Leaky Rectified Linear Unit (LReLU) and Exponential Linear Units (ELU) [9]. Finally, we will mention the activation function used at the output layer, the softmax:

$$f(x)_j = \frac{e^{x_j}}{\sum\limits_{k=1}^{K} e^{x_k}} \text{ for } j = 1..K \tag{4.13}$$

Softmax turns the output of K functions into a probability distribution and is used for multiclass classification [11].

4.3.3 Training

The goal during training is to minimize a loss function. Loss functions include the mean squared error (MSE), Kullback–Leibler divergence and categorical cross entropy [11]. As we are doing multiclass sequential classification, we will concentrate on categorical cross entropy. The reason it is used more than MSE is that MSE penalizes wrong results heavily due to the way the error is calculated. Categorical cross entropy is defined as:

$$L(f(x_i), y_i) = -\sum_{j}^{C} y_i \log f(x_i) \tag{4.14}$$

There are many methods how to update the parameters of a neural network, such as evolutionary algorithms or expectation maximization, but we will mainly concentrate on gradient descent using backpropagation.

Gradient descent methods Gradient descent (ascent) is an iterative method to find a local minimum (maximum) of a function. It computes the gradient of the function to update the weighs.

$$w_{i+1} = w_i - \lambda_i \nabla f_{(w_i)}, i = 0, 1, ... \tag{4.15}$$

Similarly to training conditional random fields, the main advantage of the gradient descent method are the memory requirements. We only need the first derivative, which is $O(n)$ parameters. For very large datasets, an approximation of gradient descent named stochastic gradient descent is employed, where we use either a randomly picked single sample or a small batch instead of the whole dataset. The drawback is generally a slow convergence speed. We can tune λ as a learning rate parameter and many minimizers use it to improve convergence.

We will mention second order algorithms for completion's sake and discuss why they are not usable in neural networks. A commonly mentioned optimization method is Newton's method, which is derived from the quadratic approximation of the target function using Taylor's series expansion.

$$w_{i+1} = w_i - \lambda_i H^{-1} \nabla f_{(w_i)}, i = 0, 1, ... \tag{4.16}$$

where H is the Hessian or the matrix of all second derivatives. and λ is the step size or learning rate again. The advantage is faster convergence. However, the drawbacks are more pronounced in neural networks due to the large number of parameters. Firstly, calculating the Hessian and its inverse for each layer is computationally very expensive, making training very slow despite faster convergence. Secondly, the Hessian has $O(n^2)$ parameters to store which is unfeasible for large networks. Finally, the Hessian makes it impossible to distribute the neural network among different GPUs or machines for large scale learning. Current trends tend towards calculations which can be done locally and thus in parallel.

4.3.3.1 Backpropagation

Backpropagation is the use of the chain rule of derivatives on the composed functions to update the weights at each layer [33]. We first do a *forward pass* by inputting the data in the network and propagating it to the last layer. Afterwards we calculate the error by comparing the output with the true labels (in our case we use categorical crossentropy). Afterwards, we update the weights going starting with the last layer and going backwards towards the input. The change in a given hidden layer depends on its own derivative and its output weighted by the succeeding layer's derivative.

4.3.3.2 Optimizers

Common improvements to gradient descent methods include tuning the learning rate. The easiest one is exponential decay, where we gradually lower the learning rate between iterations. Further algorithms such as Adaptive subgradient methods (Adagrad) try to remove the learning rate hyperparameter completely. Alternatively, one can add another term which would be easier to compute than the Hessian. We will mention both in the following list:

- Stochastic gradient descent with momentum adds a portion of the previous output to the gradient. intuitively, the momentum helps the gradient to accelerate in the correct direction.
$$w_{i+1} = w_i - (\gamma \cdot w_{i-1} + \lambda_i \nabla f_{(w_i)}), i = 0, 1, ... \tag{4.17}$$
- Adaptive moment estimation (Adam) [20] is a method, which computes different learning rates for each parameter.

4.3.4 Challenges

4.3.4.1 Hyperparameters

Neural networks generally contain a large number of hyperparameters and decisions. From network architecture (Number of dense layers, number of nodes in each layer, dimensions of the features or the choice of activation functions) to learning rate and weight initialization, each decision has an impact on the resulting generalization and training speed. Common machine learning approaches for hyperparameter search, such as cross-validation, tend to be infeasible for very large neural networks where training can take weeks.

One way to make a neural network more robust against fluctuations from initial parameters is called batch normalization [18]. Batch normalization normalizes the input of a layer to a normal distribution. This addresses the problem that the distributions of layer inputs often change along with the previous layers parameters during training, making learning more difficult. Normalizing the layer input makes it less dependent on previous layer changes and allows us to use higher learning rates. We found this to be empirically true in chapter 6 as it lead to a great improvement in performance.

Other hyperparameters, such as the learning rate, can be automatically estimated by the optimizers. For batch size, a common decision is to choose the largest one that fits into the memory after setting the architecture.

Most architecture decisions come from experience, but current research concentrates on automatically generating neural network architectures for the given task using reinforcement learning [2].

4.3.4.2 Vanishing and Exploding Gradients

We have already mentioned the vanishing gradient problem in 4.3.2. An opposite problem is the exploding gradient, where the value of the weight update approaches infinity, making further learning impossible. This is commonly detected very early on as the value of the loss function quickly approaches infinity as well. A simple solution to this so called gradient clipping [30], where we normalize the gradient if its L2 norm reaches the threshold.

4.3.4.3 Local minima

Neural Network optimization is not a convex problem. A notable issue is getting stuck in a local minimum. The naive approach is to restart the network several times. Most advanced optimizers tend to increase the learning rate hoping that the we jump out of the gradient valley. There is however no general guarantee of global convergence, making training rather tricky and an ongoing problem with neural network training.

4.3.4.4 Overfitting

Neural Networks naturally tend to overfit due to their expressive power. It is easier to remember a certain feature to be at a specific point than to learn the structure of the problem. Overfitting is usually detected by having a validation set and looking at the classification error and loss there. One way to prevent overfitting is early stopping. We stop training the network when the validation error does not improve. A commonly cited form of regularization is dropout, where we randomly turn off a portion of neurons during training. This forces the network to be more robust and not to concentrate on a single neuron output, which prevents overfitting. It has been said [17] that dropout can be interpreted as averaging several models trained in an ensemble. Batch normalization is also cited as a form of regularization [18].

4.3.5 Modelling power

The universal approximation theorem states [15] than neural network with a single layer can approximate continuous real valued functions with arbitrary accuracy, given enough hidden nodes. The theorem does not state how many nodes are required, only that it is a finite number. One can imagine this as given enough hidden nodes, the neural network can learn a simple mapping from any combination of inputs to a correct output. This is undesirable as it is an extreme form of overfitting where the model does not generalize at all.

The true power of neural networks is *representation learning*. This is easy to see with convolutional neural networks for computer vision, where each filter learns to detect a simple shape such as a line or a circle. These filters are then the fed to the next layer which detects more complex shapes until finally a fully connected layer does the classification. This is considerably more difficult to visualize for text based tasks, but many approaches were proposed and implemented. We tried two of these, word embeddings which embed words into a continuous space and character embeddings, which go one level lower and represent characters. We then tried to encode words out of them using a smaller recurrent neural network or use their n-grams as features. We will discuss both approaches in 4.3.6. To model long-term relations between words, we will introduce recurrent neural networks in 4.3.7 We will also discuss the benefits of augmenting raw data with additional, external features.

4.3.6 Features

With conditional random fields, we created a range of indicator functions for each word and feature and did sequence modelling using those. The easiest approach for a neural network is one-hot encoding. Suppose we have a vocabulary of size 20000. We would encode each word using its own vector which would have a one at a certain index and contain zeroes otherwise. After multiplying this with a weight matrix, the one would select the correct row and obtain the correct n-dimensional word embedding. Alternatively, we can directly feed the word embeddings to the network.

4.3.6.1 Word2Vec

We mentioned embedding words into a continuous space but we did not elaborate how to obtain an embedding where certain semantic relations between words are preserved. A successful approach [26] is Word2Vec. Word2Vec is a shallow two layer neural network which takes a large corpus of text and either tries to predict the next word from the surrounding words (we call this continuous Bag-of-Words), or tries to predict the surrounding words from the current one (we call this continuous skipgram). By training this neural network, we end up with a vector representation of words in the weight matrix. Words that are contained in similar contexts are located in closer proximity in the vector space. This correlates to the Distributional hypothesis [34] which states that *linguistic items with similar distributions have similar meanings.* Certain word embeddings can be even used in mathematical formulas. A commonly quoted example is:

$$king - man + woman = queen \tag{4.18}$$

In our work, we tried untrained word embeddings (i.e. a simple one hot encoding, leaving the network to learn the necessary relations) and pretrained continuous skipgram Word2Vec model given by the Seznam.cz company for research purposes. We also tried to append several additional features using one hot encoding. The features included:

- POS tags: While word embeddings retain semantic and potentially syntactic information, we wanted to emphasize the role of the word in the *current* sentence instead of a generalized average.
- Gazetteers: We added gazetteers because we assumed the neural network can't possibly learn that.
- Boolean features: We also tried to add several boolean features (isCapitalised, contains@, isNumber) to see if the additional information could improve performance.

Finally, we added a 1D convolutional layer. The idea was to make small (4-5) word contexts and force the network to look at the short term dependencies (disregarding locality) instead of just the long term dependency. In simpler terms, we don't care where the trigram "New York City" is in the sentence, as long as the 3 words are together.

4.3.7 Recurrent Neural Networks

So far, we have mainly described feedforward neural networks, which take the whole input at once. We will now introduce a modification which adds an inner state to the neuron and allows it to be modified. After outputting the activation, the neuron also has a loop back to itself. Due to this structure we call these Recurrent Neural Networks (RNN). This inner memory allows us to model long-term dependencies (the output is also dependent on the entire preceding history) in a sequence and such networks are thus suited for time series modelling and sequence classification. We will first describe the RNN conceptually and afterwards mention one concrete implementation. Note the weight matrix is the same

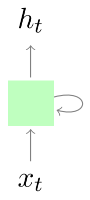

Figure 4.3: A recurrent cell at time point t takes the input x_t and outputs h_t

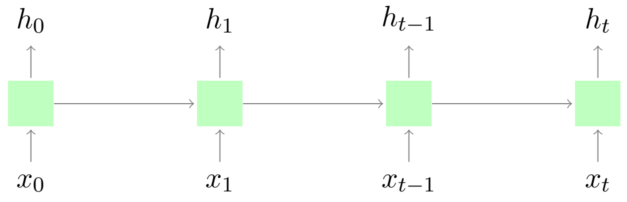

Figure 4.4: Unrolling the recurrent cell makes it easier to see the transitions between states h_0 and h_t

between the timesteps. By looking at the unrolled network, we can see it as a very deep (up to the length of the sequence) neural network with fixed weights. RNNs in their base form, while being theoretically able to, tend to have great difficulties to learn long term dependencies due to the vanishing gradient problem which is accentuated by the deep structure of the network [3] [13]. Simply put, the error gradient between timesteps t_i and t_j tends to grow too small for $j \gg i$ and usually only the short-term relations are learned. This was addressed by introducing Long Short Term Memory (LSTM) networks introduced in [14].

4.3.7.1 LSTM

While a basic RNN might contain a simple sigmoid or tanh neural network in the cell, an LSTM contains 4 neural networks in a special structure. We call them gates. The information can freely pass through the network and is modified by the gates if needed. Thus the cells further down the pipeline can still learn the relations.

- The **forget gate** decides whether to keep the incoming information or not.

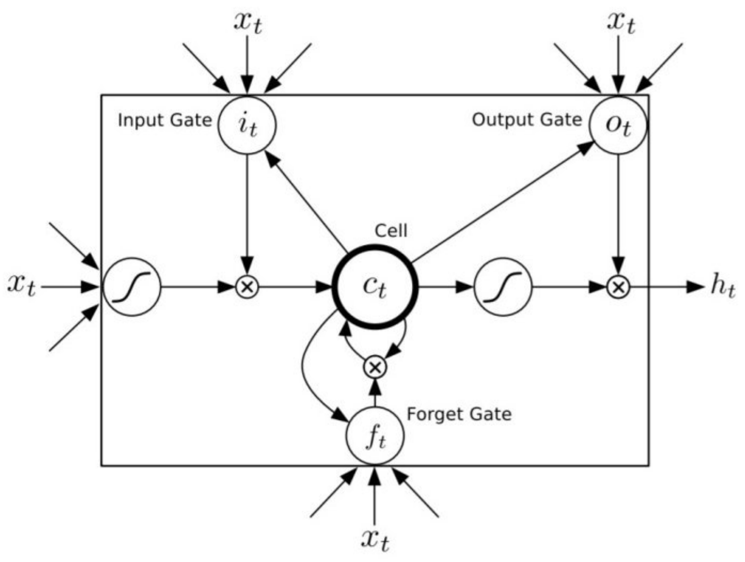

Figure 4.5: A scheme of a LSTM cell taken from [14]

- The **input gate** decides what values in the cell we want to update.
- The **cell** is the network that generates the potential values.
- The **output gate** decides which part of the cell state to output.

LSTM networks have been very successful in a variety of tasks such as speech recognition [14], machine translation [42] [1], image captioning [45], question answering [46] or language modelling [41]. We will also mention Gated Recurrent Units (GRU), a simplified form of LSTMs that contains two gates instead of three [7], leading to a smaller number of parameters, which relates to faster learning rate and better performance with smaller amounts of data. We will test that empirically in chapter 6.

Chapter 5

Implementation

In this section, we will describe our system architecture and mention the libraries we used. Finally, we will discuss our chosen neural network architectures. We chose Python3 as the programming language for its suite of text libraries and neural network bindings, which allowed us to reuse code between different classifiers. We have a set of tools to read both the conll2003 and CNEC2.0 datasets and set the granularity of NE tags. Most work was spent on the feature extractor, which is an extensible framework that generates features usable by both the CRF implementation and the neural networks. Finally we added a simple web service written in the Python Flask framework.

5.1 External Annotators

We used several additional annotators. For word lemmas, we used Morphological Dictionary and Tagger (MorphoDiTa) [40], an open source tool for morphological lemmas made by the Institute of Formal and Applied Linguistics at Charles University. For the tagger, a language model is needed[1]. We also used a word stemmer[2]. Finally, we used a SyntaxNet[3] implementation for part-of-speech tagging. We queried the REST API using the *requests* Python library and saved the POS tags in a json format afterwards for further use. For Brown clustering, we chose a C++ implementation by Percy Liang [24] and trained it on the Czech Wikipedia.

5.2 Feature Extractor

We instantiate our feature extractor with a list of strings representing function names. The feature extractor contains a dictionary of feature functions internally and transforms the text by sequentially applying each feature function and appending the result to a list. A typical feature function starts with `ft_`. For example: `ft_POS_current(*params)` returns the POS tag for the current word. The input parameters are the current token, its position in the

[1]We used czech-morfflex-pdt-161115 [37].

[2]`http://research.variancia.com/czech_stemmer/`

[3]`https://research.googleblog.com/2016/05/announcing-syntaxnet-worlds-most.html`

sentence starting with 0 and a reference to the whole sentence. This allows each feature function to access neighbouring words and output features based on that. Internal functions (REST api query, dictionary loading) are prepended with an underscore. To add a feature function, one simply implements it in `feature_extractor.py`, adds it to the dictionary of functions and adds the feature label to the model configuration file.

5.3 CRF

We used the C++ implementation CRFSuite [29] with Python bindings for the linear chain conditional random fields. The features are fed as text strings for both training and tagging and the CRFSuite generates its own internal implementation. As mentioned in 4.2.6, we employed L2 regularization of the weights and additionally pruned feature functions that did not occur more than 10 times in the training data. The latter was done to reduce the overall size of the model.

5.4 Neural Networks

For the neural networks, we picked the Keras [8] a neural network framework that has recently been promoted to a Tensorflow frontend. We had no need to write special graph operations, so we deemed the high level interface to be enough. While RNNs can theoretically take inputs of any length, they cannot process batches of sentences with variable length[4]. This is purely an implementation issue. Due to the way the recurrent cells are initialized in memory, the recurrent layers are usually unrolled to a certain fixed length for batch learning. Thus the three possible approaches are:

1. Set the batch size to one, training one sentence at a time. The network is then reconfigured for the next one.
2. Group sentences with the same lengths together and train batches of those sentences.
3. Pad (or cut) all sentences to the same size and train batches of those. The padded values can either be masked or are more often simply set to 0.

Setting the batch size to one makes training extremely slow, so 2 and 3 are more common. The second approach is more memory efficient as one long sequence can force all the data to be padded to its size, leading to large matrices filled with zeroes. In the Tensorflow, developers typically unroll the RNN cells to several sizes, one for each length bucket (hence the name bucketing). This leads to several neural networks with shared parameters that get updated after every mini batch. This allows the model to output more accurate variable length sentences. For our task, we used padding for two reasons. Firstly, we wanted to experiment with any batch size, which would be disallowed by approach 2. Secondly, the sentences were close enough to each other (with one outlier) that padding would give a reasonable length. For our task, we padded the sentence to 60 words.

[4]Tensorflow currently allows dynamic unrolling, but the feature was not available during the implementation of this work.

For the untrained word embeddings, we ran through the training dataset to create a mapping of words to an index. We also had separate embedding for out-of-vocabulary words, numbers, punctuation and padding. For the pretrained embeddings, we used the given mapping for both training and testing set with the same additional embeddings.

For our experiments with RNNs, we started with a bidirectional LSTM as a base. A bidirectional LSTM has 2 layers, each reading the text from an opposite direction. The result is then concatenated and fed into the classification layer. The main motivation was that words are influenced by both previous and future context. The layer and batch size were picked empirically to fit into the video RAM of our training GPU. We set the LSTM layer size to 256, batch size to 512 and trained the network for 25 epochs. We chose Adam as the optimizer.

Chapter 6

Experiments

We did several experiments by incrementally adding features. We opted for this approach to better see the impact of each feature category. We could not perform permutation tests to measure the statistical significance due the time it takes to train the models.

For CoNLL2003, we used the predefined tags given in the dataset: Person, Location, Organization and Misc with the BIO scheme. We then evaluated the precision, recall and F1 score.

To be inline with the papers on CNEC2.0 [39, 38] , we evaluated the precision, recall and F1 score based on three definitions of correctly detected NEs.

- Both the span of the NE instance was detected correctly and a correct NE type tag was assigned. (Type)
- The span of the NE instance was detected correctly and a correct supertype tag (i.e., the first character of the NE type tag) was assigned. (Supertype)
- The span of the NE instance was detected correctly. (Span)

We trained all CNEC2.0 models using the modified BILOU scheme in [38], on all 46 tags. We also trained separate models for supertypes and only the span, but opted to use the same metric as in literature [38].

6.1 CRF

6.1.1 Baseline Approach

For our baseline approach, we used the word itself along with its neighbours in a window of size 1 and 2 as a feature function. This amounted to 19,711 feature functions after feature selection. The performance was heavily skewed toward precision, as it basically tried to match the words. We believe the performance of a label-only CRF system could be improved by omitting feature selection, which would allow us to keep all words in the training set. However, this would lead to the number of features being more than ten times larger. The total number of features would be over 300k. We also note the performance did

Table 6.1: CRF with the current and neighbouring word tokens

	Precision	Recall	F1 score
Type	79.74	22.88	35.56
Subtype	82.04	23.55	36.59
Span	85.63	24.99	38.68

not jump between the subtasks. Detecting the span (just the beginning and content of the named entity) is only marginally better than detecting the type (classify all 46 classes, along with the correct span).

6.1.2 Morphological features

Afterwards, we morphological features, such as *is_ capitalized*, *contains_ at* and *contains_ digit* suffixes and prefixes of length two and three, the word shape described in 4.2.5. This almost doubled the performance. We can see the increase mainly in recall.

Table 6.2: CRF with the current and neighbouring word tokens and morphological features

	Precision	Recall	F1 score
Type	69.68	51.72	59.37
Subtype	75.46	56.03	64.31
Span	83.18	62.78	71.56

6.1.3 POS tags

Adding POS tags yielded comparably small improvements compared to the morphological features.

Table 6.3: CRF with the current and neighbouring word tokens and morphological features and POS tags

	Precision	Recall	F1 score
Type	70.78	53.75	61.10
Subtype	77.04	58.52	66.51
Span	85.82	66.28	74.79

6.1.4 Brown clusters, gazetteers

Finally, we added Brown clustering at different granularities (4, 8, 12 and 20) to better deal with out of vocabulary words and gazetteers to add external information. This lead to our currently best result. We also tried the same configuration without features, which lead to a model with 500k features instead of 40k, for a minimal (less than 1.0 F1 score among all categories) performance increase. However, the actual model sizes were 2.57MB and 33.5MB, which is negligible.

Table 6.4: CRF with the current and neighbouring word token, morphological features, POS tags, Brown clusters and gazetteers

	Precision	Recall	F1 score
Type	76.33	58.36	66.15
Subtype	82.36	62.99	71.38
Span	88.20	68.58	77.16

6.2 RNN

6.2.1 Baseline Approach

For our baseline with recurrent neural networks, we decided on a single bidirectional LSTM layer with untrained embeddings. We chose 300 as the embedding dimension to be comparable with our pretrained skipgram embeddings. The performance was poor, but better than our CRF baseline.

Table 6.5: 1 bidirectional layer and untrained embeddings

	Precision	Recall	F1 score
Type	45.75	28.75	35.31
Subtype	53.62	33.72	41.40
Span	70.80	45.25	55.21

6.2.2 Two bidirectional LSTM layers

Afterwards, we doubled the number of parameters by adding a second bidirectional LSTM layer. The main motivation for that was to see whether the neural network could learn a more complex data representation. Deeper LSTM networks were also said to have better performance in machine translation tasks [42]. However, for our task, this proved to be too many parameters for the network to learn anything in 25 epochs leading to worse performance in all tasks.

Table 6.6: 2 bidirectional LSTM layers and untrained embeddings

	Precision	Recall	F1 score
Type	38.64	27.43	32.09
Subtype	46.43	32.97	38.56
Span	65.79	47.50	55.16

6.2.3 GRU + batch normalization

After seeing the performance decrease, we decided to try more advanced RNN cell implementations instead of increasing the model size. We thus employed GRU cells that contain a lower number of parameters to learn. Additionally, we employed batch normalization. The performance rose considerably, especially with the types, which was the most difficult task. Seeing the performance increase, we decided to employ GRU cells with batch normalization in all subsequent experiments.

Table 6.7: 1 bidirectional GRU layer, batch normalization and untrained embeddings

	Precision	Recall	F1 score
Type	52.14	33.19	40.56
Subtype	58.25	37.10	45.33
Span	71.88	45.36	56.60

6.2.4 Pretrained word embeddings

After using the provided pretrained word2vec embeddings, the performance rose again leading us to believe that the pretrained word embeddings do contain relevant semantic and syntactic information which is useful for classification. The small dataset did not allow us to learn a sufficiently powerful representation. By adding the pretrained model (a form of transfer learning), we improved the F1 score by over almost 10 points on types.

Table 6.8: 1 bidirectional GRU layer, batch normalization and pretrained embeddings

	Precision	Recall	F1 score
Type	58.32	43.44	49.79
Supertype	63.17	47.06	53.94
Span	70.74	53.58	60.98

6.2.5 Adding POS tags

We added the POS tag of each token by concatenating a one-hot encoded vector. The idea was twofold. Firstly, we wanted to add more sentence-specific information instead of the averaged information contained in the embeddings. Secondly, we wanted to add more information to out-of-vocabulary words. The performance gain was similar to the one we saw when we added POS tags to our CRF model.

Table 6.9: 1 bidirectional GRU layer, batch normalization and pretrained embeddings, along with concatenated POS tags

	Precision	Recall	F1 score
Type	59.58	46.59	52.29
Supertype	64.92	50.79	56.99
Span	72.06	57.31	63.84

6.2.6 Concatenating additional features

We tried adding gazetteers (last name, first name, street name) and further boolean features as a single one-hot vector. This lead to our best result. We appended the features to the embedding before the recurrent layer. We also tried to append it after the recurrent layer, before the classification layer, but this lead to worse results.

Table 6.10: 1 bidirectional GRU layer, batch normalization and pretrained embeddings, along with concatenated POS tags and boolean features

	Precision	Recall	F1 score
Type	62.38	48.25	54.42
Subtype	66.94	52.51	59.21
Span	74.75	58.79	65.82

6.2.7 Word convolution

Another experiment was to employ 1D convolution on words to force short-term relations using a 1D convolution on the word embeddings before the RNN. The main benefit of a convolutional layer is that it is spatially invariant. The idea was that that the filters would learn word tuples (regardless of their position in the sentence) and their relevance. However, the performance decreased. We believe it was due to our choice of concatenating the results of the convolution, increasing the dimension if the RNN input dramatically. We note the improved precision at the cost of recall.

Table 6.11: 1 bidirectional GRU layer, batch normalization and pretrained embeddings, concatenated POS tags and boolean features, word convolution

	Precision	Recall	F1 score
Type	65.50	39.69	49.43
Subtype	70.84	42.94	53.47
Span	77.84	47.96	59.35

6.2.8 Character level features

Finally, we tried to add character level information using embeddings. We extracted the characters of each word and embedded each them into a 32-dimensional vector. The resulting matrix was four-dimensional (batch × words_in_sentence × chars_in_word × char_embedding). We then tried two approaches. We performed character convolution and we tried encoding the characters themselves into quasi-word embeddings using a RNN architecture. Neither of those showed significant performance increase, but we attribute that mainly to hyperparameter and architectural decisions. Appending additional features also led to an increase of trained parameters which might have been too much for our small task. We believe the results would be better if we could pretrain the character embeddings on a different task beforehand.

6.2.8.1 Character convolution

Our decision to try out character convolution was mainly inspired by language modelling research [49, 19, 4], which showed improved performance while using a lower number of trainable parameters. We used a convolutional layer and concatenated the resulting vector to the embedding. We had to use 2D convolution due to the added dimension and reshape and permute the character input appropriately. We believe

Table 6.12: 1 bidirectional GRU layer, batch normalization and pretrained embeddings, concatenated POS tags and boolean features, character convolution

	Precision	Recall	F1 score
Type	56.90	44.69	49.74
Subtype	53.02	48.94	55.10
Span	70.98	56.06	62.64

6.2.8.2 Character-level word encoding

Another approach was to encode each character embedding into a quasi-word embedding using a small RNN network and append that to our pretrained embeddings as an additional feature. The main issue was mainly the encoding the character embeddings into words. We padded each quasi-word to a length of 15, which was probably too much and resulted in many quasi-words being padded with zeros. Additionally, the dataset was too small and we believe the 25 epochs were not enough. We believe the next step would be to remove the word embeddings and train on character embeddings exclusively.

Table 6.13: 1 bidirectional GRU layer, batch normalization and pretrained embeddings, concatenated POS tags and boolean features, character encodings

	Precision	Recall	F1 score
Type	59.86	40.44	48.27
Subtype	66.21	44.74	53.40
Span	74.78	51.37	60.90

6.3 CoNLL2003

We did two smaller experiments on the CoNLL2003 dataset. We used a less extensive feature set (no external information or lemmatization, which was language specific) than the one used for CNEC2.0, but got reasonable results regardless. For CRFs, we used all morphological features (suffixes, prefixes, capitalization, word shape) along with the given POS tags and word chunk tags. This improved over the baseline considerably but did not reach the current state-of-art at 88.76 F1 score. We would require character level features [21] and English language gazetteers to match. We note the best result using a CRF was 84.04 F1 score [25]. We believe the implementation of ensemble classifiers [10] to be out of scope of this work as it is a meta learning approach instead of a concrete sequence modelling technique.

For RNN, we used the same model as our baseline (1 bidirectional GRU layer, untrained embeddings with dimension 300, one-hot encoded POS tags). The results for RNNs were very poor and we did not experiment with different neural network architectures due to time constraints. We believe the poor performance was caused by a combination of embedding size and the dimension of the recurrent layer.

Table 6.14: CoNLL2003 task comparison

	Precision	Recall	F1 score
baseline	71.91	50.90	59.61
CRF	79.20	76.88	78.02
RNN	52.52	58.06	54.20
State of art	88.99	88.54	88.76

6.4 Results comparison

Finally, we show a table comparing our experiments on the CNEC2.0 dataset with the current state-of-art. We omitted the convolutional experiments as their performance was not a big

Table 6.15: Comparison of F1 scores on types of all our models.

model	F1 score on types
RNN baseline	35.31
RNN, GRU + bnorm	40.56
CRF, words	48.27
RNN, pretrained	49.79
RNN, POS, fts	54.42
CRF, morphological features	59.37
CRF, POS tags	61.10
Decision trees [36]	62.00
CRF, all fts	66.15
State of art [39]	79.23

improvement.

Chapter 7

Conclusion

To summarize this work, we researched the current methods for sequence classification in general and in named entity recognition and then decided to look at two concrete examples in detail. We discussed the sequential nature of text data and how to model it. We framed the linear-chain CRF as an optimization task, where we perform maximum likelihood estimation of the parameters. We then described the used feature functions, which we handpicked based on intuition and literature. For neural networks, we started with the general description of feed forward networks along with our research on common challenges and their solutions. We did it as we encountered these issues frequently while training own neural networks. Afterwards, we went to describe a modification of the feed forward architecture, called recurrent neural networks, to learn long-term relations.

Then we implemented a NER system that extracts features from both the CNEC2.0 and ConLL2003 dataset, and performed several experiments for both classifiers. With conditional random fields, we were mainly interested in how the features directly impact classification performance. We started with local features, such as the word or neighbouring words. Afterwards, we added morphological features which described the words better. We noted a significant increase of performance there, which led us to believe that the rich morphological structure of the Czech language needs to be considered when doing text classification. Finally, we added POS tags to model word relations in the sentence and gazetteers together with brown clustering to add external information. We were briefly interested the number of generated features as well, as word distributions follow the Zipf's law [31], we could remove a large number of them without significantly affecting performance. From our CRF experiments, we gained the insight about the relevance of feature quality.

In recurrent neural networks, we mainly experimented with different training methods and ways to represent the data. We compared the baseline LSTM model with more advanced GRU and batch normalization and noted the considerable performance improvement along with faster training. Afterwards we employed pretrained word embeddings as a form of transfer learning. Then we tried external features such as POS tags. Finally, we tried more novel approaches using word convolution and character-level features. We were mainly inspired by the performance gain seen in the CRF module. However, we were unable to outperform our previous models or reach the state of art, probably because of a combination of hyperparameters and structural decisions. We mainly believe the word embedding dimension of 300 was too large.

Most of the experiments concentrated on the CNEC2.0 dataset as CoNLL2003 has been extensively tested using a diverse range of methods, but we outperformed the baseline using a basic feature set.

For further work, we intend to experiment further with a convolutional approach. We also want to improve the character level features as we did not have enough time to tune the architecture.

Bibliography

[1] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473, 2014.

[2] B. Baker, O. Gupta, N. Naik, and R. Raskar. Designing neural network architectures using reinforcement learning. *CoRR*, abs/1611.02167, 2016.

[3] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *Trans. Neur. Netw.*, 5(2):157–166, Mar. 1994.

[4] J. Bradbury, S. Merity, C. Xiong, and R. Socher. Quasi-recurrent neural networks. *CoRR*, abs/1611.01576, 2016.

[5] P. F. Brown, P. V. deSouza, R. L. Mercer, V. J. D. Pietra, and J. C. Lai. Class-based n-gram models of natural language. *Comput. Linguist.*, 18(4):467–479, Dec. 1992.

[6] X. Carreras, L. Màrquez, and L. Padró. A simple named entity extractor using adaboost. In W. Daelemans and M. Osborne, editors, *Proceedings of CoNLL-2003*, pages 152–155. Edmonton, Canada, 2003.

[7] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio. Learning phrase representations using RNN encoder-decoder for statistical machine translation. *CoRR*, abs/1406.1078, 2014.

[8] F. Chollet. keras. `https://github.com/fchollet/keras`, 2015.

[9] D. Clevert, T. Unterthiner, and S. Hochreiter. Fast and accurate deep network learning by exponential linear units (elus). *CoRR*, abs/1511.07289, 2015.

[10] R. Florian, A. Ittycheriah, H. Jing, and T. Zhang. Named entity recognition through classifier combination. In W. Daelemans and M. Osborne, editors, *Proceedings of CoNLL-2003*, pages 168–171. Edmonton, Canada, 2003.

[11] I. Goodfellow, Y. Bengio, and A. Courville. *Deep Learning.* MIT Press, 2016. `http://www.deeplearningbook.org`.

[12] J. Hammerton. Named entity recognition with long short-term memory. In W. Daelemans and M. Osborne, editors, *Proceedings of CoNLL-2003*, pages 172–175. Edmonton, Canada, 2003.

[13] S. Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[14] S. Hochreiter and J. Schmidhuber. Long short-term memory. *Neural Comput.*, 9(8):1735–1780, Nov. 1997.

[15] K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251 – 257, 1991.

[16] D. H. Hubel and T. N. Wiesel. Receptive fields and functional architecture of monkey striate cortex. *Journal of Physiology (London)*, 195:215–243, 1968.

[17] A. G. O. II, T. Mikolov, and D. Reitter. Learning simpler language models with the delta recurrent neural network framework. *CoRR*, abs/1703.08864, 2017.

[18] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *CoRR*, abs/1502.03167, 2015.

[19] N. Kalchbrenner, L. Espeholt, K. Simonyan, A. van den Oord, A. Graves, and K. Kavukcuoglu. Neural machine translation in linear time. *CoRR*, abs/1610.10099, 2016.

[20] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[21] D. Klein, J. Smarr, H. Nguyen, and C. D. Manning. Named entity recognition with character-level models. In W. Daelemans and M. Osborne, editors, *Proceedings of CoNLL-2003*, pages 180–183. Edmonton, Canada, 2003.

[22] M. Konkol and M. Konopík. *Maximum Entropy Named Entity Recognition for Czech Language*, pages 203–210. Springer Berlin Heidelberg, Berlin, Heidelberg, 2011.

[23] J. D. Lafferty, A. McCallum, and F. C. N. Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *Proceedings of the Eighteenth International Conference on Machine Learning*, ICML '01, pages 282–289, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[24] P. Liang. Implementation of the brown hierarchical word clustering algorithm. `https://github.com/percyliang/brown-cluster`, 2012.

[25] A. McCallum and W. Li. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In W. Daelemans and M. Osborne, editors, *Proceedings of CoNLL-2003*, pages 188–191. Edmonton, Canada, 2003.

[26] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781, 2013.

[27] J. Mount. The equivalence of logistic regression and maximum entropy models, 2011.

[28] R. H. Nielsen. Theory of the backpropagation neural network. In *Proceedings of the International Joint Conference on Neural Networks* (Washington, DC), volume I, pages 593–605. Piscataway, NJ: IEEE, 1989.

[29] N. Okazaki. Crfsuite: a fast implementation of conditional random fields (crfs), 2007.

[30] R. Pascanu, T. Mikolov, and Y. Bengio. Understanding the exploding gradient problem. *CoRR*, abs/1211.5063, 2012.

[31] D. M. W. Powers. Applications and explanations of zipf's law. In *Proceedings of the Joint Conferences on New Methods in Language Processing and Computational Natural Language Learning*, NeMLaP3/CoNLL '98, pages 151–160, Stroudsburg, PA, USA, 1998. Association for Computational Linguistics.

[32] L. R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, Feb 1989.

[33] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988.

[34] M. Sahlgren. The distributional hypothesis.

[35] M. Ševčíková, Z. Žabokrtský, and O. Krůza. *Named Entities in Czech: Annotating Data and Developing NE Tagger*, pages 188–195. Springer Berlin Heidelberg, Berlin, Heidelberg, 2007.

[36] M. Sevcíková, Z. Zabokrtský, and O. Kruza. Named entities in czech: Annotating data and developing NE tagger. In V. Matousek and P. Mautner, editors, *Text, Speech and Dialogue, 10th International Conference, TSD 2007, Pilsen, Czech Republic, September 3-7, 2007, Proceedings*, volume 4629 of *Lecture Notes in Computer Science*, pages 188–195. Springer, 2007.

[37] M. Straka and J. Straková. Czech models (MorfFlex CZ 161115 + PDT 3.0) for MorphoDiTa 161115, 2016. LINDAT/CLARIN digital library at the Institute of Formal and Applied Linguistics, Charles University.

[38] J. Straková, M. Straka, and J. Hajič. *A New State-of-The-Art Czech Named Entity Recognizer*, pages 68–75. Springer Berlin Heidelberg, Berlin, Heidelberg, 2013.

[39] J. Straková, M. Straka, and J. Hajič. Neural networks for featureless named entity recognition in czech. In P. Sojka, A. Horák, I. Kopeček, and K. Pala, editors, *Text, Speech, and Dialogue: 19th International Conference, TSD 2016*, number 9924 in Lecture Notes in Computer Science, pages 173–181, Cham / Heidelberg / New York / Dordrecht / London, 2016. Masaryk University, Springer International Publishing.

[40] J. Straková, M. Straka, and J. Hajič. Open-Source Tools for Morphology, Lemmatization, POS Tagging and Named Entity Recognition. In *Proceedings of 52nd Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 13–18, Baltimore, Maryland, June 2014. Association for Computational Linguistics.

[41] M. Sundermeyer, R. Schlüter, and H. Ney. Lstm neural networks for language modeling. In *Interspeech*, pages 194–197, 2012.

[42] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215, 2014.

[43] C. Sutton and A. McCallum. An introduction to conditional random fields. *Found. Trends Mach. Learn.*, 4(4):267–373, Apr. 2012.

[44] M. Tkachenko and A. Simanovsky. Named entity recognition: Exploring features. In J. Jancsary, editor, *Proceedings of KONVENS 2012*, pages 118–127. ÖGAI, September 2012. Main track: oral presentations.

[45] C. Wang, H. Yang, C. Bartz, and C. Meinel. Image captioning with deep bidirectional lstms. *CoRR*, abs/1604.00790, 2016.

[46] D. Wang and E. Nyberg. A long short-term memory model for answer sentence selection in question answering. In *ACL*, 2015.

[47] D. Wu, G. Ngai, and M. Carpuat. A stacked, voted, stacked model for named entity recognition. In W. Daelemans and M. Osborne, editors, *Proceedings of CoNLL-2003*, pages 200–203. Edmonton, Canada, 2003.

[48] M. D. Zeiler, M. Ranzato, R. Monga, M. Mao, K. Yang, Q. V. Le, P. Nguyen, A. Senior, V. Vanhoucke, J. Dean, and G. E. Hinton. On rectified linear units for speech processing. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 3517–3521, May 2013.

[49] X. Zhang, J. J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification. *CoRR*, abs/1509.01626, 2015.

Appendix A

List of abbreviations

BFGS	Broyden–Fletcher–Goldfarb–Shanno algorithm
CRF	Conditional Random Fields
CNEC	Czech Named Entity Corpus
GRU	Gated Recurrent Unit
LSTM	Long Short Term Memory
MSE	Mean Squared Error
NER	Named Entity Recognition
POS	Part-of-speech
RNN	Recurrent Neural Network
SVM	Support Vector Machine

Appendix B

Installation

To install this, please install the following python dependencies using pypy:

```
pip install numpy
pip install sklearn
pip install scipy
pip install ufal.morphodita
pip install python-crfsuite
pip install tensorflow
pip install keras
```

Then you can run the CRF module by first setting up a model.txt (we provided one as an example) and running

```
python3 -m src.CRF_NER.CRF_NER -t named_ent_train.txt /
named_ent_etest.txt models.txt BILOU
```

The last parameter is the granularity, you can also use supertype and BIO. For the RNN based classifier, run:

```
python3 -m src.keras_NER.keras_NER
```

Both classifiers will output a file with `_textoutput.json` as a suffix. You can measure the performance by calling:

```
perl retokenize_and_eval.pl named_ent_etest.treex YourModel_textoutput.json
```

Appendix C

Contents of CD

```
|-- adresy.txt
|-- czech_last_names
|-- czech_names
|-- czech_stemmer.py
|-- eval
|    |-- compare_ne_outputs.gold
|    |-- compare_ne_outputs.system
|    |-- compare_ne_outputs_v3.pl
|    |-- named_ent_etest.treex
|    |-- retokenize_and_eval.pl
|-- features.txt
|-- LICENSE
|-- models.txt
|-- named_ent_dtest.txt
|-- named_ent_etest.txt
|-- named_ent_train.txt
|-- named_ent.txt
|-- thesis.pdf
|-- paths
|-- POS_final.json
|-- README.md
|-- src
|   |-- common
|   |   |-- eval.py
|   |   |-- feature_extractor.py
|   |   |-- __init__.py
|   |   |-- NER_utils.py
|   |-- CRF_NER
|   |   |-- CRF_NER.py
|   |   |-- __init__.py
|   |-- __init__.py
|   |-- keras_NER
|   |   |-- keras_NER.py
|-- tag_indices_bilou.json
|-- token_indices.json
```