FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF MASTER'S THESIS

Title:	Semantic understanding of natural conversation
Student:	Bc. Petr Lorenc
Supervisor:	Ing. Jan Šedivý, CSc.
Study Programme:	Informatics
Study Branch:	Knowledge Engineering
Department:	Department of Applied Mathematics
Validity:	Until the end of winter semester 2019/20

Instructions

The conversational AI applications are recently gaining customer interest. This project focuses on finding the best algorithm for semantic understanding of natural utterances in a social bot. The theses will first define the intent and entity in dialog utterances. Use the definition of the intent and entity for tagging real conversations supplied by the supervisor. Next, identify and test sequential machine learning algorithms for intent and entity recognition. The final result of the theses is a detailed comparison of the selected algorithms regarding accuracy, memory requirements, computational complexity, and speed of convergence on a tagged training set.

References

Will be provided by the supervisor.

Ing. Karel Klouda, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague August 20, 2018

FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

Master's thesis

Semantic understanding of natural conversation

Bc. Petr Lorenc

Department of theoretical computer science
Supervisor: Ing. Jan Šedivý, CSc.

January 8, 2019

Acknowledgements

Firstly I would like to thank my family for their support during all my studies and especially while writing this thesis. Also, I would like to thank to my supervisor Ing. Jan Šedivý, CSc. for the possibility to work on this thesis and many thanks also to the team of eClub, especially to Ing. Jan Pichl for advice and support with this thesis.

Declaration

Citation of this thesis

Lorenc, Petr. *Semantic understanding of natural conversation.* Master's thesis. Czech Technical University in Prague, Faculty of Information Technology, 2019.

Abstrakt

Tato diplomová práce se zaměřuje na nalezení nejlepšího algoritmu pro sémantické porozumění přirozených konverzací. Výsledný algoritmus lze použít například u chatbotů. Součástí práce je dataset, který vznikl pro potřeby chatovací aplikace Alquist a který byl vytvořen na základě skutečných konverzací. Práce prozkoumává sekvenční algoritmy strojového učení pro rozpoznávání intentu a určování entit. Výsledkem práce je detailní porovnání přesnosti, požadavků na paměť a rychlosti konvergence vybraných algoritmů. Na základě porovnání algoritmů je v práci navrhnut nový model, který se zakládá na propojení rozpoznávání intentu a určování jmenných entit.

Klíčová slova klasifikace intentu, určování entit, strojové učení, zpracování přirozeného jazyka

Abstract

This master's thesis aims to find the best algorithm for semantic understanding of natural dialogs. The result can be used in a conversation AI. A part of the thesis is also a dataset, based on needs for a chatbot application Alquist and which was based on real conversations. The thesis also identifies and examines sequential machine learning algorithms for intent and entity recognition. The result of the thesis is a detailed comparison of the selected algorithms regarding accuracy, memory requirements and computational complexity. Based on the results, a new model which joins intent and entity recognition together is created.

Keywords intent recognition, entity recognition, machine learning, natural language processing

Contents

List of Figures

List of Tables

Listings

Introduction

Motivation and objectives

The usage of conversation AI[1] (chatbots) is rapidly growing up, more and more people are getting used to speaking with conversational devices (Amazon Alexa, Apple Siri, Google Assistant). The most significant use of conversation AI is in the call centers. The chatbots are used to cooperate with humans to fasten the conversation or replace the human for providing time unlimited service.

The conversation AI system is usually text-based. We do not have to focus on an outside appearance and hardware of that system (skeleton, microphone, speaker, . . .). The conversation AI should cooperate with ASR (speech-to-text) and speech synthesis (text-to-speech). In the NLP module, which is a central component of the conversation AI system, we classify intent, do named entity recognition, analyze the sentiment and many more – with the aim to get as much valuable information from user utterance as possible.

The intent and entity recognition are essential parts of the conversation AI system because, for example, we have a sentence "I would like to move to London." We would like to extract the information that the intent is "change address" and also recognize, that the entity is "London". In some cases, for example in chatbot system Alquist[2], we require to catch not only the true named entities defined here [16] but also pseudo-entities. A good example of pseudo-entity is "rock music" in a sentence "Let's listen to rock music."

All predictions have to be in a real-time. This master's thesis will focus on creating a new dataset, based on the original algorithm, which combines the automatic generation process and manual labeling. Then we will evaluate new techniques of joint intent and entity recognition in comparison with techniques which are dealing with intent and entity recognition separately.

[1] Artificial Intelligence
[2] http://alquistai.com/

We will implement the prototype of the best model into the chatbot system Alquist.

Structure

Firstly, the chapter “Natural Language Processing” defines what we call an entity and intent in the context of the conversational AI system Alquist. The next chapter “Methods” introduces machine learning principles for used models. The chapter “Experiments” is about introducing several already existing algorithms and measure their results on an automatically generated data. That chapter also describes a process of creation generated data. We introduce the new joint model for entity and intent recognition, and we evaluate its performance regarding accuracy, speed and computational requirements. The last part, the conclusion, is focused on the results and possible future work.

Chapter 1

Natural Language Processing

Natural Language Processing (NLP) is the scientific field which is getting more and more important nowadays. It combines probabilistic and machine learning methods with human interaction. The NLP is very broad and includes for example question answering, language translation, text summarizing and also generating a text which describes an image. We can choose three main approaches for processing text:

- *Hand-written rules* — This method was very popular at the beginning of 80's[17]. It had a big advantage in absolute control over output, so it was much easier to find a problematic parts and repaired them. On the other hand, it was exhausting to comprehend all possible conditions. The typical rule could be written like this: 'If I found text in a database then it is an entity.' or 'If the text starts with a capital letter then it is an entity.'

- *Statistical approach* — With the rise of computation power the algorithms transfer their base computation from humans to computers. It is the reason why we can use Conditional Random Fields, Perceptron or neural network[18]. The computation demand for training neural network is limiting. It is also the reason why big companies are focusing on developing more powerful graphics units, which are used for that task.

- *Combination of both* — At the beginning of the statistical approach was very popular to use *Decision trees* which combines both approaches. Their output is in the form of "If-then-else rules", but they are learned automatically based on data and their labels.

We will focus mainly on the *statistical approach*, where we are dependent only on the quality of our data and architecture of a model and more or less independent of human fine-tuning.

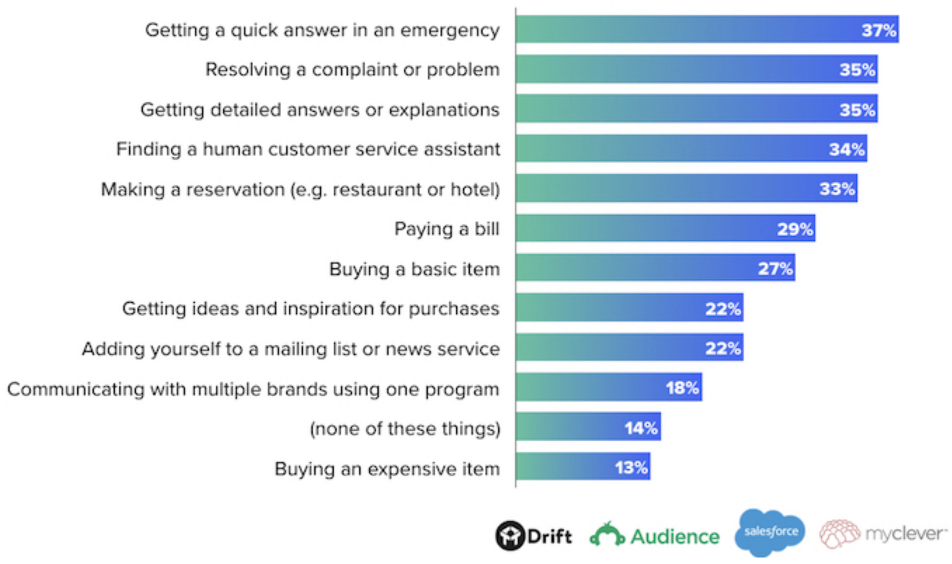


Figure 1.1: Predicted fields of chatbot usage[1]

1.1 Chatbots

From the beginning of human's interaction with the computer, there was a desire to be able to communicate with them with natural language. Between pioneers, we can count ELIZA [19] which was developed by Weizenbaum in 1960s and was based on a set of hand-written rules. Today's popularity of chatbots is getting bigger thanks to the system like Amazon Alexa, Google Home or Siri in Apple systems which can have more than 100 millions of users worldwide. The predition of chatbot usage can be seen on Figure 1.1. It is based on an online survey with thousands of people.

We can divide the usage of chatbots into several categories:

- Task-oriented conversation — The purpose of the conversation is to make a certain task (reservation of a hotel, book a flight, find a piece of information...)
- Chit-chat conversation — The purpose is not much specific, and there are only a few requirements for that system. Unfortunately, it means that it is much harder to create that system. The conversation should be engaging for a human, it should be about whatever topic (or at least broad spectrum of topics) the user want (the open domain) and it should also be persistent across conversation turns (for example it should not say that it likes the band Linkin Park and in the next utterance says that does not)

According to [20] more than 20 billions of turns of conversation was made until 2017 on the platform XiaoIce, which is a chatbot released by Microsoft for Chinese customers. Each conversation has 20 turns on average. It sounds like very promising data for future development in chatbots' field.

Usually chatbot gets a *query* q and system returns a *response* r. Respond model has usually two options:

- Generation-based system — Based on Statistical Machine Translation (SMT) and big amount of data, we can train generating system (Encoder-Decoder model). *Encoding* (based on Recurrent Neural Network (RNN)) is a function which take current input x_t and previous hidden state h_{t-1} to produce current hidden state h_t. *Decoding* is in a certain way reverse process. It takes previous output y_{t-1} and current hidden state h_t to produce current output y_t and next hidden state h_{t+1}.

- Retrieval-based system — Main of this system is searching for a mapping from q to r. It can be based on similarity between vectors of q and r (TF-IDF, one-hot or output representation of RNN like Long Short Term Memory (LSTM) or Gated Recurrent Unit (GRU) and nowadays even Convolutional Neural Networks (CNN)).

1.2 Learning algorithms types

We can divide learning algorithms into four types based on a dataset we have:

- Supervised learning — The dataset is in a form where we know output for given input in advance. The supervised learning is a technique to create a probabilistic model, which can predict probability distribution over input based on features extracted from it[21].

- Semi-supervised learning — The part of dataset is labeled and the algorithm is trying to apply that knowledge on unlabelled data.

- Unsupervised learning — The dataset is in a form where we know only input, and we do not know the output.

- Reinforcement learning — We does not have an feedback for a given input, until end of an experiment. For example, in computer games, we will make a move (an input), and we get a score (an output) at the end of the game (an experiment).

Difference between supervised and unsupervised learning can be seen on Figure 1.2.

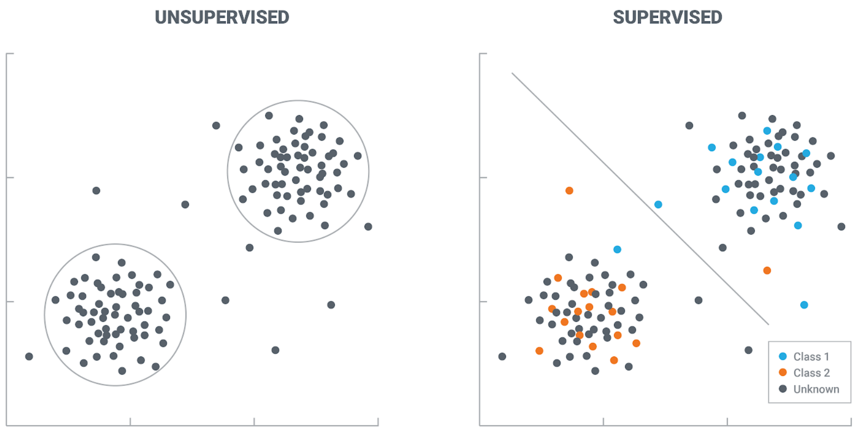


Figure 1.2: Supervised and Unsupervised learning[2]

1.3 Natural language processing fields

The NLP is included in most of today's system which interacts with humans. We can utilize it for evaluating semantic (positive or negative attitude) of comment over a product to know which one is problematic. We can extract popular topic from news and rewrite them exactly for the need of the consumer. Translation is another very hot topic following by chatbots. It is impossible to give the full list of all NLP fields, but there are just some of then which we found in several basic systems:

- *Syntax based - Lemmatization, Part-of-Speech (POS), Stemming, Word segmentation*
- *Named Entity Recognition (NER)*
- *Machine translation*
- *Intent recognition (IR)*
- *Sentiment analysis*
- *Question answering*
- *Natural language generation*
- *Text summarization*
- *Text-to-speech and Speech-to-text*

1.4 Entity recognition

Our system has to be incorporated with already created and used system Alquist, so we have to make some retreats from the common definition of *Named Entity.* 'In the expression "Named Entity", the word "Named" aims to restrict the task to only those entities for which there are one or many rigid designators. Rigid designators include proper names as well as certain natural kind terms like biological species and substances.'[16] So, in a sentence *"I would like to move to London"*, we can extract *London* as a named entity. It is useful for searching in knowledge graphs or databases. On the other hand, if we use that definition in a sentence like *"I want to listen to rock band Guns N' Roses"* we extract only **Guns N' Roses** which can be enough in a case that recognition will be accurate. However, if we imagine a situation where we have only a small database of rocks band, and there is no record of "Guns N' Roses". Then we have two choices. First is to respond *"We don't know that band"* to the user. The other possibility is to extract also some helpful information from the sentence like a word **"rock"**. If a system (and especially a database/knowledge graph) counts with this possibility then it can search for all bands which are playing rock music and randomly choose one of them resulting in a much more pleasant answer: *"I am sorry. I don't know that rock band but I can play you some other rock band. For example **randomly chosen rock band**"*. We will call the word **"rock"** our pseudo-entity.

The approach in the Alquist is much more closer to the latter approach. Fortunately, all this is mainly based on a given data. We will concern mainly on supervised machine learning techniques which are resistant for changing the definition of a named entity as long as we provide new suitable data. However, we will keep that in mind, while creating our dataset.

We will also avoid using the gazetteer[22], because of our loose definition of the entity. The gazetteer is a usually just a list of entities which we use for searching. In our case, the entity can be for example **rock** (meaning a music genre), but with using a gazetteer it will also catch **rock** (stone).

1.4.1 Data

We will stick with the commonly used format of data and use IOB encoding (an example on Listing 1.1) according to [21]. ***I*** stands for Inside Entity, ***O*** stands for Outside Entity and ***B*** means Beginning of Entity.

```
        I     0     0
        like       0     0
        John       B     B-actor
        Wayne      I     I-actor
        .     0     0
```

Listing 1.1: Example of dataset for named entity recognition

Other options are:

- *IO Encoding* — ***I*** stands for Inside Entity and ***O*** stands for Outside Entity which leads in loosing information if there will be two entities next to each other

- *BMEWO Encoding*[23] — Similar like IOB but add ***E***nd-of-entity tag, ***M***id-entity tag and single-token-entity tag (***W***)

- *XML marking* — Can be useful to store data in this format because possibility of adding further information about the entity but not very usable for evaluating and training models

1.5 Intent recognition

The other way to know about the goals of the user is to recognize his intent. In the Alquist case (and many others [13] [24][25]) is very useful to know the intent in a case when we also need to extract entities. For example, in an sentence *"Can you please play some rock music"* where we determine that the intent is *play_music*, then the entity should be *rock*. But if we get intent *tell_news* in a sentence *"What can you tell me about rock music"*, then the entity should be *rock music*. Because we need to look for further information about *rock music* in the knowledge base or the database - and if we look only for *rock* we can also get information about a stone.

As in the case of NER we will create methods which can be easily transferred to the broader range of usage and depends only on data. So, all calculation will be dependent only on the provided data and more or less independent on the definition of intent. The independence also has a big advantage in a way that we can compare algorithms on different datasets.

The intent recognition can also be described as a task of sequence classification. We get a sequence of words (the order is important), and we want to make a prediction (assign one label from the final set of possible outputs). This is very similar to Sentiment analysis because we can easily transfer our data between these two tasks. One disadvantage is that we need several separate models for each intent. So we will avoid it. Our typical datasets look like in Listing 1.2.

```
    I want to talk rock music.    __label__play_music
    Let's talk about politics.    __label__tell_news
    What is the weather today?    __label__tell_weather
    ...
```

Listing 1.2: Example of dataset for intent recognition

1.6 Metrics

Because we are working with sequential data, upon standard metrics, we need also to include a metric which takes the whole sentence into account. If we look into common NER dataset we will see that it is usually strongly unbalanced. So it is very common that if we predict *non-entity* everywhere we can reach even 90% precision with 100% recall for *non-entity* class. The results are quite high but the model is useless. We can fight this by looking for precision and recall for all of the classes and then do micro/macro/weighted average of scores. This phenomenon usually does not occur in a data for intent recognition. Next possible solution is to define a new metric which takes into account the sentence as a whole:

$$M = \frac{\sum^{s} 1 \text{ if } \forall x \in I_s, \forall y \in T_s : \text{tag}_x == \text{tag}_y \text{ else } 0}{\# \text{ sentences}} \tag{1.1}$$

where s is the current sentence, I_s is a set of correct labels for a sentence s and T_s is a set of predicted labels for a sentence. This attempt can be useful when even a little mistake can cause a big troubles.

In our case we will stick with more common practise and measure the performance for NLP task in F1 Score[26]. The limitation is that it can be measured only on binary problems (negative/positive, zero/one ...), but it can be calculated for each class separately and then average them together.

$$F = \frac{2 \cdot \text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}} \tag{1.2}$$

where *precision* and *recall* are defined:

$$\text{precision} = \frac{\text{true positive}}{\text{true positive} + \text{false positive}} \tag{1.3}$$

$$\text{recall} = \frac{\text{true positive}}{\text{true positive} + \text{false negative}} \tag{1.4}$$

where *true positive* means items where model gives the right answer, *false positive* are items where model give an correct answer but should give negative and *false negative* is where model give a negative one, but it should be positive. More on Figure 1.3.

We will use an scikit-learn implementation of weighted F1 Score[3] where each label is weighted by its support.

[3]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1$_score.html$

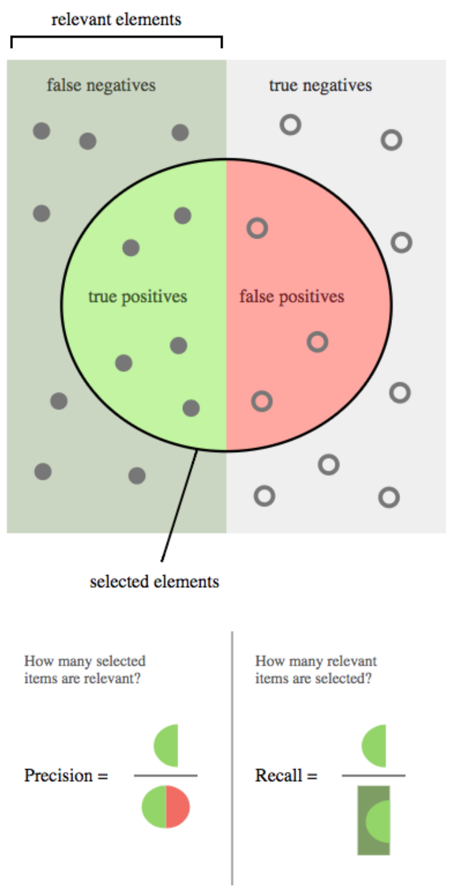


Figure 1.3: Precision and recall[3]

Chapter 2

Methods

2.1 Statistical models

A statistical model represents, often in considerably idealized form, the data-generating process. Usually thought of it as a pair (S, P) where S is the set of possible observations (the sample space), and P is a set of probability distributions over S. Because we cannot get an infinite amount of data, we are claiming that our model can lead to only a simplification or approximation of reality[27].

2.1.1 Markov chain

To get to more sophisticated algorithms, firstly we need to define what is *Markov Chain.* Markov chain is a model, which is telling us a probability of certain sequence of states. It can be easily described by graph on Figure 2.1 or with set of states $S = (s_1, s_2, ..., s_N)$, initial state s_s (or an initial probability distribution over states) and a transition probability matrix $A = (a_{11}, a_{12}, ...a_{N1}, ..., a_{NN})$ where a_{ij} means probability of transition from s_i to s_j, so $P(s_i|s_{i-1}) = a_{ii-1}$. There is also a condition that $\sum_{i=1}^{i=n} a_{in} = 1$ for $\forall n$ and Markov assumption which will be described below.

Markov assumption tells us that process is memory-less which mean that probability of transition to s_i is based only on s_{i-1} and not previous states. We can write:

$$P(s_i|s_1, ..., s_{i-1}) = P(s_i|s_{i-1}) \tag{2.1}$$

2.1.2 Hidden Markov Model

Hidden Markov models (HMM) are a powerful probabilistic tool for modeling sequential data. We can assume that from each state we can emit an observation. The observations are usually modeled as multinomial distributions over

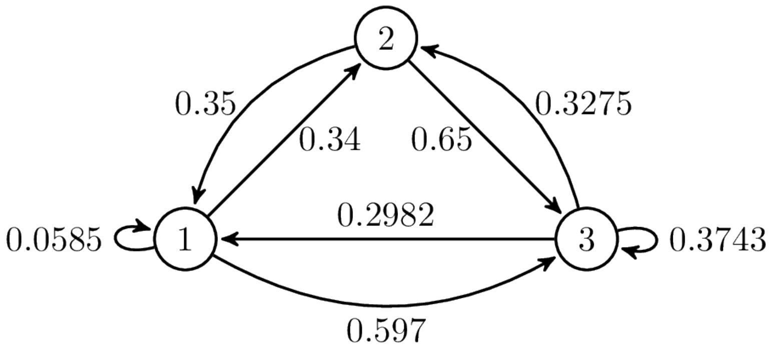


Figure 2.1: Approximation of Markov chain[4]

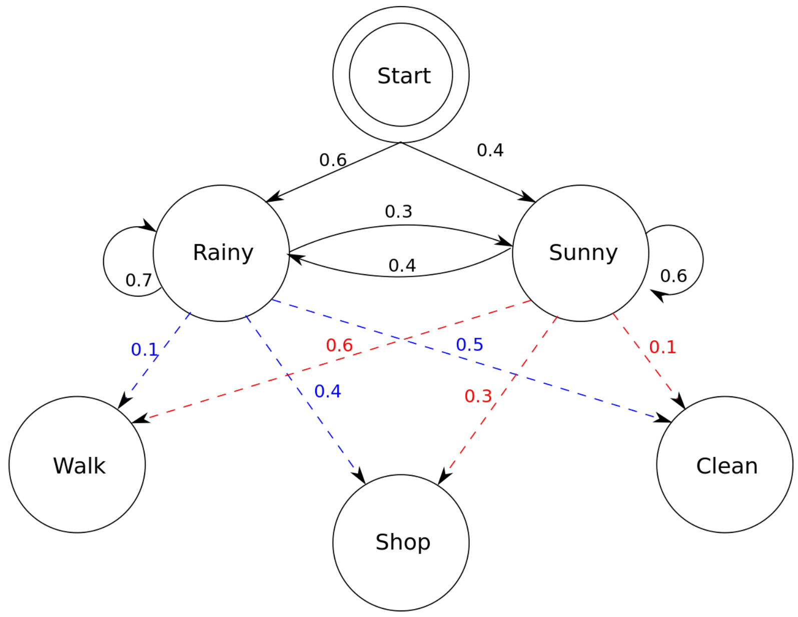


Figure 2.2: Hidden Markov Model[5], where **Walk**, **Shop** and **Clean** are visible states and **Rainy** and **Sunny** are hidden states

a discrete vocabulary, and the HMM parameters are set to maximize the likelihood of the observations. For that we need to define $O = (o_1, o_2, ..., o_t)$ which is a set of possible observations and emission matrix $E = (e_{11}, e_{21}, .., e_{n1}, ..., e_{nt})$ which define probability of observation from O based on state S. We can also write $e_{ij} = P(o_i|s_j)$. The Figure 2.2 show base of Hidden Markov Model. The word *hidden* is used because usually we don't see states but the observations is all information which we get. Then we can try to predict most probable sequence of states based on the observation. The HMM is generative model which means that a model describing how a series of observations O^t can probabilistically generate a series of states S^t.

$$\begin{aligned} P(O,S) = P(S,O) = \\ = P(s_1, s_2, ..., s_N, o_1, o_2, ..., o_T) = P(s_1) \cdot \prod_{n=2}^{n=N} P(s_n|s_{n-1}) \cdot \prod_{n=1}^{n=T} P(o_n|s_n) \end{aligned} \tag{2.2}$$

Normally, we will use three function, when we work with HMM:

- Forward algorithm
- Viterbi algorithm - the most probable sequence of states which can emit given observations
- Baum-Welch algorithm - calculate matrix A and E from given states and observations

2.1.3 Conditional Random Fields

The HMM was a generative model for sequences, the Conditional Random fields (CRF) is the discriminative model for sequences. It works in the following way. We train our model to predict a set of states directly from a set of observations. The generative and discriminative models can be converted to each other by Bayes's rule[28]. In practice, the approaches are distinct, each with potential advantages and disadvantages of each approach. The Linear-chain CRF is kind of restriction of CRF which counts only with current and previous observations.

General form looks very similar to logistic regression[29]:

$$\begin{aligned} P(\mathbf{O}|\mathbf{S}) &= \frac{1}{Z(\mathbf{S})} \prod_{i=1}^{n} \phi(o_i, o_{i-1}, S) \\ &= \frac{1}{Z(S, \lambda, \mu)} \exp \sum_{i=1}^{n} \sum_{k} \lambda_k f_k(o_i, o_{i-1}, \mathbf{S}) + \sum_{l} \mu_l g_l(o_i, \mathbf{S}) \end{aligned} \tag{2.3}$$

where $Z(\mathbf{S}, \lambda, \mu)$ is a normalization constant:

$$Z(\mathbf{S}, \lambda, \mu) = \sum_{o} \exp \sum_{i=1}^{n} \sum_{k} \lambda_k f_k(o_i, o_{i-1}, \mathbf{S}) + \sum_{l} \mu_l g_l(o_i, \mathbf{S}) \tag{2.4}$$

and f and g are feature function which return 0 or 1.

The CRF is supervised method so we need a train data of size D in a form $\{o_d, d_d\}_{d=1}^{D}$. Where o_d is our d-th observation (sometimes called y_d) and s_d is our d-th state (sometimes called x_d). Learning CRF means find the optimal parameters λ^* and μ^* through gradient updates.

$$\lambda^* \mu^* = \arg\max_{\lambda\mu} \prod P(o_d | s_d, \lambda, \mu) \tag{2.5}$$

Then for inference we used given λ^* and μ^*:

$$y^* = \arg\max_{y} \exp(\sum_{i=1}^{n} (\sum_{k} \lambda_k f_k(o_i, o_{i-1}, \mathbf{S})) + \sum_{l} \mu_l g_l(o_i, \mathbf{S})) \tag{2.6}$$

The CRF as HMM are very commonly used for a labelling sequences.

2.2 Neural-based approaches

According to [30] we can divide neural networks into several generation:

- Based on *perceptrons and threshold approaches* with binary output — for example Hopfield nets and Boltzmann machines
- Based on *activation functions* — for example feedforward neural networks and recurrent neural networks
- Based on *spiking neurons* — simulating electrical potential in our brains

In this work we will focus mainly on *second generation* - RNN, which include LSTM and also on CNN.

2.2.1 Word Representation

If we want to train our model we need to represent our input (text data) to the model because it cannot classically work with text. All training can be perceived like an updating specific matrix where each element of that matrix is a number. The simplest way is one-hot encoding which supposes that we know the vocabulary in advance or that we limit the number of words we want to use. Unfortunately we lose all the semantic information and for example words “queen” and “queens” are not related at all. The other problem is in wasting the space because we have an array where almost all elements are zero.

Both problems can be solved by training a language model on a large corpus of data.

2.2.1.1 Language model

The language model is a statistical model which is telling us the probability over a sequence of words. It can be used to choose $(P(w_1, w_2, ..., w_N))$ or generate $(\forall q \in Q^N P(q)$ where Q^N is set of all possible combination of N-th words) the most probable sequence of words.

To get a basic idea about compressing word representation firstly, we need to look at a way for modeling language. The commonly used solution can be N-gram language model:

$$P(w|c) = \frac{\text{Count}(w, c)}{\text{Count}(c)} \tag{2.7}$$

where w is a word from our vocabulary and c is a context of that word (previous n words). This model returns probability. Practically the model has several limitations - assumes exact match on our context words (not suitable for morphologically rich languages like Russian or Czech) and the bigger context we assume, the more possible combinations we get which lead to more computation and data demanding system[31].

This system models words as atomic units - without the notion of similarity between words because they are represented as indices in a vocabulary. This choice has several good reasons - simplicity, robustness and the observation that simple models trained on huge amounts of data outperform complex systems trained on less data[6].

2.2.1.2 Word Representation in Vector Space

The current rise of machine learning techniques in recent years leads to one of the most promising concepts which is a distributed representation of words[32]. That means the representation of words as continuous vectors which are based on the distributional hypothesis (words with similar context has a similar meaning). The architecture is known as Skip-gram and Continuous Bag-of-Words (CBOW) model[31][6]. Both models are shallow neural networks which try to learn language model for a given corpus. The main difference between Skip-gram model and CBOW model can be seen on Figure 2.4 and in general is in assumption what is an input and what should be output.

Training complexity of Skip-gram model is:

$$O = E \cdot T \cdot Q = E \cdot T \cdot C \cdot (D + D \cdot \log_2(V)) \tag{2.8}$$

Training complexity of CBOW is:

$$O = E \cdot T \cdot Q = E \cdot T \cdot N \cdot D + D \cdot \log_2(V) \tag{2.9}$$

where E is amount of training epoch (between 3-50), T is number of word in training set (up to 1 billion), C is the maximum distance of words from which we choose R random words (so from range $<$ 1;C $>$), V is a size of

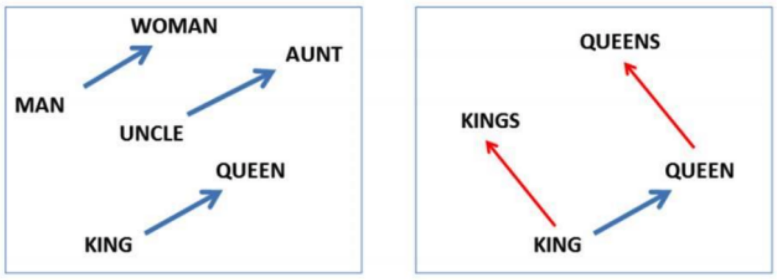


Figure 2.3: Relationship which can occur after training word2vec model[6]

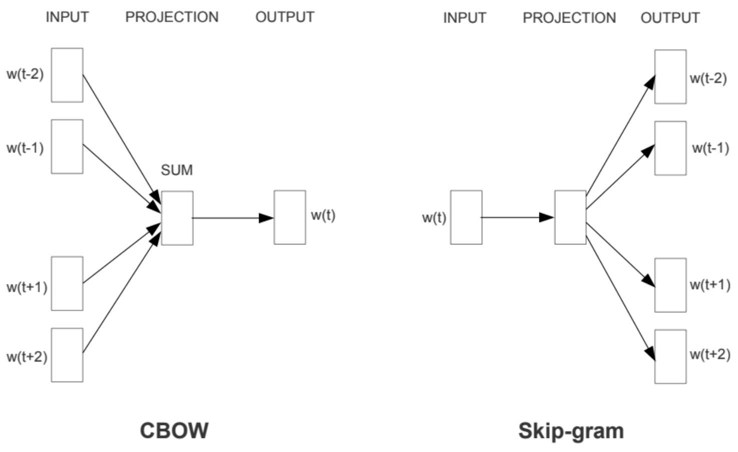


Figure 2.4: Difference between CBOW and Skip-gram model[6]

vocabulary, D is dimension of embeddings and N is the input size (a common choice is 10)[6].

The training objective of the Skip-gram model is to find word representations that are useful for predicting the surrounding words in a sentence or a document. For a sequence of training words $w_1, w_2, ..., w_T$ we are trying to maximize the average log probability:

$$\frac{1}{T}\sum_{t=1}^{T}\sum_{-c\leq j\leq cj\neq 0}\log p(w_{t+1}|w_t) \tag{2.10}$$

where c is a function of the center word w_t (getting a training context for that word). The larger value we set the more examples we get, at the expense of the training time.[33]

For computing $p(w_{t+1}|w_t)$ the authors use hierarchical softmax which was first introduced by Morin and Bengio[34] and decrease the number of needed evaluation from evaluating W output nodes to approximately $\log_2(W)$ nodes.

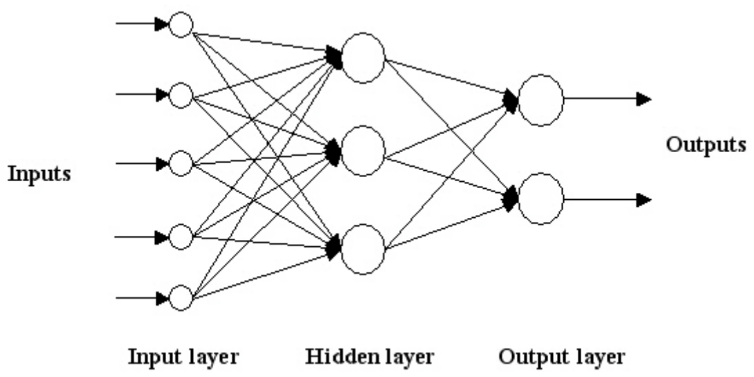


Figure 2.5: Feed-forward neural network with one hidden layer[7]

They also use binary Huffman tree to speed up the calculation for the frequent words. Another technique for speeding process up is negative sampling (there are k negative samples for which we want to minimize the probability) and subsampling of frequent words with the formula:

$$P(w_i) = 1 - \sqrt{\frac{t}{f(w_i)}} \tag{2.11}$$

where $f(w_i)$ is how many times the word w_i occurs and t is heuristically chosen threshold.

Firstly we do forward propagation for a given input (word) to get a probability of output words (over vocabulary). We know the correct context, so we use back-propagation to adjust weights to get more correct probabilities next time. After training the model, we can see preserved relationship between words (on Figure 2.3). Then, for example, if we take vector(King) minus vector(Man) plus vector(Woman) then we get a vector which is very close to vector(Queen) .

2.2.2 Neural Network

The neural network is trying to simulate what we believe our brain does. Of course, there are a lot of limitation and unknown areas. So when we talk about artificial neural networks, we mean certain approximation of our knowledge about our brain. One significant feature of our brain is the ability to *adapt* in a new behavior or ability to *learn* new skills.

We simulate this process by defining a weight matrix W which represent one layer in our network. The dimension of W is $R^{o \times i}$ where i is a size of input and o size of the output. We can stack several layers onto each other to get a more robust model as can see on Figure 2.5.

The basic equation for one layer is:

$$o_l = f(W_l i_l + b_l) \tag{2.12}$$

where o_l is output vector on l-th layer of dimension $R^{o\times 1}$, i_l is input vector on l-th layer of dimension $R^{i\times 1}$, f is a non-linear activation function (usually tanh), b_l is bias for l-th layer and W_l is weight matrix for layer l.

The values in the weight matrix have to learn. The usual way for learning is to use *back-propagation algorithm.* In general, it means of updating networks our weights by back-propagating a gradient vector in which each element is defined as the derivative of an error measure for a parameter. Error signals are usually defined as the difference of the actual network outputs and the desired outputs[7].

2.2.3 Recurrent Neural Network

RNN is a type of neural networks which successfully deal with sequential data. The input is a sequence of $(x_1, x_2, ..., x_N)$ and the output of RNN is a sequence of hidden states $(h_1, h_2, ..., h_N)$.

$$h_t = f(W_x x_t + W_h h_{t-1} + b_n) \tag{2.13}$$

Hidden states represent a piece of specific information about the input sequence (strongly depends on our loss function) and we are entirely free to calculate. For NER we can use the whole sequence, for we typically use only last hidden state h_N and propagate it to some dense neural network.

Classical RNN fail to learn long dependencies in a sequence and tend to be biased towards their most recent inputs in the sequence[35]. So we will work with the improved version called **LSTM**. They were designed to combat this problem by adding a memory cell. The memory cell is much more suitable to extract long-range dependencies[36]. We will follow classic architecture suggested by Hochreiter and Schmidhuber in 1997:

$$\text{input gate} : i_t = f(W_i x_t + W_i h_{t-1} + b_i) \tag{2.14}$$

$$\text{forget gate} : f_t = f(W_f x_t + W_f h_{t-1} + b_f) \tag{2.15}$$

$$\text{output gate} : o_t = f(W_o x_t + W_o h_{t-1} + b_o) \tag{2.16}$$

$$\text{inner state} : u_t = \tanh\left(W_u x_t + W_u h_{t-1} + b_u\right) \tag{2.17}$$

$$\text{memory cell} : c_t = i_t \times u_t + f_t \times c_{t-1} \tag{2.18}$$

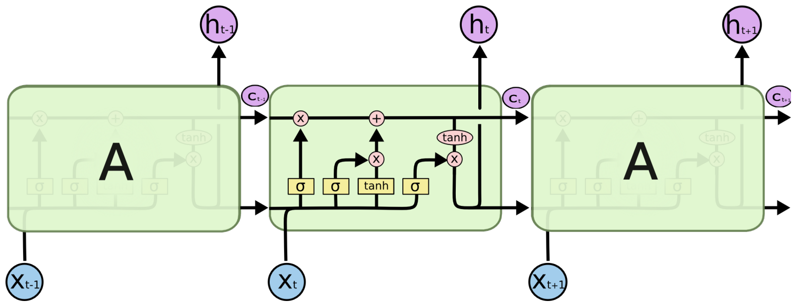


Figure 2.6: Long-short term memory[8]

$$\text{hidden state}: h_t = o_t \times \tanh\left(c_{t-1}\right) \tag{2.19}$$

where $\times$ is element-wise multiplication, x_t is d-dimensional input vector at time t and W are weight matrix which need to be learn. The learning is based on gradient descent. A common choice is Adaptive Moment Estimation (Adam) which combines the advantages of several previous algorithms[37].

A slightly more dramatic variation on the LSTM is the GRU[38]. It combines the forget and input gates into a single update gate. It also merges the memory cell state and hidden state and makes some other changes. The resulting model is simpler than standard LSTM models and has been growing increasingly popular. We can see basic architecture in Figure 2.6.

2.2.3.1 Bidirectional LSTM

To overcome the limitations of a regular RNN, Schuster[39] propose a bidirectional recurrent neural network that can be trained using all available input information in the past and future of a specific time frame.

In general, there are two independent LSTM networks in a forward and backward direction (as shown in Figure 2.7), which are trained independently (through back-propagation).

- Forward pass
- Backward pass
- Update weights

The output at each timestamp t is a concatenation of h_t and h'_t. Because the size of the network is doubled, the quantity of parameters is also doubled.

2.2.4 Convolutional Neural Network

The CNN were specifically designed to deal with the variability of two dimensional shapes[40]. There are two essential parts in almost every CNN:

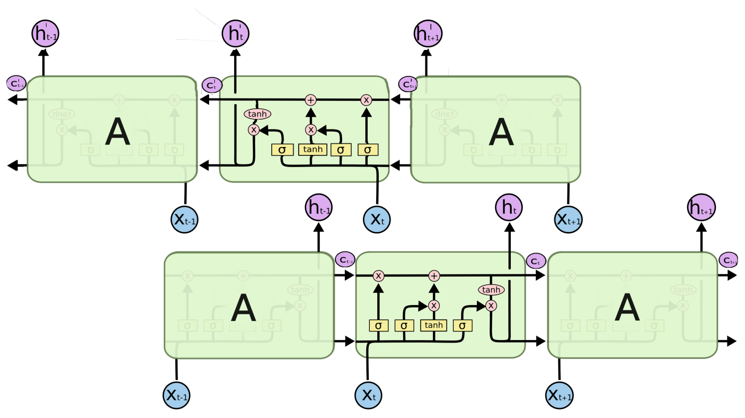
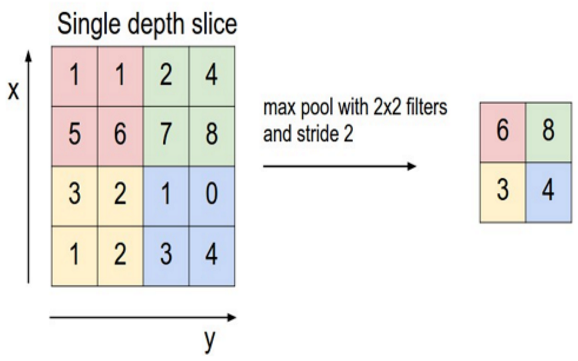


Figure 2.7: Bidirectional Long-short term memory, modified[8]



1	1	2	4
5	6	7	8
3	2	1	0
1	2	3	4

6	8
3	4

Figure 2.8: Max-pooling[9]

filter/kernel (we will use term filter) and **max-pooling layer**. We can imagine filter as matrix $W \in R^{w \times h}$ where w and h are chosen by us and says how big area we should monitor by this specific filter. The max-pooling layer is non-linear function which picks up the maximum in a certain area defined by us (usually 2×2 or 3×3). For max-pooling (shown on Figure 2.8 following formula $X^{\text{new}}_{0,0} = \max\left(X^{\text{old}}_{0,0}, X^{\text{old}}_{0,1}, X^{\text{old}}_{1,0}, X^{\text{old}}_{1,1}\right) = \max\left(1,2,3,2\right) = 3$) with a size of 2×2 on a 2D input of size $d \times n$ it would be like:

$$\begin{bmatrix} x_{11} & x_{12} & x_{13} & \dots & x_{1n} \\ x_{21} & x_{22} & x_{23} & \dots & x_{2n} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ x_{d1} & x_{d2} & x_{d3} & \dots & x_{dn} \end{bmatrix} = \begin{bmatrix} \max\left(x_{11}, x_{12}, x_{21}, x_{22}\right) & \dots \\ \max\left(x_{21}, x_{22}, x_{31}, x_{32}\right) & \dots \\ \vdots & \ddots \end{bmatrix}$$

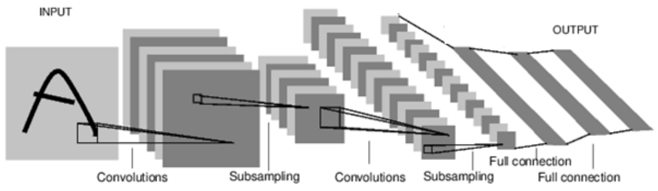


Figure 2.9: Convolutional Neural Networks[9]

so filter W of size 2×2 would work like this

$$x_{11}^{\text{new}} = \sum_{i=1}^{2}\sum_{j=1}^{2} w_{ijij}$$

$$\vdots$$

$$x_{d-1n-1}^{\text{new}} = \sum_{i=d-1}^{d}\sum_{j=n-1}^{n} w_{ijij}$$

so it would change the dimension of the original input. If the input was $R^{d\times n}$ then after applying the filter of size 2×2 it would be $R^{d-1\times n-1}$. We can fight with this by adding 0 to a border of the matrix (for all of the values which are not onto input matrix). Then it would be like this:

$$x_{11}^{\text{new}} = \sum_{i=0}^{1}\sum_{j=0}^{1} w_{ijij}$$

$$\vdots$$

$$x_{d-1n-1}^{\text{new}} = \sum_{i=d}^{d+1}\sum_{j=n}^{n+1} w_{ijij}$$

On Figure 2.9 is shown very basic system where you can see connection between CNN and acronym-next-pages.

2.2.4.1 Convolution neural networks over sequences

Despite little tuning of hyperparameters, even a simple CNN with one layer of convolution performs remarkably well in working with sequential data[10]. We can apply CNN over words or characters. Our embeddings of dimension d for a word of character length L or a sentence of word length L can be written in the matrix $M \in R^{d\times L}$ like:

$$M = \begin{bmatrix} x_1 & x_2 & \dots & x_L \end{bmatrix} \tag{2.20}$$

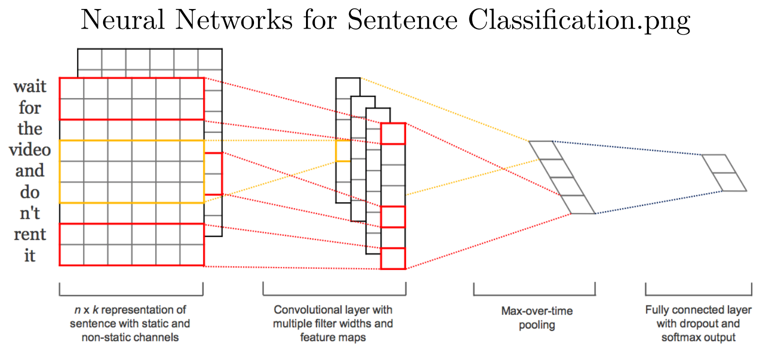

Neural Networks for Sentence Classification.png
wait
for
the
video
and
do
n't
rent
it
n x k representation of sentence with static and non-static channels
Convolutional layer with multiple filter widths and feature maps
Max-over-time pooling
Fully connected layer with dropout and softmax output

Figure 2.10: Convolutional Neural Networks for Sentence Classification[10]

where $x_n \in R^d$ or for the illustration we can look at Figure 2.10.

To learn patterns over the embeddings, a convolution operation with a filter $w \in R^{d \times s}$ is applied to a window of s embeddings (can be representations of characters or words). It results in a feature matrix C. Which can be also written that each feature $c_i \in C$ is extracted from a window of embeddings $x_{i:i+s1} \in R^{d \times s}$, as follows:

$$c_i = f(w \cdot x_{i:i+s1} + b) \tag{2.21}$$

where f is an activation function, e.g. sigmoid or tanh, and $v \in R$ is a bias.

We will combine RNN with CNN similar to [11], which showed promising approach. Their model is described on Figure 2.11 and Figure 2.12.

2.2.5 Attention model

The recurrent model with the attention (see Figure 2.13) can solve certain problems with long-term dependencies in pure recurrent networks[12].

It is strongly coupled with the recurrent networks and its hidden states h_i. We need to find a suitable combination for an output C:

$$C = \sum_{t=1}^{T} \alpha_t \cdot h_t \tag{2.22}$$

where T is number of hidden states $h_1, h_2, ..., h_T$ (for example taken from LSTM) and α_t is a softmax output of learned function a (can be a feed-forward acronym-next-pages):

$$e_t = a(h_t) \tag{2.23}$$

$$\alpha_t = \frac{\exp e_t}{\sum_{k=1}^{T} \exp e_k} \tag{2.24}$$

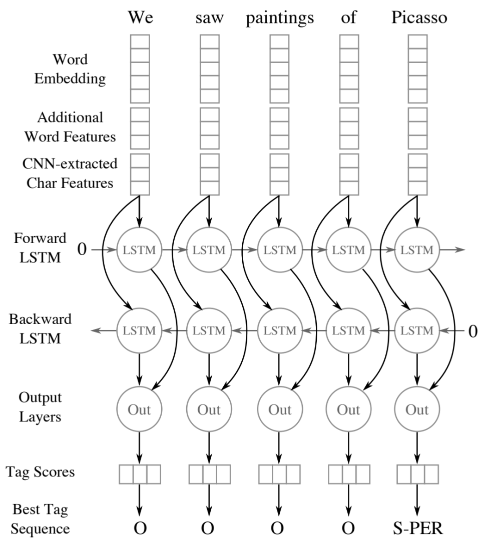


Figure 2.11: Named Entity Recognition with Bidirectional LSTM-CNNs[11] - Recurrent part

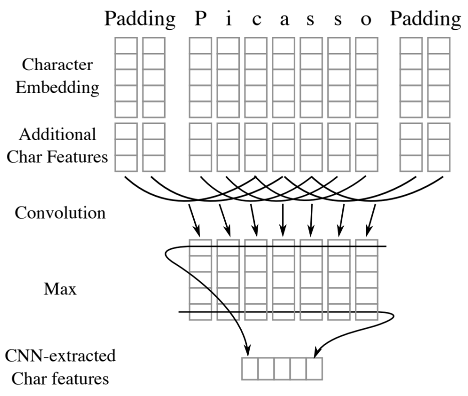


Figure 2.12: Named Entity Recognition with Bidirectional LSTM-CNNs[11] - Convolution part

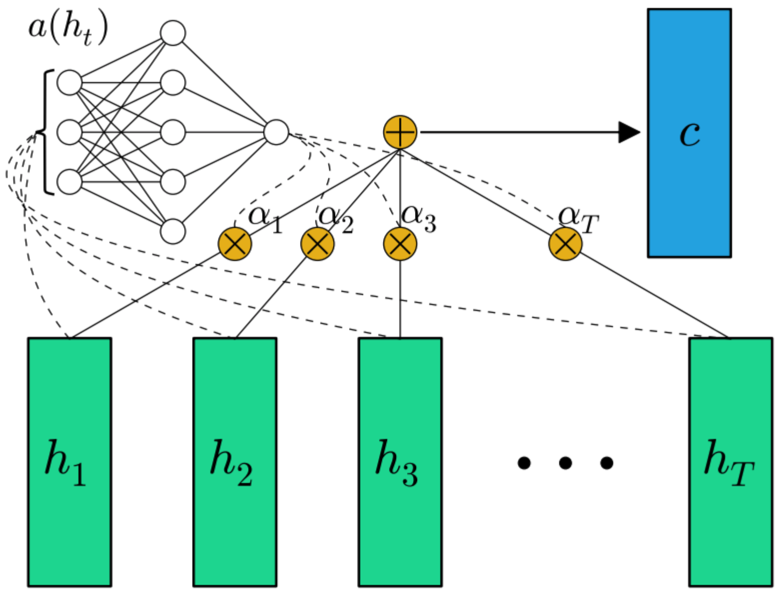


Figure 2.13: Networks with Attention[12]

There is also possibility to use pure word embeddings as a hidden states $h_1, h_2, ..., h_T$ to avoid RNN units.

2.2.6 Deep contextualized word representations

Currently, many teams all over the world are focusing on improving performance over many tasks with word embeddings. Combination of previous methods is one of the state-of-art models which is called ELMo (Embeddings for Language Models) [41]. They say that they reached new state-of-the-art results in SQuAD, SNLI, NER or even POS. Their approach works because the language model learns on a huge amount of monolingual data which are easy to obtain and then we fine-tuned our model on a specific task like NER or IR. They also use character embeddings as an input which is dealing with out-of-vocabulary words by default and then CNN over this char embeddings to create a vector representation of the word (for more detail see Figure 2.14). After they obtain a word embedding they use it as an input into bidirectional LSTM which has to return the probability of next/previous (because of bi-direction) words. After this training phase we have pre-trained model which is used to get a word embedding based on the context of the whole sentence. For example a word "bank" can be used in multiple sentences with different meaning, so the embedding will differ too. The final embedding is calculated like a linear combination of outputs of inner RNN, where different RNN catch a different information (semantic or syntactic). We can see it on the Figure 2.15.

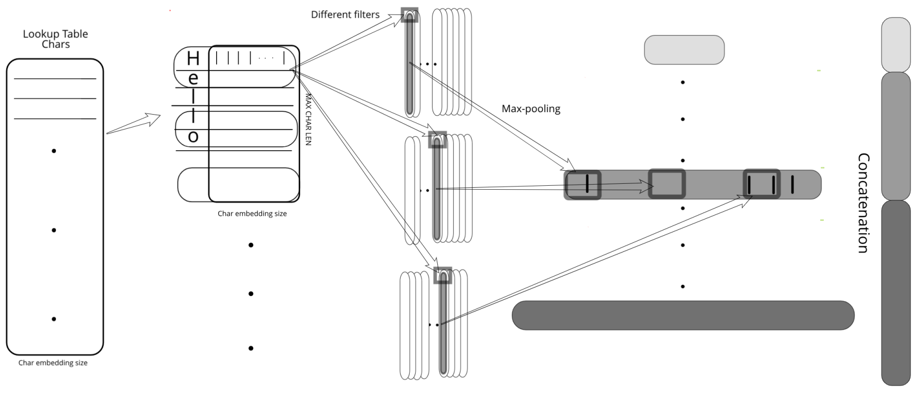


Figure 2.14: Character embeddings in ELMo. Each word is a matrix of embedding. There are several CNN with different size of filters which output is max-pooled. The final word vector is concatenation of outputs from max-pooled layer.

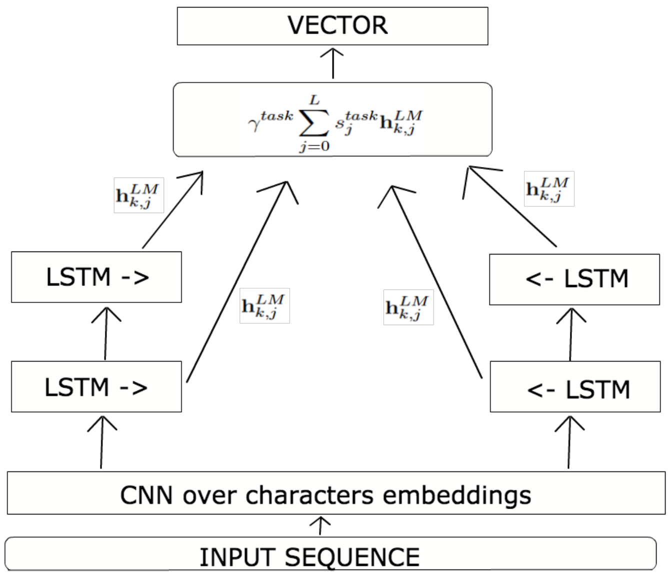


Figure 2.15: ELMo architecture for training and also for obtaining vector/embedding. $h_{k,j}^{LM}$ is the output of LSTM for k-th token and j-th layer of LSTM.

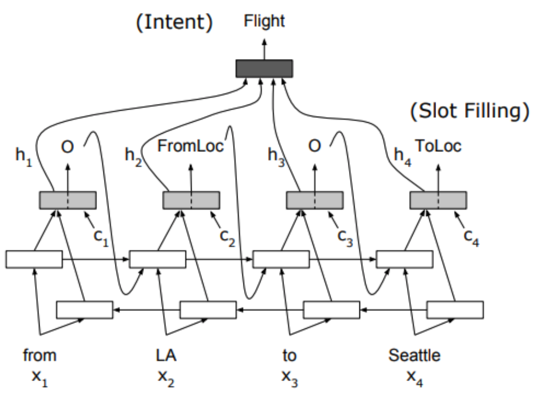


Figure 2.16: Attention-based RNN model for joint intent recognition and sequence labelling[13]

2.2.7 Joint intent recognition and sequence labeling

Few works[13][24] are focusing on a combination of a sequence labelling (for example slot filling[4] which is similar to NER) and classification. They are using different approaches like Encoder-Decoder RNN[24] or a combination of LSTM with CNN[13]. They are suggesting the model on Figure 2.16 as one of the most promising. It combines advantages of recurrent neural networks and attention model.

A bidirectional RNN reads the source sequence in both forward and backward directions. Sequence label dependencies are modeled in the forward RNN. The hidden state h_i at each step is a concatenation of the forward state and backward state. This hidden state h_i is then combined with the context vector c_i produced by the attention model to get the intent distribution. Then we select the most feasible label.

2.2.8 Overfitting

The overfitting is the common obstacle of large models with thousands of parameters with combination with a low amount of data. There are two types of overfitting[42]:

- Big model for a few examples — a model that is more flexible than it needs to be and sometimes includes irrelevant components or predictors (quadratic model for only two examples)

[4]For example, the task is to find a place in a text where should be information about a flight, destination, time, . . .

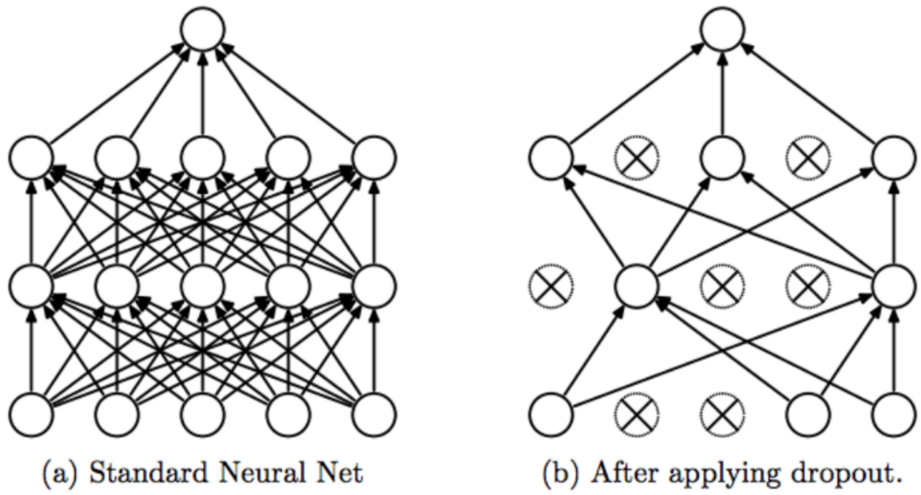


Figure 2.17: Dropout in acronym-next-pages[14]

- Wrong model for our data — a model, includes irrelevant components. One example can be imagined like using a polynomial of excessive degree (quadratic model on a linear data)

We should avoid these combinations due to several difficulties which they can cause:

- Wasting computing resources
- Make a harder prediction for unseen data
- Worse interpretability which leads to worse portability of a model

Because overfitting problem is well-know[14][18][23] there are few attempts to fight against it (overfitting can happen in all sorts of model but we will focus on acronym-next-pages):

- Adding Dropout to acronym-next-pages — As shown in Figure 2.17 we will use a probability distribution to choose which neurons should be dropped and not used in a computation. A parameter of dropout is the ratio (means how many neurons we would like to hide)
- Adding Regularization to acronym-next-pages — Adding regularization term to loss function which will punish models with huger parameters
- Early stopping — Stop training of a model when the loss on the validation set is getting higher

Chapter 3

Experiments

The previous section talks about data sources we will be used. In the practical part, we will use several models and compare them to each other based on that data.

3.1 Data

The data is a critical part of every model. We will focus mainly on text data so there are several important moves which we should consider:

- Tokenization
- Stemming/Lemmatization
- Spelling errors

We will work with word embeddings. First choice would be fastText[5] which prove superior over other types of embeddings (like word2vec[33] or Glove[43]) in several tasks [44]. Because of that we do not have to deal with **stemming/lemmatization** or *miss-click detection* because of the language modeling technique used for creating word embedding. It deals with that obstacle out of the box. However, what we should consider is proper tokenization.

We will also include ELMo embeddings (described in Section 2.2.6) into our experiments because of their awareness about the context of a whole sentence and proven strong result over many NLP tasks[41]. There are several pretrained models[6], but we will use a version suitable for library Keras[7]. The

[5]Pretrained embeddings available at https://fasttext.cc/docs/en/crawl-vectors.html
[6]https://allennlp.org/elmo
[7]Modification of https://github.com/strongio/keras-elmo/blob/master/Elmo%20Keras.ipynb

Name	# examples	# unique words	Average length	# intents
Train Set	*5452*	*9448*	*10.2*	*50*
Test Set	*500*	*1103*	*7.5*	*42*

Table 3.1: Data for Question Classification

TensorFlow documentation is available[8] and provide good source of information about the interface. We can train our **task specific** parameters (which is used for multiplying the outputs of inner LSTM). The output vector is 1024 big and is created as the weighted sum of the inner layers in both directions.

Both embeddings are publicly available as a pre-trained models.

3.1.1 Datasets for intent recognition

There were several conditions which need to be met. Firstly the dataset has to be publicly available, then there should also be results based on the dataset to allow us to compare our results with them and last, but not least it should be the task which is closely related to the one we want to do in Alquist. Based on our conditions, firstly, we chose the datasets presented in [45], where is a comparison of publicly available tool (like RASA[9], API.ai[10], IBM Watson[11] . . .) for intent recognition. Unfortunately, there is no essential information about models which are used by these tools and another drawback is a very tiny number of examples in a train and test set (three hundreds). The amount of data is not enough for training acronym-next-pages so we do not want to include it in our experiments because of high probability of overfitting the data. Another data source was chosen based on [46],[10] and [45]. The **Question Classification dataset** created by Xin Li and Dan Roth[47] became standard for broad spectrum of NLP tasks. The data are publicly available[12] and consists of almost 5500 training questions and balanced 500 testing question. The question taxonomy[13] consists 6 broader categories (such as question about **ABBREVIATION**, **ENTITY**, **DESCRIPTION**, **HUMAN**, **LOCATION** and **NUMERIC**) and 50 subcategories. More information can be seen on Table 3.1.

[8]https://tfhub.dev/google/elmo/2
[9]https://rasa.com/
[10]https://dialogflow.com/
[11]https://www.ibm.com/watson/
[12]http://cogcomp.org/Data/QA/QC/
[13]http://cogcomp.org/Data/QA/QC/definition.html

	# Articles	**# Sentences**	**# Tokens**
Training Set	*946*	*14,987*	*203,621*
Developments Set	*216*	*3,466*	*51,362*
Test Set	*231*	*3,684*	*46,435*

Table 3.2: CoNLL 2003

	# LOC	**# MISC**	**# ORG**	**# PER**
Training Set	*7140*	*3438*	*6321*	*6600*
Developments Set	*1837*	*922*	*1341*	*1842*
Test Set	*1668*	*702*	*1661*	*1617*

Table 3.3: CoNLL 2003 entity

Word	**POS tag**	**Chunk tag**	**Named entity tag**

Table 3.4: CoNLL 2003 format

3.1.2 Datasets for entity recognition

Datasets for entity recognition was chosen based on data which are most often used in previous works [11]. Dataset is called CoNLL 2003[48] and is publicly available[14]. Dataset was introduced at CoNLL-2003 Shared task for language-independent named entity recognition[26] - it is the reason why there are two languages included - English and German. Only English was used due to copyright license. There is only one change to original dataset, and it is changing spaces in a line to tabs. The change was suitable for reusability code for other datasets.

3.1.3 Automatically generated dataset

The automatically generated dataset was created from dialogs graph used in a chatbot system Alquist. The Figure 3.1 shows a graph which is used for a dialog generation. Each dialog is located in a folder with a name describing his intent. We used name of the folder for **intents**. The folder structure of dialogs is mandatory. Each dialog graph contains nodes, where we can define text template. It determines sentences which chatbot system or a user could say. The dialog graph also include entity mark which is placeholder for substituting words. Base dialog graphs was taken from a chatbot system Alquist and then manually updated to required format or deleted if there was no mentions about entities or a place for adding entity mark. The output looks like:

[14]https://www.clips.uantwerpen.be/conll2003/ner/

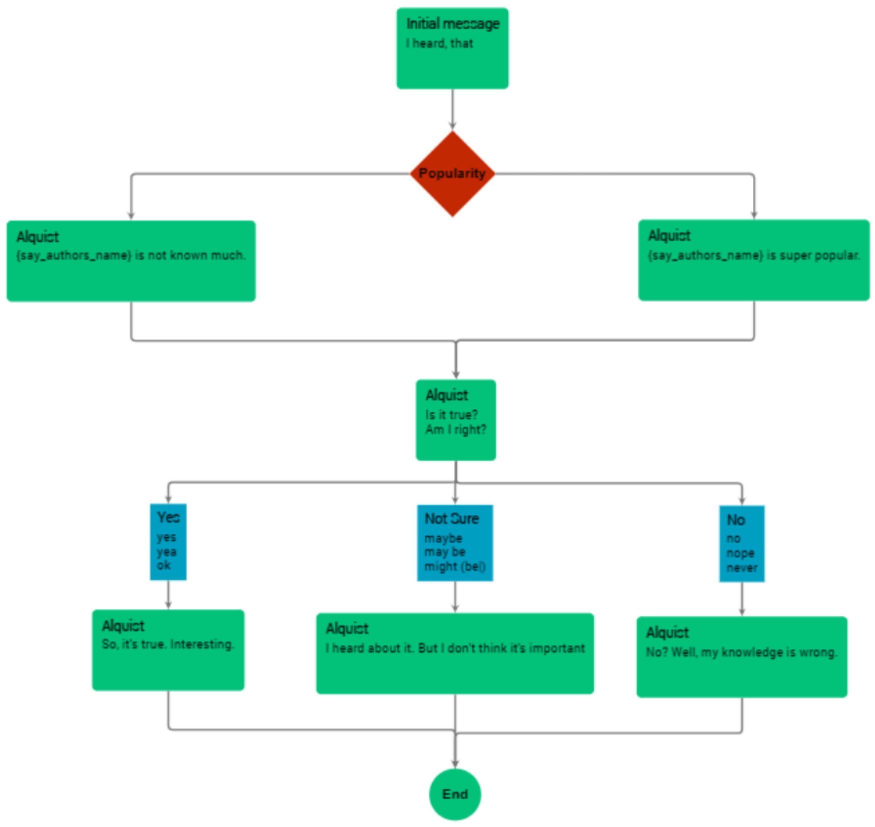

Initial message
I heard, that
Popularity
Alquist
{say_authors_name} is not known much.
Alquist
{say_authors_name} is super popular.
Alquist
Is it true?
Am I right?
Yes
yes
yea
ok
Not Sure
maybe
may be
might (be|)
No
no
nope
never
Alquist
So, it's true. Interesting.
Alquist
I heard about it. But I don't think it's important
Alquist
No? Well, my knowledge is wrong.
End

Figure 3.1: Graph used for generating dialog data [15]

- Let's talk about **#Entity#**!
- Do you like ****#Pseudo_entity#****?

where everything inside hashtags is substituted with entity/pseudo-entity (for example it would be **London** and **rock music** in the illustration above). The names for entities were taken from knowledge graph based on Microsoft Concept Graph[15] which was updated by Alquist's team with new data. For generating the dataset was used 51 models with minimum expansion size set to 500 (every template should generate at least 500 passes through graph).

The statistics for a generated dataset are shown in Table 3.5.

The generated dataset is unbalanced because of **89%** of non-entities. The reason is that there are low occurrence of entities in a natural text. But we will use techniques against overfitting to fight problem of unbalanced dataset.

[15]Available at https://concept.research.microsoft.com/Home/Introduction

Name	# Sentences	# Examples	Average length of words	Unique values
Training Set	*63629*	*X*	*8.42*	*4025 words*
Training Set NER	*X*	*535967*	*X*	*30*
Training Set INTENT	*X*	*63629*	*X*	*61*
Test Set	*15880*	*X*	*8.40*	*2951 words*
Test Set NER	*X*	*133542*	*X*	*30*
Test Set INTENT	*X*	*15880*	*X*	*61*

Table 3.5: Generated data

Name	# Sentences	# Examples	Average length of words	Unique values
Training Set	*4978*	*X*	*4.45*	*943*
Training Set NER	*X*	*4978*	*X*	*129*
Training Set INTENT	*X*	*4978*	*X*	*26*
Test Set	*893*	*X*	*4.39*	*943*
Test Set NER	*X*	*893*	*X*	*129*
Test Set INTENT	*X*	*893*	*X*	*26*

Table 3.6: ATIS dataset

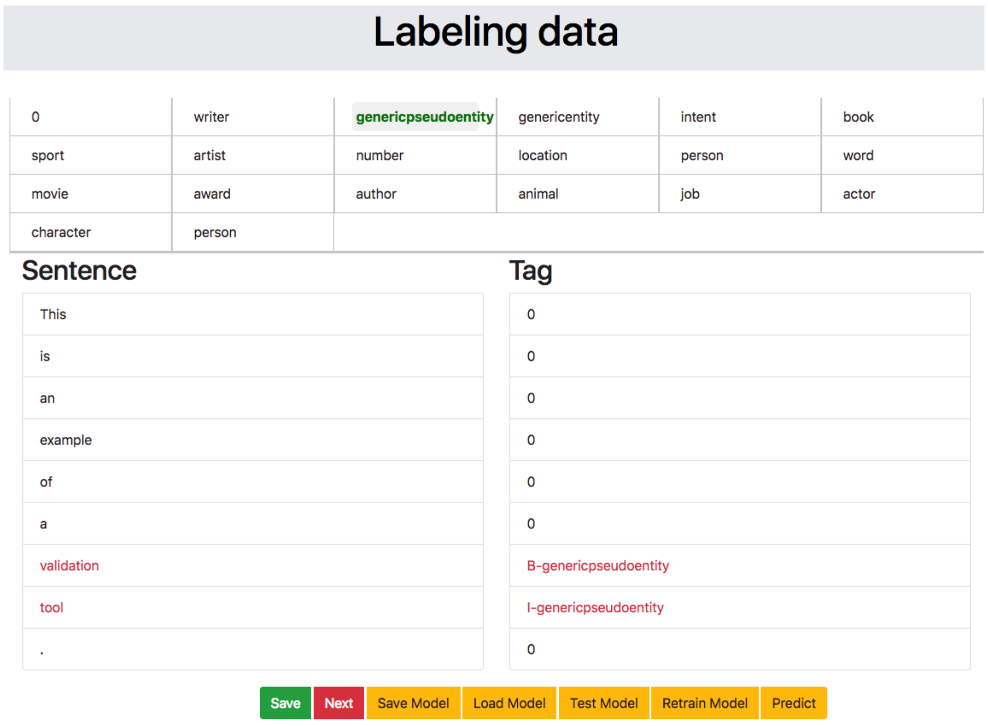
3.1.4 ATIS dataset

The dataset was created by team in Microsoft and is publicly available[16]. The dataset (described in Table 3.6) contains spoken utterances classified into one of 26 intents. Each token in a query utterance is aligned with IOB labels. Primarly, the dataset is used for intent recognition and slot filling, but the slot filling and entity recognition are interchangeable.

3.1.5 Manual labeling tool

Because we want to have as much valid data as possible, we created a labeling tool (which is shown in Figure 3.2) for the purpose of this theses. The labeling tool uses web interface which loads data in a format specified in config file. After loading data, they are shown to the user, and he has a possibility to

[16]github.com/Microsoft/CNTK/tree/master/Examples/LanguageUnderstanding/ATIS


Labeling data

0	writer	genericpseudoentity	genericentity	intent	book
sport	artist	number	location	person	word
movie	award	author	animal	job	actor
character	person				

Sentence	Tag
This	0
is	0
an	0
example	0
of	0
a	0
validation	B-genericpseudoentity
tool	I-genericpseudoentity
.	0

Figure 3.2: Tool for manual labelling

put right labels to right words. The tool was used for manual correction of generated dataset.

3.2 Models

The models are trained and evaluated on data described above. The number of parameters is taken from the summary output provided by library Keras[17], which was chosen mainly for a possibility of fast creating and testing models. The time consumption is based on a tool directly provided by Jupyter Notebook[18] and RAM consumption is measured by a tool ipython_memory_usage[19], which monitor usage of RAM with system memory profiler. Each model was trained and measured in a fresh environment on Amazon Web Service EC2 G3[20] (details in Table 3.7). The server GPU is based on graphics card Tesla M60 which can be monitored by a tool *nvidia-smi*[21]. The measurements are

[17]https://keras.io

[18]https://ipython.readthedocs.io/en/stable/interactive/magics.htmlmagic-time

[19]https://github.com/ianozsvald/ipython_memory_usage

[20]https://aws.amazon.com/ec2/instance-types/g3/

[21]The NVIDIA System Management Interface, more information available at https://developer.nvidia.com/nvidia-system-management-interface

Name	**GPUs**	**vCPU**	**RAM (GiB)**	**GPU Memory (GiB)**
g3.4xlarge	*1*	*16*	*122*	*8*

Table 3.7: Amazon Web Service EC2

based on average from 10 different runs and confidential interval are calculated with 95% confidence assumes that the samples are drawn from a Gaussian distribution.

3.2.1 Entity recognition

In this part, we will be working with sequence input and sequence output. We need that the model will predict a label for each word. In an experiments below we were using fastText and ELMo embeddings. CRF was one of the most effective approaches for NER, it achieved excellent performances on several tasks, including NER[49]. The disadvantage is a need for defining quality features functions (for example orthographic features). The orthographic convertor is a basic function with the set of rules rewriting the text to series of marks (for example chars in uppercase to **C**, the lowercase to **c** and numbers to **n**). However, we can substitute the manual creation of feature function by connecting an output of acronym-next-pages as an input to CRF.

The approach was following:

- (1) Use forward and backward RNN + fastText
- (2) Use forward and backward RNN + fastText + CRF
- (3) Use forward and backward RNN + ELMo
- (4) Use forward and backward RNN + ELMo + CRF
- (5) Combine CNN over word-based and char-based input + fastText
- (6) Combine CNN over word-based and char-based input + fastText + CRF
- (7) Combine CNN over word-based and char-based input + ELMo
- (8) Combine CNN over word-based and char-based input + ELMo + CRF

The main aim was to find best algorithm and compare embeddings. The model "Combine CNN over word-based and char-based input" was based on Section "Convolution neural networks over sequences". The model "Use forward and backward RNN" was based on Section "Bidirectional LSTM".

From the result in Table 3.8 we can see that using CRF is better in a manner of the result but also adding more power requirements for training. The next conclusion is that ELMo is better in entity recognition task than fastText but add next power intensive layer and broadly increase the number of parameters because the size of the output vector is 1024 (instead of 300) in available pre-trained model.

Model	Train Validation accuracy	F1 score	# Parameters	Avg. GPU memory/power usage	Avg. Time	Avg. RAM usage
(1)	0.9843 ± 0.0009 0.9699 ± 0.0005	0.9541 ± 0.0005	**441,866**	39/6%	**7min 1s**	181.65 MB
(2)	0.9867 ± 0.0008 0.9715 ± 0.0007	0.9561 ± 0.0006	473,602	36/5%	8min 21s	192.98 MB
(3)	0.9987 ± 0.0002 0.9796 ± 0.0004	0.9644 ± 0.0006	1,183,246	90/49%	8min 36s	1700.14 MB
(4)	**0.9991** ± 0.0003 0.9808 ± 0.0004	0.9659 ± 0.0004	1,183,366	88/51%	11min 20s	2273.50 MB
(5)	0.9913 ± 0.0006 0.9857 ± 0.0006	0.9700 ± 0.0002	481,544	45/9%	8min 48s	**121.20** MB
(6)	0.9930 ± 0.0001 0.9856 ± 0.0004	0.9698 ± 0.0010	481,664	39/10%	16min 48s	136.28 MB
(7)	0.9965 ± 0.0004 0.9851 ± 0.0006	0.9702 ± 0.0006	1,222,924	86/47%	10min 57s	2037.30 MB
(8)	0.9974 ± 0.0009 **0.9857** ± 0.0008	**0.9704** ± 0.0008	1,223,044	80/40%	14min 53s	2353.89 MB

Table 3.8: Results of entity recognition

3.2.2 intent recognition

intent recognition (in opposite to NER) is classification task. We output one label for a given sequence of text. In an experiments below I was using fastText and ELMo embeddings. The approach was folowing:

- (1) Use forward RNN + fastText
- (2) Combine forward and backward RNN + fastText
- (3) Combine forward and backward RNN + ELMo
- (4) Combine word-based and char-based CNN + fastText
- (5) Use Attention layer + fastText
- (6) Use Attention layer + ELMo
- (7) Combine Attention layer with RNN + fastText
- (8) Combine Attention layer with RNN + ELMo

The principal purpose was to find the best algorithm and compare embeddings. Results in a given task can be seen in a Table 3.9. All models use full memory (8 GiB) of GPU, but the difference was in memory/power usage. The values in a table are from measurements with random shuffle where the training set is 4906 big and validation set is 546. The training was terminated if the validation loss was worse for 3 consecutive runs.

As results we can say that ELMo embeddings are superior to fastText embeddings for the intent recognition task. On the other side, there is a drawback in power requirements and worse integration to Keras (which I hope get better soon). The Figure 3.5 show model loss for most promising architectures with the best curve for (6).

Model	**Train Validation accuracy**	**F1 score**	**# Parameters**	**Avg. GPU memory/power usage**	**Avg. Time**	**Avg. RAM usage**
(1)	0.902 ± 0.019 0.773 ± 0.013	0.774 ± 0.009	242,610	38/5%	42s	**32.64** MB
(2)	0.913 ± 0.011 0.771 ± 0.007	0.773 ± 0.006	478,642	36/6%	58s	56.56 MB
(3)	0.942 ± 0.009 0.787 ± 0.007	0.784 ± 0.006	1,193,526	87/48%	2min 25s	2124.71 MB
(4)	0.934 ± 0.012 0.785 ± 0.008	0.779 ± 0.005	488,906	66/17%	**21.2**s	54.50 MB
(5)	0.934 ± 0.009 0.781 ± 0.006	0.777 ± 0.004	**118,972**	40/7%	1min 2s	68.32 MB
(6)	0.962 ± 0.005 0.794 ± 0.003	0.819 ± 0.005	137,783	92/56%	2min 9s	2029.03 MB
(7)	0.931 ± 0.008 0.778 ± 0.005	0.775 ± 0.004	454,716	39/7%	53s	89.16 MB
(8)	**0.998** ± 0.001 **0.818** ± 0.008	**0.819** ± 0.004	1,220,151	90/60%	1min 53s	2153.08 MB

Table 3.9: Results of intent recognition

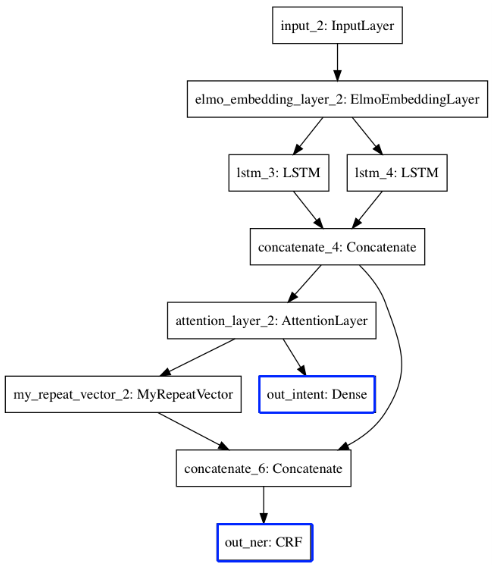

input_2: InputLayer
elmo_embedding_layer_2: ElmoEmbeddingLayer
lstm_3: LSTM
lstm_4: LSTM
concatenate_4: Concatenate
attention_layer_2: AttentionLayer
my_repeat_vector_2: MyRepeatVector
out_intent: Dense
concatenate_6: Concatenate
out_ner: CRF

Figure 3.3: Model which combine Attention layer with RNN

3.2.3 Combined intent and entity recognition

In this part, we will work only with the ELMo embedding because of their superior results over fastText (see Table 3.8 and Table 3.9).

There will be four models. First two are just the best model on each task trained on dialogs data.

Then we will work with a model (number 3) based on "Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling."[13] Because we need to reproduce their example in a real application and there is not a sample implementation of it, we will re-implement their model, which is a combination of RNN and Attention layer. From our previous results on NER task, we can also conclude that CRF can be a reasonable extension of NERs part. The architecture is shown in Figure 3.3.

Another model (number 4) will be based firmly on previous results. The best entity recognition approach was "Combine CNN over word-based and char-based input + ELMo + CRF" and best IR approach was "Combine Attention layer with RNN + ELMo". Our approach will be to connect these two models together. Unfortunately, it will inevitably lead to making some assumption about the connected model, so to optimize it, we will propose another model which will use only the best features from each of them. The intent part will be very similar to model 3, with using Attention layer which will make a weighted selection over RNN output. The most significant change in the architecture will be in RNN part where we will use char-embeddings and also the case information for the input. We will use CRF because of superior results over plain RNN[25]. The architecture is shown in Figure 3.4.

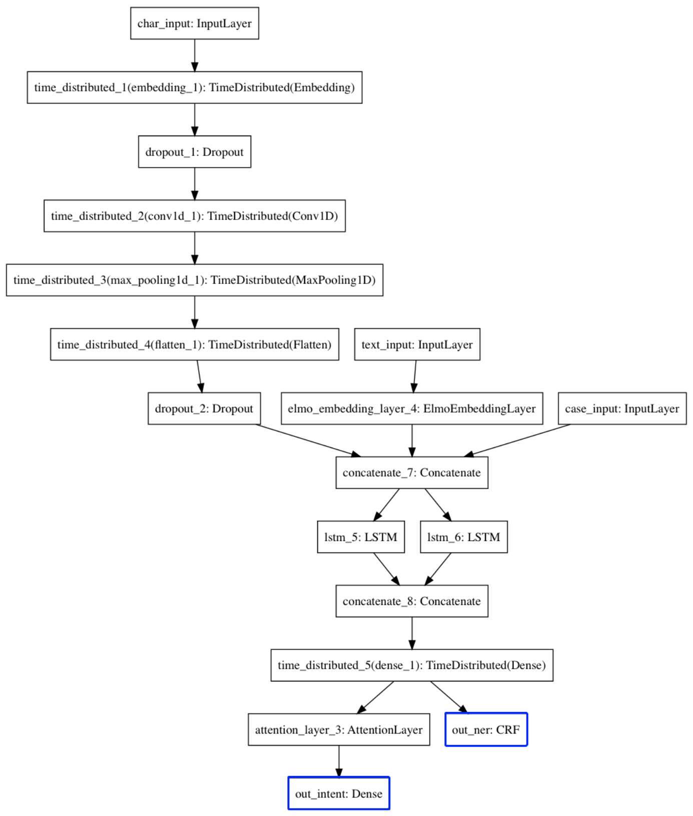

char_input: InputLayer
time_distributed_1(embedding_1): TimeDistributed(Embedding)
dropout_1: Dropout
time_distributed_2(conv1d_1): TimeDistributed(Conv1D)
time_distributed_3(max_pooling1d_1): TimeDistributed(MaxPooling1D)
time_distributed_4(flatten_1): TimeDistributed(Flatten)
text_input: InputLayer
dropout_2: Dropout
elmo_embedding_layer_4: ElmoEmbeddingLayer
case_input: InputLayer
concatenate_7: Concatenate
lstm_5: LSTM
lstm_6: LSTM
concatenate_8: Concatenate
time_distributed_5(dense_1): TimeDistributed(Dense)
attention_layer_3: AttentionLayer
out_ner: CRF
out_intent: Dense

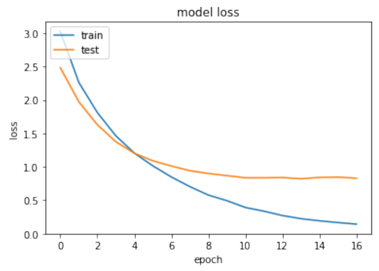
Figure 3.4: Model which Combine best model for entity recognition and IR

The testing process will be following:

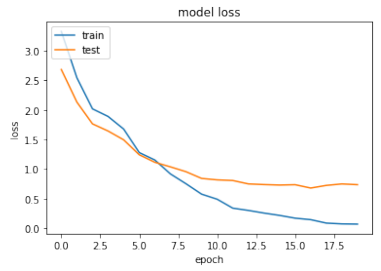
- (1) Measure best model for IR on generated data
- (2) Measure best model for entity recognition on generated data
- (3) Combine best model for entity recognition and IR
- (4) Combine Attention layer with RNN + ELMo - similar to [13]

Model	**Train accuracy (NER) (INTENT)**	**Validation accuracy (NER) (INTENT)**	**F1 score (NER) (INTENT)**	**# Params**	**Avg. GPU memory power usage**	**Avg. Time**	**Avg. RAM usage**
(1)	X 0.83498 ± 0.00248	X 0.80025 ± 0.00128	X 0.80008 ± 0.00130	1,221,828	**74/53%**	**21min 23s**	**1093.22 MB**
(2)	**0.99939** ± 0.00003 X	**0.99869** ± 0.00018 X	**0.99901** ± 0.00006 X	1,229,826	89/59%	47min 3s	1606.13 MB
(3)	0.99888 ± 0.00007 0.83531 ± 0.00251	0.99703 ± 0.00016 0.80402 ± 0.00197	0.99805 ± 0.00016 0.80801 ± 0.00265	1,266,882	87/48%	28min 5s	1124.71 MB
(4)	0.99938 ± 0.00005 **0.83893** ± 0.00016	0.99743 ± 0.00016 **0.80453** ± 0.00106	0.99814 ± 0.00006 **0.80877** ± 0.00078	**1,216,374**	82/54%	24min 49s	1311.02 MB

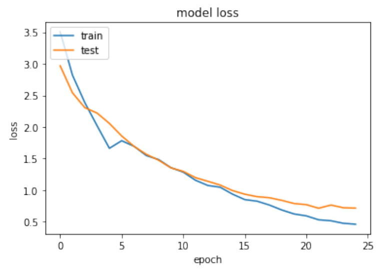
Table 3.10: Results of combined models



(a) Model loss for (1)



(b) Model loss for (2)



(c) Model loss for (4 - NER)

Figure 3.5: Model loss for combined models

From the results in Table 3.10 we can see that combination of entity recognition and IR is convenient. The results confirm common opinion [13][24][25] to combine it. The results in entity recognition are worse by a relative decreasement 0.008% in F1 score but there is 0.9% increasement in F1 score for IR. We should also take into account that the combined model is faster in training, would be easier to maintain and requires only the half size of RAM and GPU memory. The model also supports today's trend to make one general model for several connected tasks.

To support our statement over joint model we evaluated our algorithm even on ATIS dataset. We focused only on classic metrics like accuracy and F1 score. The result can be seen in Table 3.11.

```
$ curl -i -H "Content-Type: application/json" -X POST -d '{"text
    ":["Lets", "listen", "to", "rock", "music", "from", "bob", "
    dylan", "."]}' http://localhost:5000/predict
{
  "predictions_intent": "#Artist_Favorite#",
  "predictions_ner": [
    { "ner_label": "O", "word": "Lets" },
    { "ner_label": "O", "word": "listen"},
    { "ner_label": "O", "word": "to"},
    { "ner_label": "B-genericpseudoentity", "word": "rock"},
    { "ner_label": "O", "word": "music"},
    { "ner_label": "O", "word": "from"},
    { "ner_label": "B-genericpseudoentity", "word": "bob" },
    { "ner_label": "I-genericpseudoentity", "word": "dylan"},
    { "ner_label": "O", "word": "." } ],
  "success": true,
  "text": ["Lets", "listen", "to", "rock", "music", "from", "bob",
     "dylan", "."]
}
```

Listing 3.1: Example of request on model REST API

Because of simpler input (which take as an input only sequence of words where the only necessary pre-processing step is tokenization) and also more simpler model with less parameters, less memory requirements and less training time, we implemented model number 4 (Combination of Attention layer with RNN). The implementation was based on a REST API[22] architecture and JSON[23] output, where a prediction model is loaded when a server starts and communication is provided with standard POST[24] request. Then, if we start it on the server, it can be used in different applications and especially in Alquist. The example input and output from a server can be seen on Listing 3.1.

[22]Representational State Transfer Application Programming Interface
[23]JavaScript Object Notation
[24]Server accept data and return response

Metric	Alg. (1)	Alg. (2)	Alg. (3)	Alg. (4)	CNN TriCRF[50]	RecNN+Viterbi[51]
Train acc. NER	X	**0.99965** ± 0.00020	0.99882 ± 0.00156	0.99939 ± 0.00095	X	X
Validation acc. NER	X	**0.99359** ± 0.00041	0.98945 ± 0.00101	0.98981 ± 0.00069	X	X
Train acc. IR	0.99955 ± 0.00029	X	0.99953 ± 0.00026	**0.99970** ± 0.00007	X	X
Validation acc. IR	0.96204 ± 0.00125	X	0.96385 ± 0.00067	**0.96767** ± 0.00125	X	X
Test F1 NER	X	0.95486 ± 0.00040	**0.97718** ± 0.00159	0.94887 ± 0.00118	93.96	95.42
Test F1 IR	**0.95479** ± 0.00101	X	0.94754 ± 0.00123	0.94148 ± 0.00131	95.40	94.09

Table 3.11: Results on ATIS dataset

Conclusion

This master's thesis is focusing on NER and intent recognition as described in the initial chapters. We were exploring machine learning algorithms on a newly created dataset. For generating the dataset, we developed an original algorithm. We described the generation algorithm in the section "Automatically generated dataset". The main trick lies in combining the automatic generation process and manual labeling. The generated dataset is suitable for chatbot system Alquist. The main result is a comparison of different algorithms regarding accuracy, speed and computational requirements.

The study proceeded in two steps. First, we evaluated the NER and intent recognition separately. Then, we designed new algorithms combining the NER and intent. The main result of the master's thesis is proving that the joint models are, in general, preserving the NER accuracy and decreasing the intent recognition error rate. In our experiment, see Table 3.10 and Table 3.11, we compare results of our joint model with published results in the scientific papers. The joint model is proven to be more efficient and requires less computational resources over separated models for each task.

The master's thesis is provided with the trained model described in the last chapter. The proposed algorithms were successfully integrated into the chatbot system Alquist.

Future research, based on this master's thesis, can be focused on fine-tuning of the joint model, using gradually developing information in dialogs or using other types of embeddings (like BERT[52]).

Bibliography

[1] Devaney, E. The 2018 State of Chatbots Report: How Chatbots Are Reshaping Online Experiences. , no. 23, Jan. 2018, [Online; accessed 30-December-2018]. Available from: `https://www.drift.com/blog/chatbots-report/`

[2] khatun, A. Let's know Supervised and Unsupervised in an easy way. July 2018, [Online; accessed 30-August-2018]. Available from: `https://chatbotsmagazine.com/lets-know-supervised-and-unsupervised-in-an-easy-way-9168363e06ab`

[3] Walber. Precision and recall. Nov 2014, [Online; accessed 28-December-2018]. Available from: `https://commons.wikimedia.org/wiki/File:Precisionrecall.svg`

[4] Elenktik. Approximation of Markov chain. 2015, [Online; accessed 22-December-2018]. Available from: `https://commons.wikimedia.org/wiki/File:Mvchain_approx_C2.png`

[5] Cloud, A. HMM, MEMM, and CRF: A Comparative Analysis of Statistical Modeling Methods. May 2018, [Online; accessed 22-December-2018]. Available from: `https://medium.com/@Alibaba_Cloud/hmm-memm-and-crf-a-comparative-analysis-of-statistical-modeling-methods-49fc32a73586`

[6] Mikolov, T.; Chen, K.; et al. Efficient Estimation of Word Representations in Vector Space. *CoRR*, volume abs/1301.3781, 2013, [Online; accessed 29-December-2018], 1301.3781. Available from: `http://arxiv.org/abs/1301.3781`

[7] Sazli, M. A brief review of feed-forward neural networks. 2006: pp. 11–17, doi:10.1501/0003168, [Online; accessed 28-December-2018].

[8] Christopher Olah. Understanding LSTM Networks. 2015, [Online; accessed December 19, 2018]. Available from: `http://colah.github.io/posts/2015-08-Understanding-LSTMs/`

[9] Adit Deshpande. A Beginner's Guide To Understanding Convolutional Neural Networks. July 2016, [Online; accessed December 19, 2018]. Available from: `https://adeshpande3.github.io/A-Beginner%27s-Guide-To-Understanding-Convolutional-Neural-Networks/`

[10] Kim, Y. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Association for Computational Linguistics, 2014, pp. 1746–1751, doi:10.3115/v1/D14-1181. Available from: `http://aclweb.org/anthology/D14-1181`

[11] Chiu, J. P. C.; Nichols, E. Named Entity Recognition with Bidirectional LSTM-CNNs. *CoRR*, volume abs/1511.08308, 2015, 1511.08308. Available from: `http://arxiv.org/abs/1511.08308`

[12] Raffel, C.; Ellis, D. P. W. Feed-Forward Networks with Attention Can Solve Some Long-Term Memory Problems. 2015, [Online; accessed 25-December-2018], 1512.08756.

[13] Ma, M.; Zhao, K.; et al. Jointly Trained Sequential Labeling and Classification by Sparse Attention Neural Networks. *CoRR*, volume abs/1709.10191, 2017, [Online; accessed 25-December-2018], 1709.10191. Available from: `http://arxiv.org/abs/1709.10191`

[14] Srivastava, N.; Hinton, G.; et al. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, volume 15, 2014: pp. 1929–1958, [Online; accessed 26-December-2018]. Available from: `http://jmlr.org/papers/v15/srivastava14a.html`

[15] Pichl, J.; Marek, P.; et al. Alquist: The Alexa Prize Socialbot Based on Sub-Dialogue Models. 2018, [Online; accessed 23-December-2018]. Available from: `https://developer.amazon.com/alexaprize/2018/proceedings`

[16] David Nadeau, S. S. A survey of named entity recognition and classification [online]. 2007, [Online; accessed 4-November-2018]. Available from: `https://nlp.cs.nyu.edu/sekine/papers/li07.pdf`

[17] Winograd, T. *Procedures as a representation for data in a computer program for understanding natural language / by Terry Winograd.* Massachusetts Institute of Technology Cambridge, 1971, 462 p. : pp., [Online; accessed 28-December-2018].

[18] Mitchell, T. M. *Machine Learning*. New York: McGraw-Hill, 1997, [Online; accessed 10-June-2018].

[19] Weizenbaum, J. *Computer Power and Human Reason: From Judgment to Calculation*. New York, NY, USA: W. H. Freeman & Co., 1976, ISBN 0716704641, [Online; accessed 28-December-2018].

[20] Yan, R. "Chitty-Chitty-Chat Bot": Deep Learning for Conversational AI. In *Proceedings of the Twenty-Seventh International Joint Conference on Artificial Intelligence, IJCAI-18*, International Joint Conferences on Artificial Intelligence Organization, 7 2018, pp. 5520–5526, doi: 10.24963/ijcai.2018/778, [Online; accessed 28-December-2018]. Available from: `https://doi.org/10.24963/ijcai.2018/778`

[21] Ramshaw, L. A.; Marcus, M. P. Text Chunking using Transformation-Based Learning. 1995, `cmp-lg/9505040`.

[22] Dey, A.; Prukayastha, B. S. Article: Named Entity Recognition using Gazetteer Method and N-gram Technique for an Inflectional Language: A Hybrid Approach. *International Journal of Computer Applications*, volume 84, no. 9, December 2013: pp. 31–35, [Online; accessed 23-December-2018].

[23] Nadeau, D.; Sekine, S. A survey of named entity recognition and classification. 2007, [Online; accessed 30-December-2018].

[24] Liu, B.; Lane, I. Attention-Based Recurrent Neural Network Models for Joint Intent Detection and Slot Filling. *CoRR*, volume abs/1609.01454, 2016, [Online; accessed 25-December-2018], `1609.01454`. Available from: `http://arxiv.org/abs/1609.01454`

[25] Xu, P.; Sarikaya, R. Convolutional neural network based triangular CRF for joint intent detection and slot filling. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, Dec 2013, pp. 78–83, doi:10.1109/ASRU.2013.6707709, [Online; accessed 24-December-2018].

[26] Tjong Kim Sang, E. F.; De Meulder, F. 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, Stroudsburg, PA, USA: Association for Computational Linguistics, 2003, pp. 142–147, doi: 10.3115/1119176.1119195, [Online; accessed 8-December-2018]. Available from: `https://doi.org/10.3115/1119176.1119195`

[27] Burnham, K. P.; Anderson, D. R. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, second edition, 2002, ISBN 0387953647, 1–488 pp., [Online; accessed 27-December-2018].

[28] Sutton, C.; McCallum, A. An Introduction to Conditional Random Fields. *Found. Trends Mach. Learn.*, volume 4, no. 4, Apr. 2012: pp. 267–373, ISSN 1935-8237, doi:10.1561/2200000013, [Online; accessed 28-December-2018]. Available from: `http://dx.doi.org/10.1561/2200000013`

[29] Xing, E. P. Conditional Random Fields. Nov 2018, [Online; accessed 28-December-2018].

[30] Maas, W. Networks of Spiking Neurons: The Third Generation of Neural Network Models. *Trans. Soc. Comput. Simul. Int.*, volume 14, no. 4, Dec. 1997: pp. 1659–1671, ISSN 0740-6797, [Online; accessed 3-December-2018]. Available from: `http://dl.acm.org/citation.cfm?id=281543.281637`

[31] Mikolov, T. Statistical Language Models Based on Neural Networks. VUT v BRNĚ, 2012, [Online; accessed 30-December-2018].

[32] Hinton, G. E.; McClelland, J. L.; et al. Parallel Distributed Processing: Explorations in the Microstructure of Cognition, Vol. 1. chapter Distributed Representations, Cambridge, MA, USA: MIT Press, 1986, ISBN 0-262-68053-X, pp. 77–109, [Online; accessed 29-December-2018]. Available from: `http://dl.acm.org/citation.cfm?id=104279.104287`

[33] Mikolov, T.; Sutskever, I.; et al. Distributed Representations of Words and Phrases and their Compositionality. *CoRR*, volume abs/1310.4546, 2013, [Online; accessed 29-December-2018], `1310.4546`. Available from: `http://arxiv.org/abs/1310.4546`

[34] Morin, F.; Bengio, Y. Hierarchical Probabilistic Neural Network Language Model. In *AISTATS*, 2005, [Online; accessed 29-December-2018].

[35] Lample, G.; Ballesteros, M.; et al. Neural Architectures for Named Entity Recognition. 2016, [Online; accessed 28-December-2018], `1603.01360`.

[36] Hochreiter, S.; Schmidhuber, J. Long Short-term Memory. *Neural computation*, volume 9, 12 1997: pp. 1735–80, doi:10.1162/neco.1997.9.8.1735, [Online; accessed 28-December-2018].

[37] Ruder, S. An overview of gradient descent optimization algorithms. 2016, [Online; accessed 27-December-2018], `1609.04747`.

[38] Cho, K.; van Merrienboer, B.; et al. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. 2014, [Online; accessed 26-December-2018], `1406.1078`.

[39] Schuster, M.; Paliwal, K. Bidirectional Recurrent Neural Networks. *Trans. Sig. Proc.*, volume 45, no. 11, Nov. 1997: pp. 2673–2681, ISSN 1053-587X, doi:10.1109/78.650093, [Online; accessed 26-December-2018]. Available from: `http://dx.doi.org/10.1109/78.650093`

[40] LeCun, Y.; Bottou, L.; et al. Gradient-Based Learning Applied to Document Recognition. 1998, [Online; accessed 17-December-2018].

[41] Peters, M. E.; Neumann, M.; et al. Deep contextualized word representations. 2018, [Online; accessed 26-December-2018], `1802.05365`.

[42] Hawkins, D. The Problem of Overfitting. *Journal of chemical information and computer sciences*, volume 44, 05 2004: pp. 1–12, doi: 10.1021/ci0342472, [Online; accessed 26-December-2018].

[43] Pennington, J.; Socher, R.; et al. GloVe: Global Vectors for Word Representation. In *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543, [Online; accessed 27-December-2018]. Available from: `http://www.aclweb.org/anthology/D14-1162`

[44] Bojanowski, P.; Grave, E.; et al. Enriching Word Vectors with Subword Information. *arXiv preprint arXiv:1607.04606*, 2016, [Online; accessed 27-December-2018].

[45] Braun, D.; Hernandez-Mendez, A.; et al. Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. In *Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue*, Association for Computational Linguistics, 2017, pp. 174–185, doi:10.18653/v1/W17-5522, [Online; accessed 29-December-2018]. Available from: `http://aclweb.org/anthology/W17-5522`

[46] Serban, I. V.; Lowe, R.; et al. A survey of available corpora for building data-driven dialogue systems. *arXiv preprint arXiv:1512.05742*, 2015, [Online; accessed 24-December-2018]. Available from: `https://arxiv.org/pdf/1512.05742.pdf`

[47] Li, X.; Roth, D. Learning Question Classifiers. In *Proceedings of the 19th International Conference on Computational Linguistics - Volume 1*, COLING '02, Stroudsburg, PA, USA: Association for Computational Linguistics, 2002, pp. 1–7, doi:10.3115/1072228.1072378, [Online; accessed 29-December-2018]. Available from: `https://doi.org/10.3115/1072228.1072378`

[48] Ratinov, L.; Roth, D. Design Challenges and Misconceptions in Named Entity Recognition. In *CoNLL*, 6 2009, [Online; accessed 7-August-2018]. Available from: `http://cogcomp.org/papers/RatinovRo09.pdf`

[49] Limsopatham, N.; Collier, N. Bidirectional LSTM for Named Entity Recognition in Twitter Messages. In *Proceedings of the 2nd Workshop on Noisy User-generated Text (WNUT)*, The COLING 2016 Organizing Committee, 2016, pp. 145–152, [Online; accessed 27-December-2018]. Available from: `http://aclweb.org/anthology/W16-3920`

[50] Guo, D.; Tür, G.; et al. Joint semantic utterance classification and slot filling with recursive neural networks. *2014 IEEE Spoken Language Technology Workshop (SLT)*, 2014: pp. 554–559, [Online; accessed 24-December-2018].

[51] Xu, P.; Sarikaya, R. Convolutional neural network based triangular CRF for joint intent detection and slot filling. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*, Dec 2013, pp. 78–83, doi:10.1109/ASRU.2013.6707709, [Online; accessed 24-December-2018].

[52] Devlin, J.; Chang, M.-W.; et al. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. 2018, [Online; accessed 27-December-2018], `1810.04805`.

Appendix A

Abbreviations

CNN	Convolutional Neural Networks
CRF	Conditional Random fields
GRU	Gated Recurrent Unit
HMM	Hidden Markov models
IR	Intent recognition
LSTM	Long Short Term Memory
NER	Named Entity Recognition
NLP	Natural Language Processing
POS	Part-of-Speech
RNN	Recurrent Neural Network
SMT	Statistical Machine Translation

Appendix B

Contents of CD

`Generating tool`	Sources of module for possibility to generate dialogs
`editor`	Editor to create dialog tree
`make_model`	Python code to create dataset from dialogs
`models`	Saved dialogs
`Manual labeling tool`	Websites to manually labeling dialogs
`app`	Source part of manual labeling tool
`data`	Data to label
`modules`	Helpers and modules
`static`	Javascript and CSS to design tool
`templates`	Flask templates
`Model REST API`	REST API server for using a trained model
`model`	Saved model and helper files
`run_keras_server.py`	The Python code to start a server
`src`	The evaluation code
`jupyters`	Interactive Jupyter notebooks
`*.ipynb`	The interactive evaluation
`python`	Python code for evaluation
`*.py`	Getting confidential interval on dataset
`Text`	Text documents
`thesis.pdf`	The master's thesis in PDF
`thesis.tex`	The master's theis in TEX