



Technische Universität Berlin

Institute of Software Engineering
and Theoretical Computer Science

Part-of-Speech Tagging
with Neural Networks
for a Conversational Agent

Master's Thesis

Master of Science (M.Sc.)

Author Andreas Müller
Major Computer Engineering
Matriculation No. 333471

Date 18th May 2018
1st supervisor Prof. Dr.-Ing. Sebastian Möller
2nd supervisor Dr. Axel Küpper

Eidesstattliche Erklärung

Hiermit versichere ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Ausführungen, die anderen veröffentlichten oder nicht veröffentlichten Schriften wörtlich oder sinngemäß entnommen wurden, habe ich kenntlich gemacht.

Die Arbeit hat in gleicher oder ähnlicher Fassung noch keiner anderen Prüfungsbehörde vorgelegen.

Berlin, den May 12, 2018

Unterschrift

Abstract

A part-of-speech tagger is a system which automatically assigns the part of speech to words using contextual information. Potential applications for part-of-speech taggers exist in many areas of computational linguistics including speech recognition, speech synthesis, machine translation or information retrieval in general.

The part-of-speech tagging task of natural language processing is also used in 'ALEX', an advisory artificial conversational agent. ALEX was developed to answer questions about modules and courses at the Technische Universität Berlin. The system takes the written natural language requests from the user and transforms them into SQL queries. To understand the natural language queries, the system uses a hidden Markov model (HMM) to assign tags to each word of the query (part-of-speech tagging). This HMM tagger was trained with manually created training templates that are filled with the data in the database to be queried. The combination of manually created sentence-templates and slot-filling resulted in many training data sentences with the same structure. This often led to wrong tagging results when the HMM tagger was presented with an input sentence, having a structure that does not occur in the training templates.

This thesis shows two different neural network approaches for the language modeling of the input sentences and evaluates and compares both neural network based tagger as well as the HMM based tagger.

Zusammenfassung

Ein Part-of-speech Tagger ist ein System, welches Wortarten anhand von Kontextinformationen automatisch den gegebenen Wörtern zuordnet. Potentielle Anwendungen solcher Tagger gibt es in vielen Bereichen der Computerlinguistik wie Spracherkennung, Sprachsynthese, maschinelle Übersetzung oder Information Retrieval im Allgemeinen.

Part-of-speech Tagging wird auch in ALEX verwendet, einem Artificial Conversational Agent. ALEX wurde entwickelt, um Fragen zu Modulen und Lehrveranstaltungen an der Technischen Universität Berlin zu beantworten. Das System nimmt die in natürlicher Sprache geschriebenen Anfragen des Benutzers und versucht diese in SQL-Abfragen umzuwandeln. Um die natürliche Sprache zu verstehen, verwendet das System ein Hidden-Markov-Model (HMM), um jedem Wort der Eingabe Wortarten zuzuweisen (Part-of-speech Tagging). Dieser HMM-Tagger wird mit manuell erstellten Trainingsvorlagen trainiert, die mit den Daten der abzufragenden Datenbank gefüllt werden. Die manuell erstellten Satzvorlagen führten zu vielen Trainingsdatensätzen mit gleicher Struktur und damit oft zu falschen Tagging-Ergebnissen, wenn der HMM-Tagger einen Eingabesatz mit einer Struktur verarbeiten sollte, die in den Trainingsvorlagen nicht vorkommt.

Diese Arbeit zeigt zwei verschiedene Ansätze für die Sprachmodellierung der Eingabesätze basierend auf neuronalen Netzwerken und bewertet und vergleicht sowohl die Neuronalen Netzwerk-basierten Tagger als auch den HMM-basierten Tagger.

Contents

List of Figures	11
List of Tables	12
Abbreviations	13
1 Introduction	14
1.1 Scope of this Thesis	15
1.2 Related Work	16
1.2.1 The Hidden Markov Model	17
1.2.2 The Artificial Neural Network Model	18
1.3 Structure of this Thesis	19
2 ALEX: Artificial Conversational Agent	21
2.1 System Overview	21
2.2 Training Data	23
2.3 The Hidden Markov Model Tagger	23
2.4 Tagging Interface	27
3 Part-of-Speech Tagging with Neural Networks	28
3.1 Feed-forward Neural Network Model	29
3.1.1 Architecture	29
3.1.2 Implementation	30
3.2 Recurrent Neural Network Model	32
3.2.1 Architecture	32
3.2.2 Implementation	33
4 Training of Language Models	35
4.1 Training Data Corpus	35
4.2 Parameter Tuning	38

Contents

5	Evaluation and Comparison	41
5.1	Test Design	41
5.2	Evaluation Results	43
5.2.1	Feed-Forward Neural Network Models	44
5.2.2	Recurrent Neural Network Models	47
5.2.3	Hidden Markov Models	47
5.3	Overall Comparison	49
6	Discussion and Conclusion	50
6.1	Summary	50
6.2	Discussion	50
6.3	Future work	50
	Bibliography	53
A	Appendix	55
A.1	Set of sentence templates	55

List of Figures

1.1	User Interface of ALEX	16
2.1	Component Overview of ALEX	22
2.2	Structure of a Hidden Markov Model	24
3.1	Structure of a Feed-forward Neural Network	30
3.2	Creation of the feature vector	31
3.3	Structure of a Recurrent Neural Network	33
5.1	FNN Evaluation: Number of Past Words	44
5.2	FNN Evaluation: Number of Past Words	45
5.3	FNN Evaluation: Hidden Layer Size	46
5.4	FNN Evaluation: Number of Training Epochs	47
5.5	HMM Evaluation	48

List of Tables

2.1	Tagging Scheme Overview	26
4.1	Sentence Template Improvements	37
4.2	Parameter combinations of FNN Models	39
4.3	Parameter combinations of RNN Models	40
5.1	Evaluation Topics using the Known Test Set	42
5.2	Evaluation Topics using the Unknown Test Set	43
5.3	Interpretation of Cohen's Kappa	44

Abbreviations

ACA	<i>Artificial Conversational Agent</i>
ANN	<i>Artificial Neural Network</i>
FNN	<i>Feed-forward Neural Network</i>
HMM	<i>Hidden Markov Model</i>
LSTM	<i>Long Short-Term Memory</i>
NLP	<i>Natural Language Processing</i>
NLTK	<i>Natural Language Toolkit</i>
POS	<i>Part-of-Speech</i>
RNN	<i>Recurrent Neural Network</i>
SGD	<i>Stochastic Gradient Descent</i>

1 Introduction

Learning is one of the most essential parts of human life. From their first day to their last, humans acquire knowledge and skills. Learning involves progress, additional value, failure and repetition. It enables growth and improvement.

In biology, learning is based on a specific strengthening of the connection of certain nerve cells in the central nervous system by facilitating signal transmission at the synapses through appropriate modifications. As a huge amendable network of connected neurons, the nervous system has been serving as a model for a research field called *Machine Learning*. This term was coined by A. Samuel¹ [18] in 1959, who distinguished between two general approaches of solving the problem of machine learning: a general-purpose randomly connected neural network approach and a special-purpose highly organized network. Following a publication of W. McCulloch about the comparison of a computer with the nervous system of a flatworm in 1949 [12], Samuel observed:

*“A comparison between the size of the switching nets
that can be reasonably constructed or simulated at the present time
and the size of the neural nets used by animals,
suggests that we have a long way to go before we obtain practical devices.”*

– Arthur Lee Samuel (1959)

Today, less than 60 years later, we have a lot of practical devices that use machine learning and artificial intelligence and, by now, they not only play a role in science but in our everyday life. Especially the processing and understanding of spoken or written natural language has a wide range of applications today. One of those areas of application are advisory artificial conversational agents (ACA) or chatbots. They are designed to provide natural language

¹ Arthur Lee Samuel was an early researcher in machine learning and artificial intelligence. He developed the first successful self-learning program: the Samuel-Checkers game [18].

answers to natural language questions, making it as easy as possible for users to interact with a specific system. ALEX is an example of an ACA that is designed to answer questions about courses and modules of the TU Berlin. This thesis aims to improve how ALEX understands and learns natural language using *artificial neural networks* (ANNs).

1.1 Scope of this Thesis

This thesis focuses on developing a neural network based part-of-speech tagger for ALEX, the advisory Artificial Conversational Agent, the training of several language models and their evaluation with corresponding test sets.

In order to devise the new language models, two different neural network architectures are implemented: a feed-forward neural network and a recurrent neural network. For the training of both neural network implementations, a corpus of tagged language data is generated with the help of various input templates, which are created based on logged user input data.

To evaluate the language models, a data set of known data² and unknown data³ is created. Both neural network models and the HMM are compared to each other using this evaluation.

According to the evaluation results, the initial HMM based part-of-speech tagger is then replaced by a new tagger. To guarantee a seamless integration, the new tagger is implemented as a separate module with the same program interface that the old tagger already utilizes. This way, no other components of the conversational agent have to be changed and the effort of the replacement is kept to a minimum.

² Data, that was already used for the training of the model

³ Data, that includes words and sentence structures, that didn't occur in the training data sets

1.2 Related Work

This thesis is built upon the work of T. Michael [14], who describes the design and implementation of ALEX in detail. The conversational agent was implemented to help students of the TU Berlin to organize their studies by providing a simple way to gain information about modules and courses. It utilizes two separate already existing baseline systems by merging their data into one relational database. This database is used as the central access point for the information that users want to retrieve.

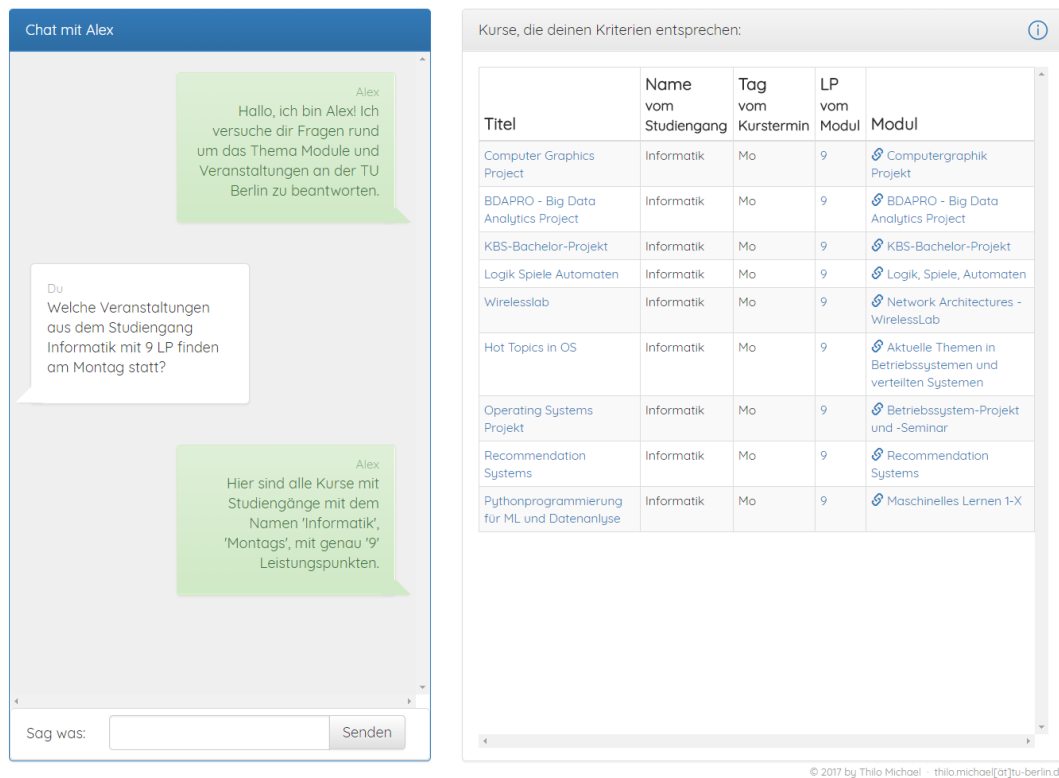


Figure 1.1: The user interface of ALEX. The left section contains the conversation with the agent and a user entry field. The right section shows the result of the generated database query in tabular form.

In this example, the user asked for all courses of the subject *computer science* that provide 9 ECTS and are scheduled on a Monday. The agent answered accordingly and provided a list of 9 courses that fulfill the conditions.

This image was captured on 21st April 2018.

ALEX consists of several processing modules:

1 Introduction

- The **tagging module** uses a hidden Markov model to calculate the parts of speech for the user input, later described in chapter 2.3
- The **query generation module** composes actual SQL queries from the tagged output data by recognizing the requested model and the return type
- The **filter extraction module** refines the query generator and provides constraint handling for it
- The **response generation module** formulates answers for the user input using natural language by processing the generated query, the recognized model and the conversation state.

Moreover, ALEX provides a user interface, which utilizes web technologies and can be accessed via a web browser. Figure 1.1 shows the user interface, in which the user asked a question and the agent returned the result in tabular form and answered accordingly.

The focus of this thesis lies on the tagging module, as the main objective is to replace the hidden Markov model by artificial neural networks.

1.2.1 The Hidden Markov Model

The hidden Markov model (HMM) is a probabilistic finite state machine that solves general classification problems. It uses the observable output data of a system to derive hidden information from it. Among other applications, HMMs are used especially for speech recognition tasks.

The preliminary work for HMMs was done by R. L. Stratonovich. He first described conditional Markov processes in 1960 [21], which were used in the following years to describe simple Markov Models and later hidden Markov models (see Baum et. al. [3][2]). The latter became a popular solution for automatic recognition of continuous speech [1] along with other applications, such as pattern recognition in general, the analysis of biological sequences (e.g. DNA) [4] and part-of-speech tagging [9].

1.2.2 The Artificial Neural Network Model

Artificial neural networks are networks that process information inspired by the biological nervous system. They consist of connected computational units, typically arranged in different layers. Such a unit (also called *artificial neuron*) can make calculations based on its inputs and pass the result to the neighboring units. These connections are weighted, so that the weight can be adjusted depending on the activity of the unit. Thus, a model based on the features of the input data can be created.

Following research by W. McCulloch, W. Pitts [13] and D. Hebb [20] on arithmetical learning methods, inspired by the connections of neurons in the 1940s, M. Minsky built the first neural network learning machine called SNARC (*Stochastic Neural Analog Reinforcement Computer*)[6] in 1951.

In the late 1950s, F. Rosenblatt developed the *Mark I Perceptron* computer and published a theorem of convergence of the perceptron[17] in 1958. He coined the term *perceptron* for an algorithm that is able to learn the assignment of input data to different classes. The perceptron represents a simple artificial neural network initially containing one single neuron⁴. F. Rosenblatt stated that any function that is representable by the model can be learned with the proposed learning method. In 1960, B. Widrow presented the ADALINE⁵ model of a neural network, for which input weights could already be adjusted by the learning algorithm [22].

A publication of M. Minsky and S. Papert [15] from 1969 analyzed and exposed some significant limitations of the basic perceptron. They pointed out that it is impossible to learn functions without linear separability (e.g. the exclusive-or problem). Due to these limitations and the fact that the processing power of computers at the time was not sufficient for larger neural networks, research interest in artificial neural networks decreased in the following years.

⁴ Chapters 3.1 and 3.2 explain the architecture of different neural network structures in detail

⁵ ADALINE is an acronym for Adaptive Linear Neuron

1 Introduction

In 1982, J. Hopfield presented a previously described neural network with feedback (known as *Hopfield network*), that was able to solve optimization problems like the *Traveling Salesman Problem*⁶. Neural network approaches started to receive more attention again, also due to the first processors based on transistor technology (microprocessors) being introduced to the market in the early 1970s, replacing the previously used tube technology in the following years, which made computers smaller and cheaper and increased their processing capacity.

For the task of POS tagging, neural network models were now able to outperform HMM based taggers. H. Schmid created and trained a multilayer Feed-forward neural network in 1994 and was able to show that it performed better than an HMM tagger [19] at that time. In 2000, Ma et. al. ran a series of comparative experiments that proved that results of a neural network tagger can be superior to those of statistical models like the HMM [11].

1.3 Structure of this Thesis

This first chapter outlined the subject of natural language processing and part-of-speech tagging in general as an introduction.

The second chapter describes the structure and functionality of the already existing ACA, ALEX, with the main focus on its language model and tagging interface.

Chapter 3 explains the implementation of a part-of-speech tagging system using two different neural network approaches.

The training of the language models including the retrieval of training data and tuning training parameters is described in Chapter 4.

⁶ The problem of the traveling salesman or round trip problem: The order of places to be visited once should be chosen in such a way that the distance covered is minimal, whereby the last place is the starting point again (round trip).

1 Introduction

Chapter 5 illustrates the evaluation of each language model with a generated test set and their comparisons.

In conclusion, 6 discusses and summarizes the evaluation results and presents an outlook on future work.

2 ALEX: Artificial Conversational Agent

Design and Implementation of an Advisory Artificial Conversational Agent by T. Michael [14] provides a detailed and comprehensive description of ALEX as a compilation of different modules. This chapter focuses on components that are relevant for language processing and were therefore adapted during this thesis: The retrieval and processing of training data (section 2.2), the hidden Markov model tagger (section 2.3) and the tagging interface (section 2.4).

2.1 System Overview

The modular structure of ALEX facilitates the separation of different functions and therefore simplifies the replaceability of certain functionalities. Besides a module crawler for current data retrieval of web content for the database and a front-end interface module, ALEX offers a tagging module. This module enables the training of a language model as well as the assignment of tags to words of a given input sentence.

Figure 2.1 shows the original architecture of ALEX. The following components are part of the tagging module and are adapted or replaced (emphasized with an orange border in figure 2.1) in this thesis:

- The **HMM Training Data** is replaced by training data that was generated with improved training sentence templates (see chapter 4.1 for a comprehensive explanation)
- **Training Data Loading and Slot Filling** are used to generate the new training data
- The **HMM Tagger** is replaced by a neural network based tagger

2 ALEX: Artificial Conversational Agent

- **Part of Speech Tagging** of input sentences is realized by the tagging function of the new tagger

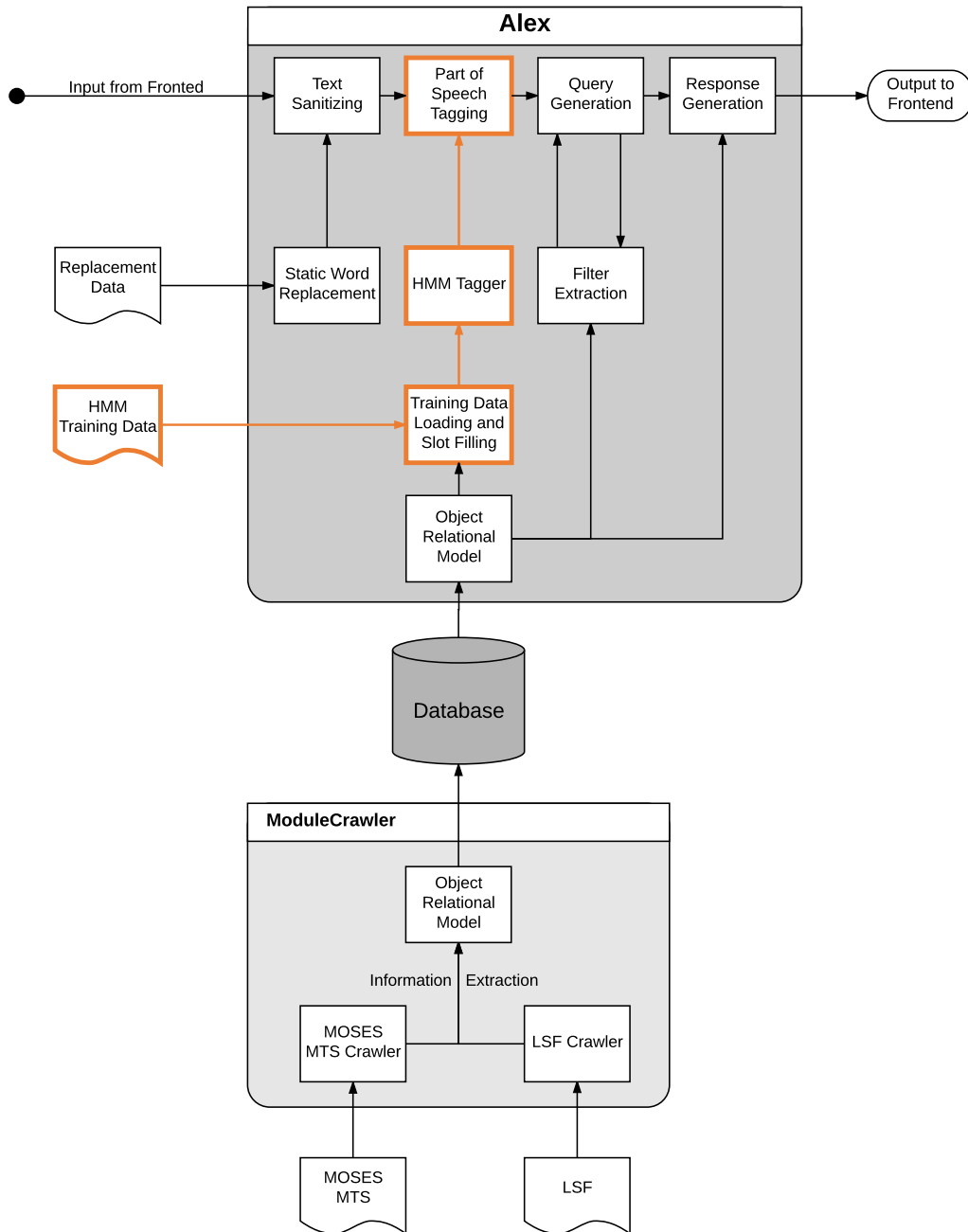


Figure 2.1: Overview of all components of ALEX. The orange borders identify components that lie within the scope of this thesis and have been adapted or replaced. Original figure by T. Michael [14]

2.2 Training Data

In order to teach ALEX to assign tags to words correctly, depending on their context, appropriate training data is required. The training data for ALEX proposed by T. Michael consists of 556,111 tagged sentences generated with 72 manually created sentence templates.

A sentence template is a sentence that provides the structure of a possible tagged training sentence with proper syntax for slot filling. It consists either of special placeholders for specific data from the database (e.g. module titles), inline choices (e.g. the same sentence with each day of the week) or a marker to simply duplicate the sentence. The different slot filling forms can be combined or used multiple times in one sentence¹.

For training the neural network models in this thesis, this slot filling mechanism is adopted and improved for the training with artificial neural networks (see chapter 4).

2.3 The Hidden Markov Model Tagger

As described in section 1.2.1 of the introduction, the hidden Markov model (HMM) is a statistical tool that uses observable output data of a system to derive hidden information from it. Areas of application are image processing, gesture recognition and natural language processing tasks, such as speech recognition in general and part-of-speech tagging in particular.

In case of POS tagging, the observable states of the HMM represent the given sequence of words, whereas the hidden states represent the corresponding parts of speech. The HMM calculates the joint probability of the whole sequence of hidden states based on transmission and output probabilities. Subsequently, it finds the maximum probability of all possible state sequences and determines as a result, which parts of speech most likely correspond to the words of the input sequence.

¹ T. Michael describes the training data structure in detail in chapter 3.4.2 of *Design and Implementation of an Advisory Artificial Conversational Agent* [14].

2 ALEX: Artificial Conversational Agent

Figure 2.2 illustrates an example of a state sequence with three hidden states (part of speech tags) and the observed word sequence in an HMM. The calculation of the joint probability P of the word sequence in this case is shown in equation 2.1, as the product of transmission and output probabilities.

$$P = p_{start} \cdot p_{out,1} \cdot p_{trans,1} \cdot p_{out,2} \cdot p_{trans,2} \cdot p_{out,3} \quad (2.1)$$

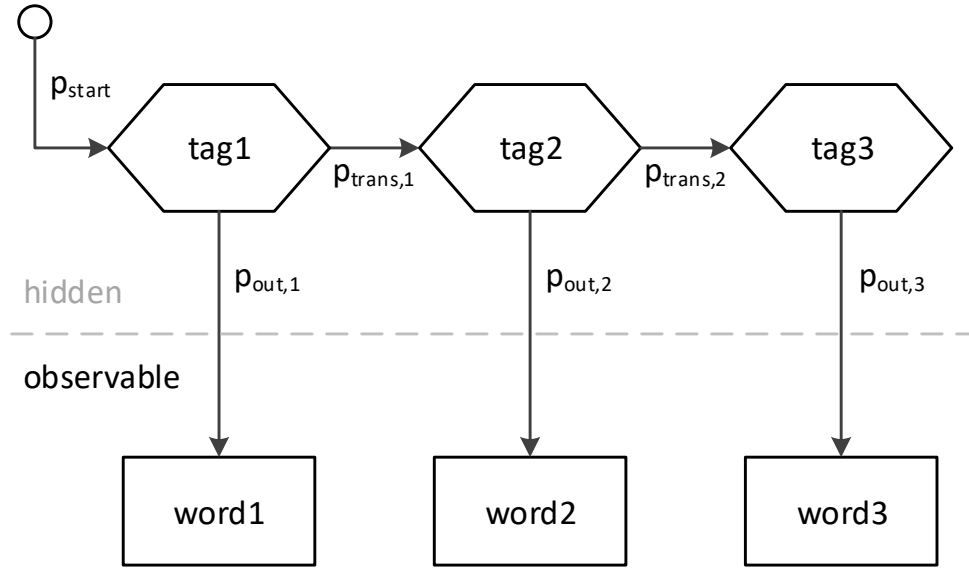


Figure 2.2: An example of a state sequence of three hidden states (tag1 – tag3) and an observed sequence of three words (word1 – word3) in a hidden Markov model. p_{start} denotes the starting probability, p_{trans} the transmission probabilities between hidden states and p_{out} the output probabilities between a hidden state and an output.

For the purpose of finding corresponding parts of speech to a given input sentence, an HMM is included in ALEX. According to T. Michael [14], a tagging scheme was developed to extract exactly the information from user input that is needed to successfully create a database query and to return the information the user asked for. This tagging scheme was intentionally

2 ALEX: *Artificial Conversational Agent*

built to be domain independent, creating the opportunity to develop a version of ALEX for any topic based on an appropriate database and training data.

To maintain this universal applicability as well as the compatibility with other modules of ALEX, the neural network approaches presented in this thesis use the same tagging scheme ALEX already utilizes. For a better understanding of the evaluation results in chapter 5, table 2.1 gives an overview of the 6 different classes of tags that ALEX uses.

2 ALEX: Artificial Conversational Agent

	FORMATS	DESCRIPTION	EXAMPLE
R	<i>Return-tags</i> , describing data that is expected to be returned	R_LIST R_SINGLE R_COUNT	" Which modules..." " Which module..." " How many modules..."
M	<i>Model-tags</i> , describing the database model, e.g. M_MTSModule or M_Course	M_[MODEL]	"Which modules ..." "Which courses ..."
C	<i>Constraint-tags</i> , filtering the result set, given a database model and corresponding field, e.g. C_MTSModule:ects	C_[MODEL] : [FIELD]	"Modules with 6 ects..."
P	<i>Property-tags</i> , indicating to include fields in the result set, e.g. P_MTSModule:ects	P_[MODEL] : [FIELD]	"Modules with 6 ects ..."
Q	<i>Comparison-tags</i> , describing an equal, greater than or less than constraint	Q_EQ Q_LT Q_GT	"...with exactly 6 ects..." "... less than 6 ects..." "... more than 6 ects..."
X	<i>Extra-tags</i> , describing words that are not relevant for the database query ²	X X_[WORD] X_[MODEL] : [FIELD]	" and ", " of ", " is " "I need help " " Professor John Doe"

Table 2.1: Overview of the tagging scheme used in ALEX, consisting of 6 different classes of tags with a total of 12 different formats. The examples contain **emphasized** words that belong to the corresponding tag formats. T. Michael [14] provides a detailed explanation of the tagging classes and its formats.

² This can refer to words with no specific meaning (tagged with X), words that have no meaning for the database query but for the system itself (e.g. the tag X_HELP for the word "help") or words that lead to a particular constraint (such as the tag X_Person:fullname for the word "Professor", leading to a name).

2.4 Tagging Interface

As described in the previous chapter, the implementation of the tagging module of ALEX utilizes a hidden Markov model for part-of-speech tagging. ALEX uses an already existing implementation of the HMM Tagger from the Natural Language Toolkit (NLTK)³, called `HiddenMarkovModelTagger`.

To replace the existing tagger, a new tagger has to provide a class with two methods: `train` and `tag`. These methods are used to create the language model and apply it to unknown data.

The `train` method creates a new instance of the tagger class, trains this class with the given training data and returns it. The training data itself must be a list of sentences, where a sentence is a list of tuples, containing each word of this sentence and its corresponding tag. The following exemplifies the structure of the training input data containing two sentences, where each word is tagged with *TAG*:

```
[
  [ ('the', TAG), ('dog', TAG), ('is', TAG), ('running', TAG) ],
  [ ('the', TAG), ('cat', TAG), ('sleeps', TAG), ('all', TAG), ('day', TAG) ]
]
```

The `tag` method attaches a tag to each word of an input sentence, according to the previously trained language model. The input has to be an unknown sentence formatted as a simple list of words:

```
[ 'an', 'unknown', 'test', 'sentence' ]
```

The output is a corresponding list of tuples containing a word and its assigned tag:

```
[ ('an', TAG), ('unknown', TAG), ('test', TAG), ('sentence', TAG) ]
```

3 The Natural Language Toolkit is a collection of *Python* programming libraries for natural language processing, see <http://nltk.org>

3 Part-of-Speech Tagging with Neural Networks

Chapter 1.2.2 introduced neural networks as a possible approach for POS tagging. The current chapter presents two different neural network architectures and their implementation that is later used to train and evaluate language models with the objective of comparing their POS tagging accuracy with the accuracy of the HMM tagger that ALEX uses.

In general, a neural network represents a mathematical function, which is able to assign specific output data to corresponding input data. This assignment is learned by processing input data whose output is known. Thus, it belongs to the category of supervised learning¹.

The network itself emerges from the interconnection of nodes (the artificial neurons), that are usually arranged in different layers. Each node represents a non-linear² activation function that calculates an output, depending on the sum of its inputs. Possible activation functions are the sigmoid function, the hyperbolic tangent or the maximum function.

The training effect is achieved by weighting the connections of the nodes and adjust these weights during training. The weights of neural networks are typically represented by real numbers. The adjustments are defined by a learning algorithm that utilizes a particular learning method. Using the initial or current weights of the network, an error (also called *loss*) between the prediction and the given label can be computed on the last layer (the output layer) with the help of a corresponding loss function. The training objective is to minimize the loss function of the current state of the network and adjust the weights accordingly. This can be achieved by using *backpropagation* with *stochastic gradient descent*.

1 *Supervised Learning* describes the process of gaining knowledge with the help of labeled data. Depending on the capability of the supervised learning algorithm to generalize from the given data, this knowledge can then be applied to unknown data.

2 The non-linearity is important because linear functions cannot describe some mutual exclusive feature combinations.

The POS tagging task requires the processing of words which are represented as character strings. For easier handling, each word is mapped to an integer value (the word id).

3.1 Feed-forward Neural Network Model

Feed-forward neural networks are a common type of ANNs, that consist of an input and an output layer with one or more hidden layers in between. An FNN propagates information through the network only in the forward direction, from the input to the output layer.

3.1.1 Architecture

The FNN architecture that is used in this thesis is shown in figure 3.1. The current word and the configured number of preceding words are each converted into word embeddings. These embeddings are reshaped to one vector, called the feature vector, which serves as the input layer for the neural network. The FNN contains one hidden layer, which is connected to the input layer with weight matrix V . The output layer represents the set of existing POS tags and is connected to the hidden layer with weight matrix W .

For each word of every sentence of the training corpus, a feature vector is built and the data is sequentially propagated to both the hidden and the output layer, predicting a POS tag for the current word and calculating the current loss and accuracy. After propagating the error back to adjust the weights and reduce the loss, the next training step is executed.

3 Part-of-Speech Tagging with Neural Networks

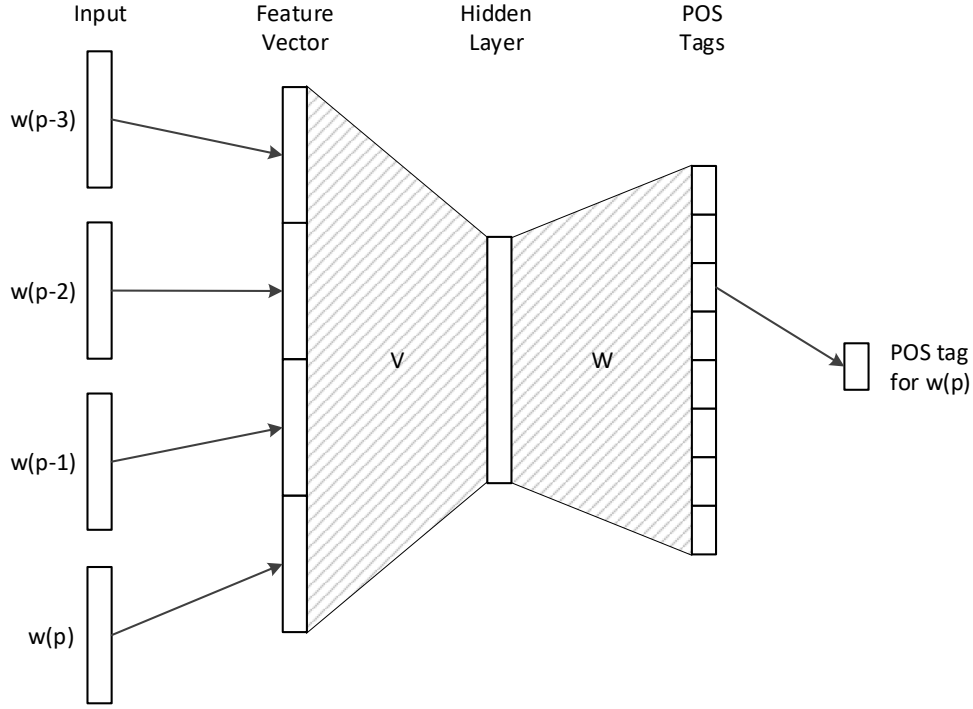


Figure 3.1: The structure of a feed-forward neural network. The feature vector is built by the embeddings of the corresponding input word at position p and by its 3 preceding words (the number of predecessors may vary, of course). V and W are weight matrices with respective matrix dimensions of feature vector size \times hidden layer size for V and hidden layer size \times number of POS tags for W .

3.1.2 Implementation

To implement the presented FNN architecture, the open source machine learning framework *TensorFlow*³ is used. In the following descriptions, all functions starting with `tf` are functions from the TensorFlow library.

The nodes of the input layer are populated by the feature vector. To create the feature vector, an embedding matrix with the dimensions of vocabulary size \times embedding size is created and initially filled with random normal distributed

³ TensorFlow is a library written in Python, that is utilized for high performance numerical computations especially for the area of machine learning. In this thesis, TensorFlow version 1.8 is used. See the official TensorFlow website: <https://www.tensorflow.org>

3 Part-of-Speech Tagging with Neural Networks

values⁴, using the `tf.truncated_normal()` function. This matrix is used to retrieve the vectors for the current word and its predecessors via `tf.nn.embedding_lookup` and reshape them to one single vector, the feature vector. Figure 3.2 illustrates the creation of the feature vector.

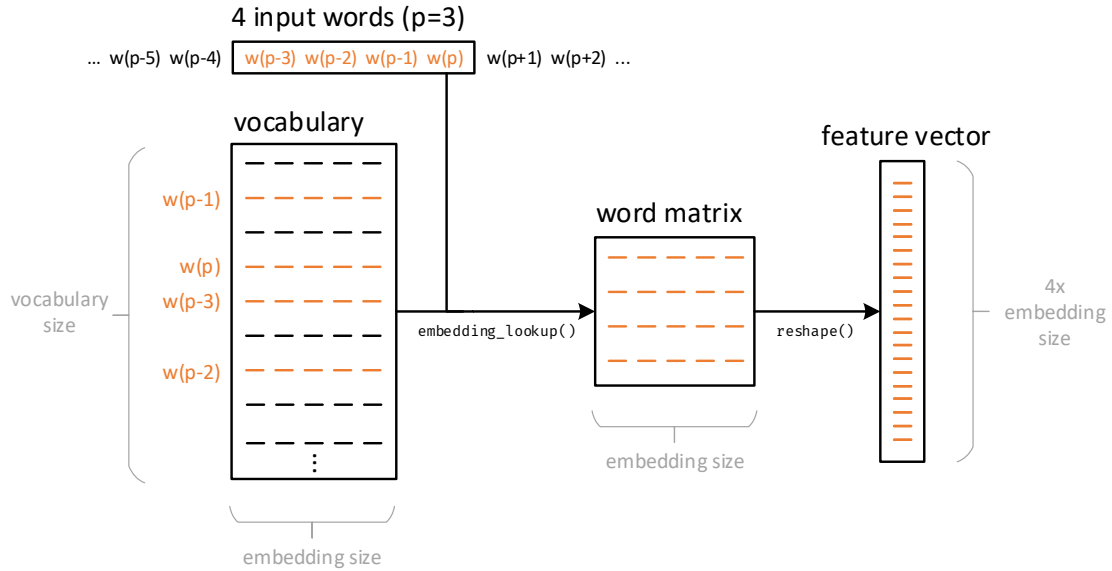


Figure 3.2: An example of the creation of a feature vector for the FNN using 3 preceding words and the current word.

The hidden layer nodes are also initialized with random normal distributed values. Their outputs are computed with the rectified linear activation function (the *rectifier*), utilizing the `tf.nn.relu()` function. These nodes are therefore also called *rectified linear units* (RELU).

Finally, the values for the output layer (the logits) are computed using a matrix multiplication of the outputs of the RELUs and the weight matrix W using the `tf.matmul()` function. Subsequently, the loss can be calculated by computing the sparse softmax cross entropy between input labels and logits, using the `tf.nn.sparse_softmax_cross_entropy_with_logits()` function.

⁴ The generated values follow a normal distribution with mean of 0 and standard deviation of 0.1, except that values whose magnitude is more than two standard deviations from the mean are dropped (truncated) and re-picked. See the TensorFlow documentation.

The predictions are calculated with the `tf.argmax()` function, that reduces the logits to the largest value, which represents the predicted POS tag. The function `tf.train.AdamOptimizer()`⁵ enables backpropagation with stochastic gradient descent to optimize the computations and results during training.

3.2 Recurrent Neural Network Model

Similar to FNNs, recurrent neural networks (RNNs) also consist of an input, a hidden and an output layer. The main difference is that information is not only propagated forward linearly, but that information from previous training steps are also taken into account in RNNs.

3.2.1 Architecture

Figure 3.3 illustrates the architecture of an RNN, which was implemented for this thesis. Unlike the FNN, the RNN only uses word ids as input features and does not utilize word embeddings. It contains one hidden layer, which is populated with the current input word on the one hand and by the output of the hidden layer of a previous training step on the other hand. This makes the RNN capable of memorizing information from the past. This capability can also be described as a long short-term memory (*LSTM*), emphasized in orange in figure 3.3. The output layer represents the set of existing POS tags and is connected to the hidden layer with weight matrix W .

Each word of every sentence of the training corpus represents an input feature, which is propagated to the hidden and the output layer together with information from previous training steps. Similar to the processing of FNNs, the POS tag is predicted for the current word and the current loss and accuracy are calculated. After propagating the error back to adjust the

⁵ The Adam algorithm is an optimizer that was proposed by D. Kingma et. al. in 2014 [8]. It provides a computationally efficient gradient-based optimization of stochastic objective functions.

3 Part-of-Speech Tagging with Neural Networks

weights and reduce the loss, the next training step is executed, taking the current training step into account.

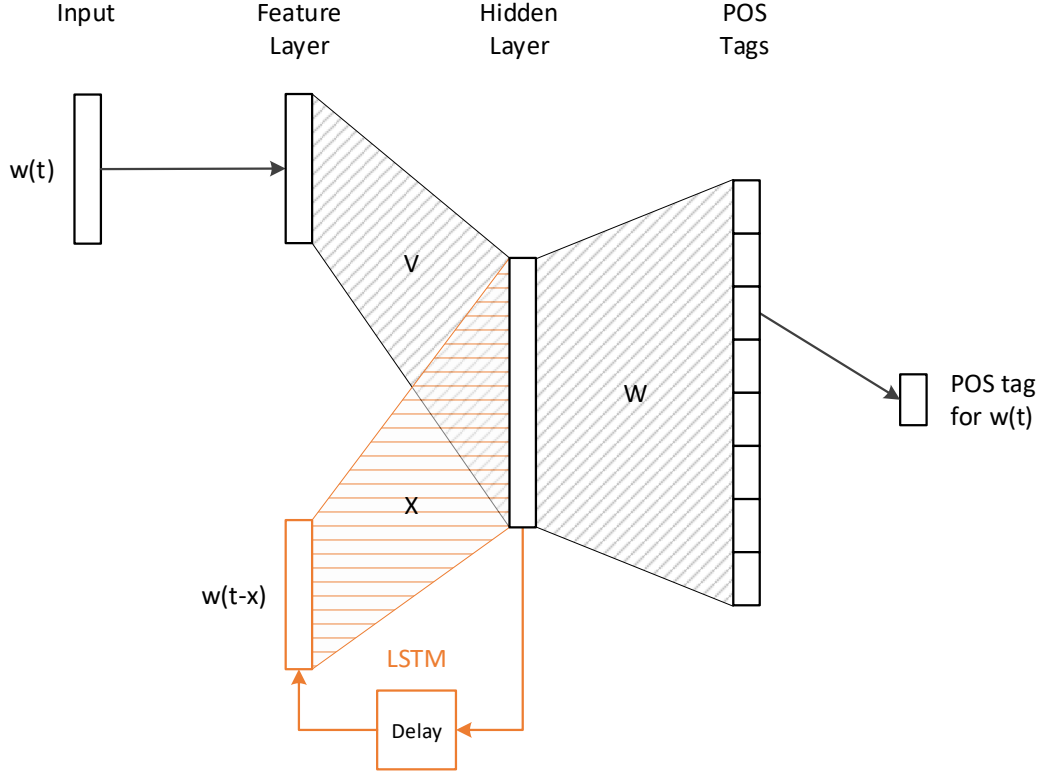


Figure 3.3: The structure of a recurrent neural network. The feature vector is the initial vector of the corresponding input word at time t . The output of the hidden layer from previously trained words (here at time $t-x$) is populated back into the same hidden layer for the current word. V , X and W are weight matrices with respective matrix dimensions of feature vector size \times hidden layer size for V , hidden layer size \times hidden layer size for X and hidden layer size \times number of POS tags for W .

3.2.2 Implementation

The implementation of the proposed RNN architecture utilizes the TensorFlow library as well. Besides the word id of the current word, the input layer has an additional dimension describing the number of past training steps that are considered for the current training step.

3 Part-of-Speech Tagging with Neural Networks

The hidden layer is built using `tf.nn.rnn_cell.LSTMCell()`⁶, a function that creates a long short-term memory (LSTM) cell for recurrent networks which is then used in `tf.nn.dynamic_rnn()` to compute the logits for the output layer.

Loss, predictions and accuracy are implemented in a similar way as in the FNN. In addition, the same optimizer is used.

⁶ This function uses an implementation proposed by S. Hochreiter et. al. in 1997 [7].

4 Training of Language Models

An optimal language model using the neural network approach proposed in this thesis, requires as much annotated training data as possible. The following sections describe the generation of tagged training data and the training of different language models, due to parameter variation, based on this generated training corpus.

4.1 Training Data Corpus

To create a training corpus with tagged sentences, the already existing sentence template set of ALEX is used as the basis for an improved and extended template set. In addition, a log of user input data¹ provides useful information on possible input sentences, that are not yet considered in the template set.

As described in chapter 2.2, the HMM tagger of ALEX uses 72 sentence templates to generate annotated training data. However, the distribution of the generated sentences is highly unbalanced. Due to the combination of placeholders for different database fields and models, which are semantically meaningless, a huge number of training sentences exist that are only partially suitable for training.

An analysis of the generated training corpus that was used to train the HMM tagger showed that one single template of the 72 sentence templates created more than 84% of the whole training corpus. This sentence template is the following²:

```
Welche  Module  werden von  Prof  {Person:firstname} {Person:lastname} angeboten
R_LIST M_MTSModule  X      X  X_Person C_Person:fullname  C_Person:fullname      X
```

1 The log data that was available for this thesis started on 30th of August, 2017 until 27th of April, 2018 and contained 1293 entries of user input.

2 The corresponding English translation of this sentence template is: *Which modules are held by Professor <firstname> <lastname>*

4 Training of Language Models

It combines all first names and all last names of each existing person in the database, leading to a huge number of name combinations that do not exist.

Another example is the combination of study program degrees and program names. This combination exists in 5 of the 72 sentence templates, i.e. all degrees are assigned to all study programs 5 times, although not every combination exists in reality. The following is an example of one of these sentence templates³:

```
Welche  Module    kann ich  im  {Program:degree}  {Program:name}  belegen
R_LIST M_MTSModule  X      X    X    C_Program:degree  C_Program:name    X
```

To address this issue, the following adjustments were made for the new template set were made:

- Because the number of occurrences is very low in the log database, the slot {Person:firstname} was removed completely from the sentence template, which limits the user input to last names only⁴.
- The slot {Program:degree} was partly replaced by the inline choice (bachelor|master|diplom), which included the main terms requested in the logs.

Furthermore, wording referring to linking words and actions, which occurs in the logs, was improved with the help of inline choices. The following table shows an excerpt of those improvements, comparing the previous and the new sentence template:

3 The corresponding English translation of this sentence template is: *Which modules can I attend to in <program-degree> <program-name>*

4 This might seem like a degradation but is justified by the fact, that only 3 of 1293 log entries (0.2%) even contained a first name, always followed by the last name. In this specific use case, where all persons are professors and lecturers, using the last name only is a reasonable approach.

4 Training of Language Models

PREVIOUS TEMPLATE	NEW TEMPLATE	ENGLISH EQUIVALENT
Alle Module vom {Program:degree} {Program:name}	Alle Module (vom von in im) {Program:degree} {Program:name}	<i>All modules (of in)</i> <i>{Program:degree}</i> <i>{Program:name}</i>
Welche Module werden von Prof {Person:firstname} {Person:lastname} angeboten	Welche Module werden von Professor {Person:lastname} (unterrichtet angeboten gehalten)	<i>Which modules are</i> <i>(taught offered held) by</i> <i>Professor {Person:lastname}</i>
Wieviele LP hat das Modul {MTSModule:title}	(Wieviele Wieviel) LP (hat bringt) das Modul {MTSModule:title}	<i>How many</i> <i>ects</i> <i>does the</i> <i>module {MTSModule:title}</i> <i>(have yield)</i>
Informationen zu Modul MTSModule:title	(Informationen Details Mehr) (zu zum) Modul {MTSModule:title}	<i>(Information details more) of the module</i> <i>{MTSModule:title}</i>

Table 4.1: An excerpt of the extension and improvement of the sentence templates by using inline choices. The last column provides the corresponding English translation for the new sentence template.

All in all, the new set of sentence templates contains 36 templates, which are exactly the same as in the old set, 29 templates that were slightly modified or extended while maintaining the same tag sequence and 35 new sentence templates. The latter were created according to a sentence structure that was found in the logs but not in the previous template set. Therefore, the new set contains a total of 100 sentence templates (including single word templates) and can be found in the appendix A.1.

The resulting training data corpus used in this thesis contains 218.700 tagged sentences consisting of 2.038.490 words, with a vocabulary of 9141 words.

4.2 Parameter Tuning

In order to achieve better evaluation results and therefore more words that are tagged correctly, several training parameters are changed. To see the effect of the training parameters on the accuracy of the resulting language model, one parameter was altered while keeping all other parameters constant.

For the training of the FNN models, the following 4 parameters were considered:

- **Number of past words (p)** specifies how many preceding words should be considered for the training of the current word
- **Embedding size (e)** describes the dimension of the word embeddings that were created for each word of the vocabulary during training
- **Hidden layer size (s)** characterizes the dimension of the hidden layer, i.e. the number of neurons that constitute the hidden layer
- **Number of training epochs (n)** indicates how often the whole training corpus is processed during training
- **Activation function (a)** ...

Based on these 5 parameters, table 4.2 shows different parameter combinations. The emphasized cells show the respective parameter that is variable. The first 4 training groups represent the basic training process to demonstrate the effect of each parameter in general. The subsequent training groups included additional parameter combinations, taking into account the evaluation results of the first training groups.

The resulting number of models is 99; however, due to an overlap in configurations of the training groups, the total number of distinct models to be trained with the FNN is ??.

4 Training of Language Models

p	e	s	n	a	MODELS
0–12	50	100	1	relu	13
1	1, 5, 10, 25	100	1	relu	4
1	50–700 ⁵	100	1	relu	14
1	50	10, 25	1	relu	2
1	50	50–1000 ⁵	1	relu	20
1	50	100	1, 5, 10	relu	3
1	50	100	20–140 ⁶	relu	7
1	250	350	1	relu, [...] ⁷	8
1	200–600 ⁵	350	1	relu	9
1	250	200–600 ⁵	1	relu	9
1	50–500 ⁵	50–500 ⁵	5	relu	10

Table 4.2: The parameter configuration for training the FNN models. The rows represent the training groups with the corresponding variable parameter (orange cells), while the columns represent the single training parameters (**p** - number of past words, **e** - embedding size, **s** - hidden layer size, **n** - number of training epochs, **a** - activation function). The last column shows the resulting number of models for each training group.

For the training of the RNN models, the following 3 parameters were considered:

- **Number of time steps (t)** specifies how many past training steps should be considered for the training of the current word

⁵ With a step size of 50

⁶ With a step size of 20

⁷ The following activation functions were considered: relu, relu6, elu, sigmoid, tanh, selu, softplus, softsign

4 Training of Language Models

- **Hidden layer size (s)** characterizes the dimension of the hidden layer, i.e. the number of neurons that constitute the hidden layer
- **Number of training epochs (n)** indicates how often the whole training corpus is processed during training

Based on these 3 parameters, table 4.3 shows different parameter combinations. The emphasized cells show the respective parameter that is variable. The resulting number of models is 18; however, due to an overlap in configurations of the training groups, the total number of models to be trained with the RNN is 16.

t	s	n	MODELS
0-10	50	10	10
1	10, 25, 50, 100	10	4
1	50	1, 5, 10, 20	4

Table 4.3: The parameter configuration for the training of RNN models. The rows represent the training groups with the corresponding variable parameter (orange cells), while the columns represent the single training parameters (**t** (number of time steps), **s** (hidden layer size), **n** (number of training epochs)). The last column shows the resulting number of models for each training group.

5 Evaluation and Comparison

After a short description of how the evaluation tests were created, this chapter provides the evaluation results of all models that were trained using the variation of training parameters previously described. Additionally, for each training group, the evaluation results are presented and the three different architectures (FNN, RNN and HMM) are compared.

5.1 Test Design

To evaluate the trained language models, two test sets were designed: a test set containing a selection of tagged sentences that actually exist in the training corpus (here called the *known test*) and a test set, for which the structure of the sentences in the known test was modified in a way that it remains semantically meaningful but does not occur in the training corpus (called the *unknown test*). The unknown test deliberately contains words, which do not exist in the vocabulary of the training corpus in order to assess how the model handles completely unknown data.

The sentences are divided into different topics to cover a wide range of possible input data. A balanced number of sentences was included in the different areas to ensure that the evaluation result does not depend on only one or two topics.

Tables 5.1 and 5.2 show the different topics represented in the known and the unknown test set, the respective number of evaluation sentences and an example sentence to illustrate each topic. Both test sets combined include a total number of 1058 tagged test sentences, consisting of 7678 word-tag tuples.

The number of correctly tagged words serves as a measure of accuracy for the respective language model. This includes unrecognized words (words that did not occur in the training corpus) that are still tagged correctly. The

5 Evaluation and Comparison

TOPIC	SENTENCES	EXAMPLE
ECTS	44	<i>Which modules have 6 ECTS</i>
Time	60	<i>Which courses are on Tuesday</i>
Faculty	56	<i>All modules of faculty 4</i>
Participants	54	<i>Which courses are limited to 30 participants</i>
Persons	61	<i>Which courses are taught by Professor Martinelli</i>
Program	60	<i>I'm searching for all modules of the computer science program</i>
Modules	80	<i>Modules with the title Cognitive Algorithms</i>
Chair	50	<i>All modules of institute Media Science</i>
Exam	32	<i>Courses with Group Lecture as examination</i>
Courses	45	<i>When does Internet Security take place</i>
Locations	42	<i>All courses in room Mainbuilding H 1012</i>
584		

Table 5.1: The evaluation topics, the number of tagged test sentences and an example sentence for each topic in the known test set. It contains a total of 584 tagged sentences, consisting of 4009 word-tag tuples.

proportion of correctly tagged words can then be compared for both test sets for each language model.

In addition, the conformity of the language models with the labeled test data is measured by using Cohen's Kappa (κ)¹. The definition of Cohen's Kappa is given in equation 5.1.

$$\kappa = \frac{p_o - p_c}{1 - p_c} \quad (5.1)$$

p_o represents the measured agreement and p_c the randomly expected agreement. Most values are between the perfect agreement ($\kappa = 1$) and an agree-

¹ Cohen's Kappa is a measurement of the agreement of two raters referring to a given set of categories (in this thesis the set of tags). It was proposed by J. Cohen in 1960 [5] and considers the possibility of agreement by chance.

5 Evaluation and Comparison

TOPIC	SENTENCES	EXAMPLE
ECTS	44	<i>All modules with 6 ECTS</i>
Time	55	<i>Courses that take place on Tuesday</i>
Faculty	56	<i>Show all modules of faculty 4 to me</i>
Participants	54	<i>List all courses that have a limitation of 30 participants</i>
Persons	58	<i>Show all courses that are taught by Professor Martinelli</i>
Program	10	<i>Show all modules of computer science</i>
Modules	60	<i>Show information about the module Cognitive Algorithms</i>
Chair	20	<i>All modules that the institute Media Science offers</i>
Exam	45	<i>Show me all courses that have Group Lecture as examination</i>
Courses	30	<i>The date of the first meeting of Internet Security</i>
Locations	42	<i>All courses in auditorium Mainbuilding H 1012</i>
474		

Table 5.2: The evaluation topics, the number of tagged test sentences and an example sentence for each topic in the unknown test set. It contains a total of 474 tagged sentences, consisting of 3669 word-tag tuples.

ment only by chance ($\kappa = 0$). Values lower than zero are interpreted as an agreement that is even worse than an agreement by chance. There are different interpretations of the level of agreement for the different values of κ . Table 5.3 presents an interpretation based on a proposal of J. Landis et. al. [10] is used in this thesis.

5.2 Evaluation Results

After introducing the test sets and the evaluation methodology, all training groups were evaluated according to the parameter configuration presented in chapter 4.2. The following sections show the evaluation results of the different architectures for each test set.

5 Evaluation and Comparison

COHEN'S KAPPA	INTERPRETATION
$\kappa \leq 0.2$	Poor agreement
$0.2 < \kappa \leq 0.4$	Fair agreement
$0.4 < \kappa \leq 0.6$	Moderate agreement
$0.6 < \kappa \leq 0.8$	Good agreement
$0.8 < \kappa \leq 1.0$	Very good agreement

Table 5.3: The interpretation of Cohen's Kappa for different levels of agreement of two raters.

5.2.1 Feed-Forward Neural Network Models

...

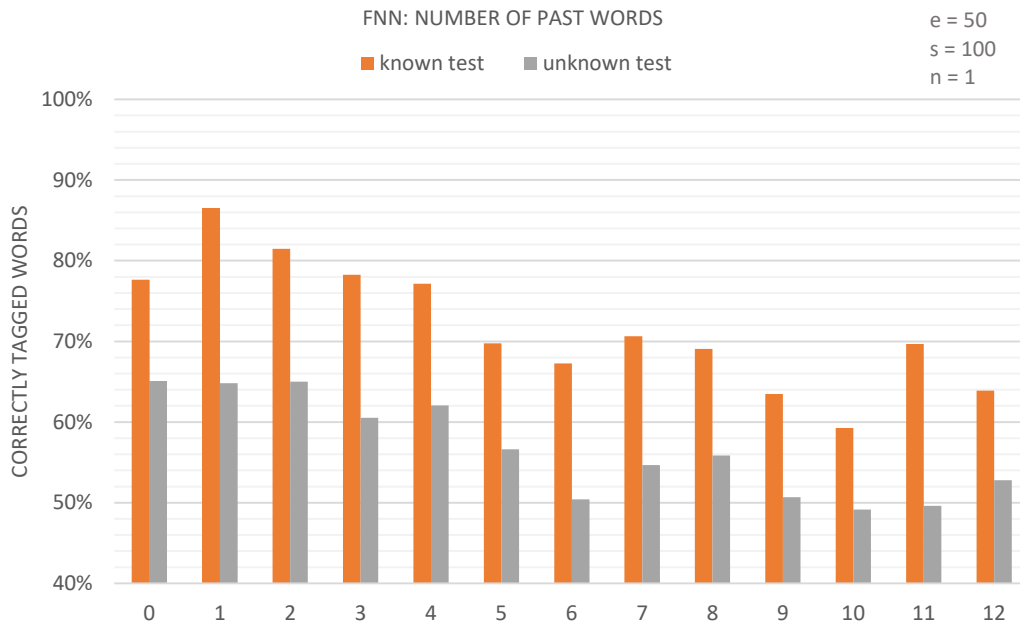


Figure 5.1: The evaluation results of the FNN for parameter p : the number of preceding words. Number of correctly tagged words in percent.

5 Evaluation and Comparison

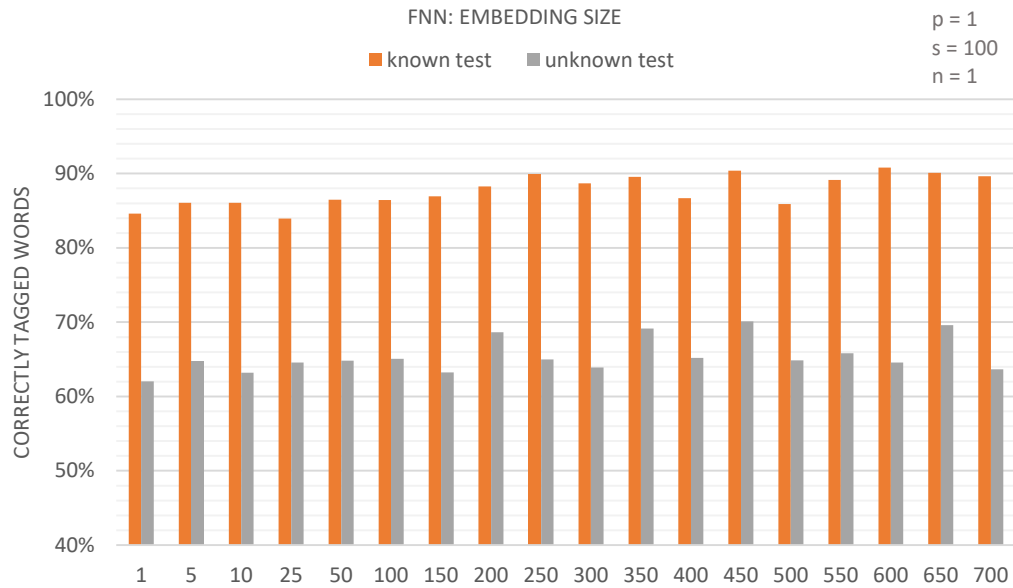


Figure 5.2: The evaluation results of the FNN for parameter e : the embedding size. Number of correctly tagged words in percent.

5 Evaluation and Comparison

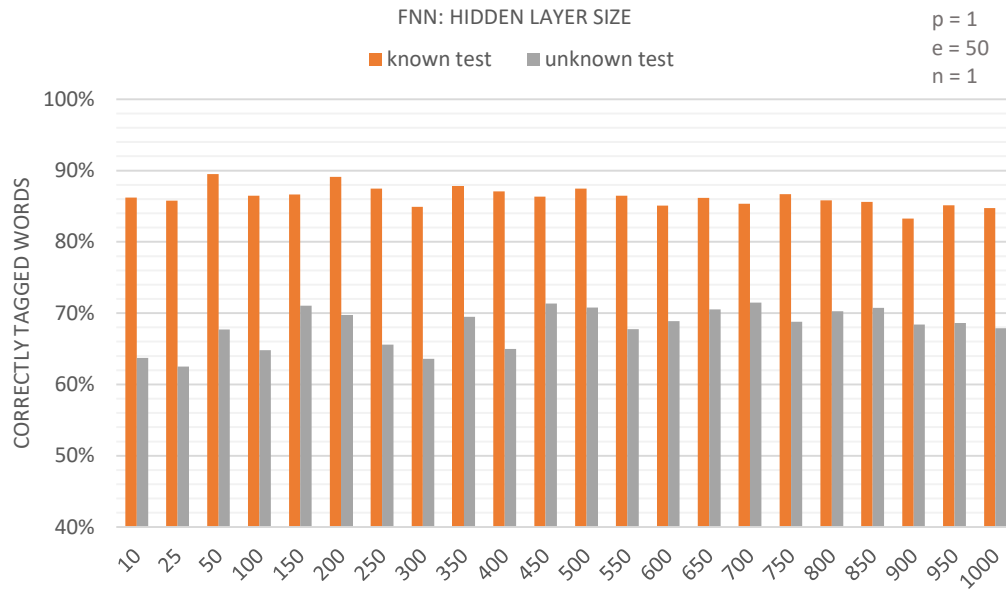


Figure 5.3: The evaluation results of the FNN for parameter s : the size of the hidden layer. Number of correctly tagged words in percent.

5 Evaluation and Comparison

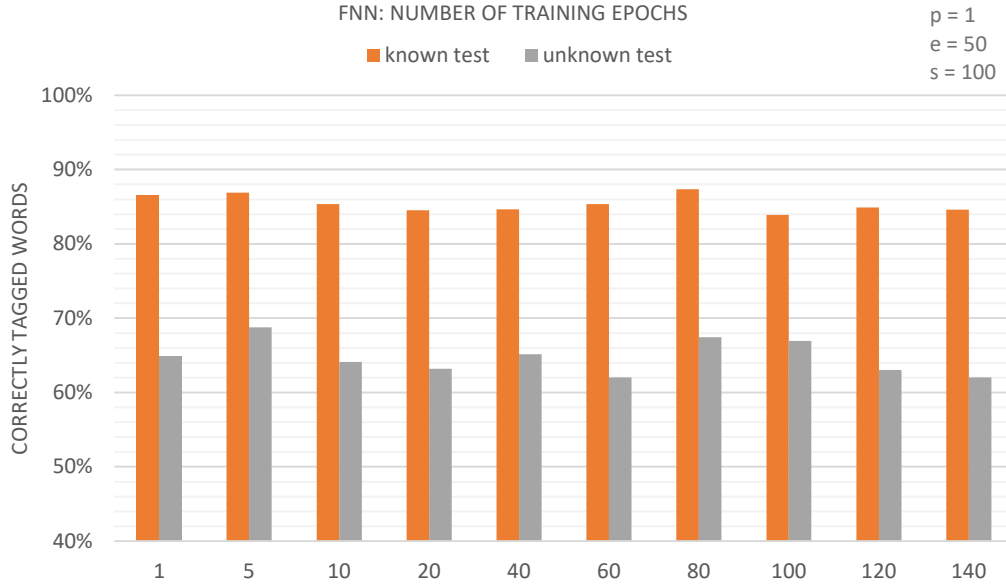


Figure 5.4: The evaluation results of the FNN for parameter n : the number of training epochs. Number of correctly tagged words in percent.

5.2.2 Recurrent Neural Network Models

...

5.2.3 Hidden Markov Models

After evaluating the neural network based language models, the HMM is evaluated in the following. For this purpose, the previous HMM which was trained on the basis of the training data generated by T. Michael [14] is compared to a new HMM trained on the basis of the training data proposed in this thesis. The evaluation was carried out using the known and the unknown test; the results are presented in figure 5.5.

5 Evaluation and Comparison

HMM: Training Set

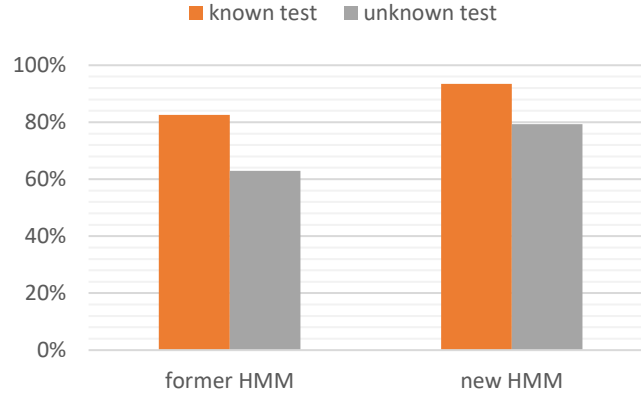


Figure 5.5: The evaluation results of the previous HMM trained by T. Michael [14] and the new HMM based on the new training data proposed in this thesis.

The previous HMM tagged 82.5% of the words of the known test and 63.0% of the unknown test correctly, whereas it was 93.4% for the known and 79.3% for the unknown test for the new HMM. It was evident that the HMM, which was based on the new training data, performed significantly better than the previous HMM. Especially the result for the unknown test is remarkable, as the accuracy increases by more than 16%.

It should be noted that the comparison of the previous and the new HMM model based on the known test has some limitations, as the known test was not fully known² to the previous HMM. Nevertheless, the major part of the known test set was based on training templates that were already used to build the corpus for the previous HMM, which is why it still scored well with more than 4 correct tags out of 5.

² As described in chapter 5.1, the known test contains 584 sentences. 375 sentences are included in the corpus of the previous HMM, resulting in 35.8% of the sentences of the known test, that were completely unknown to the previous HMM.

5.3 Overall Comparison

...

6 Discussion and Conclusion

The final chapter summarizes the work of this thesis and the evaluation results achieved. It discusses the evaluation findings and reveals potential areas for further research based on the knowledge gained in this thesis.

6.1 Summary

The aim to develop a neural network based part-of-speech tagger for the advisory Artificial Conversational Agent ALEX was accomplished within the scope of this thesis.

A tagger module was implemented featuring a feed-forward neural network as well as a recurrent neural network. A training corpus containing tagged sentences was created with the help of an improved set of sentence templates. With both neural network architectures, various language models were trained on this corpus using parameter variation. These models were then evaluated with two corresponding tests sets, containing sentences from the training corpus (known data) and unknown sentences.

6.2 Discussion

...

6.3 Future work

Although a high accuracy of the language models was achieved with an improved training corpus and the neural network approaches used in this thesis,

6 Discussion and Conclusion

further research could be done in certain areas to fine-tune and improve the templates as well as the tagging results.

As demonstrated in the preceding chapters, the methodology used in this thesis was found to be appropriate for an initial exploration into improving the tagging results. Going forward, however, there are certain areas and components of the methodology that could be optimized, which will be outlined in the following.

Rather than manually deleting training templates that contain a lot of useless combinations of data from the database, an additional module could check these combinations automatically in advance. Depending on the query result of such a data validation module, the generated training sentence would then be included or excluded from the training corpus. This way, the general sentence templates can still be used knowing that meaningless training sentences will not be considered.

In this thesis, parameters that were thought to influence the tagging results, such as the number of previous words, the size of the hidden layer, the size of the word embeddings and the number of training epochs were analyzed. Yet there are additional parameters, which could also be examined to improve the accuracy further. For the FNN, parameters of possible interest are the number of subsequent words (in addition to the number of previous words), the number of hidden layers, the type of the activation function for the neurons and the training optimizer. The use of word embeddings with a particular embedding size for the RNN could be interesting too, as well as the type of the activation function and the optimizer, which are possible additional parameters. Moreover, an exponential decay for the learning rate of the optimizer, especially for the RNN, could be explored.

Another potential way to improve the tagging accuracy, could be to use pretrained instead of randomly initialized word embeddings in the training process. This is a relevant direction to explore, as it was previously shown that word embeddings that were trained on large German text corpora contain a high level of syntactical and semantical information [16].

6 *Discussion and Conclusion*

Furthermore, to improve the evaluation process, it could be extended with a more fine-grained output concerning semantical topics such as degrees, locations, persons or module titles. This way, a statement could be made about how well the model performs on each topic and if there are significant differences between the evaluation results. This knowledge could then be used to systematically improve the training templates.

All in all, this thesis has demonstrated several ways of improving the tagger, which is not only of importance for the scope of this thesis and ALEX but which can also be used as a starting point for improving other chatbots, or systems that use a POS tagger in general.

Bibliography

- [1] J. Baker. The DRAGON system—An overview. *IEEE Transactions on Acoustics, Speech, and Signal Processing*, 23(1):24–29, February 1975.
- [2] Leonard E. Baum. An Equality with Applications to Statistical Prediction for Functions of Markov Process and to a Model to Ecology. *Bulletin of the American Mathematical Society*, 73:360–363, 1967.
- [3] Leonard E. Baum and Ted Petrie. Statistical Inference for Probabilistic Functions of Finite State Markov Chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563, 1966.
- [4] M.J. Bishop and E.A. Thompson. Maximum likelihood alignment of DNA sequences. *Journal of Molecular Biology*, 190(2):159 – 165, 1986.
- [5] Jacob Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.
- [6] Daniel Crevier. AI: The Tumultuous Search for Artificial Intelligence. 1993.
- [7] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.
- [8] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.
- [9] Julian Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech & Language*, 6(3):225 – 242, 1992.
- [10] J. Richard Landis and Gary G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1):159–174, 1977.
- [11] Qing Ma, Kiyotaka Uchimoto, Masaki Murata, and Hitoshi Isahara. Elastic Neural Networks for Part of Speech Tagging. 5, January 2000.
- [12] W. S. McCulloch. The brain computing machine. *Electrical Engineering*, 68(6):492–497, June 1949.
- [13] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics*, 5(4):115–133, December 1943.
- [14] Thilo Michael. Design and Implementation of an Advisory Artificial Conversational Agent, October 2016.

Bibliography

- [15] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.
- [16] Andreas Müller. Analyse von Wort-Vektoren deutscher Textkorpora, June 2015.
- [17] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, pages 65–386, 1958.
- [18] A. L. Samuel. Some Studies in Machine Learning Using the Game of Checkers. *IBM Journal of Research and Development*, 3(3):210–229, July 1959.
- [19] Helmut Schmid. Part-of-Speech Tagging with Neural Networks. October 1994.
- [20] G. L. Shaw. Donald Hebb: The Organization of Behavior. In Günther Palm and Ad Aertsen, editors, *Brain Theory*, pages 231–233, Berlin, Heidelberg, 1986. Springer Berlin Heidelberg.
- [21] R. L. Stratonovich. Conditional Markov Processes. *Theory of Probability & Its Applications*, 5(2):156–178, 1960.
- [22] B. Widrow. An adaptive “ADALINE” neuron using chemical “Memistors”. *Stanford Electronics Laboratories Technical Report*, 1553–2(2), 1960.

A Appendix

A.1 Set of sentence templates

Welche Module kann (ich|man) (im|in) (bachelor|master|diplom) {Program:name} belegen
R_LIST M_MTSModule X X X C_Program:degree C_Program:name X

Welche Module kann (ich|man) (im|in) {Program:name} (bachelor|master|diplom) belegen
R_LIST M_MTSModule X X X C_Program:name C_Program:degree X

Alle Module (im|in) (bachelor|master|diplom) {Program:name}
R_LIST M_MTSModule X C_Program:degree C_Program:name

Alle Module (im|in) {Program:name} (bachelor|master|diplom)
R_LIST M_MTSModule X C_Program:name C_Program:degree

Alle Module (vom|von|in) {Program:name}
R_LIST M_MTSModule X C_Program:name

Alle Module (vom|von|in|im) {Program:degree} {Program:name}
R_LIST M_MTSModule X C_Program:degree C_Program:name

Welche Module gehören zum (bachelor|master|diplom) {Program:name}
R_LIST M_MTSModule X X C_Program:degree C_Program:name

Welche Module gehören zum {Program:name} (bachelor|master|diplom)
R_LIST M_MTSModule X X C_Program:name C_Program:degree

Zeige mir alle Module im Wahlpflichtbereich {Program:name} (bachelor|master|diplom)
R_LIST X X M_MTSModule X C_CourseRegulation:group C_Program:name C_Program:degree

Zeige mir alle Module im Wahlpflichtbereich (bachelor|master|diplom) {Program:name}
R_LIST X X M_MTSModule X C_CourseRegulation:group C_Program:name C_Program:degree

Welche Module gibt es (im|in) {Program:name}
R_LIST M_MTSModule X X X C_Program:name

Welche Module gibt es im Studiengang {Program:name}
R_LIST M_MTSModule X X X X_Program:name C_Program:name

Welche Module mit dem Abschluss {Program:degree} gibt es
R_LIST M_MTSModule X X X_Program:degree C_Program:degree X X

Ich suche alle Module (vom|von) {Program:name}
X X R_LIST M_MTSModule X C_Program:name

Ich suche alle Module (vom|von|im) {Program:degree} {Program:name}
X X R_LIST M_MTSModule X C_Program:degree C_Program:name

Module mit dem Namen {MTSModule:title}
M_MTSModule X X X_MTSModule:title C_MTSModule:title

Welche Module kann (ich|man) im {Program:name} {CourseRegulation:group} belegen
R_LIST M_MTSModule X X X C_Program:name C_CourseRegulation:group X

Welche Veranstaltungen im {Program:degree} {CourseRegulation:group} gibt es

A Appendix

R_LIST M_Course X C_Program:degree C_CourseRegulation:group X X

Welche Module haben (0|1|2|3|4|5|6|7|8|9|10|11|12) ects
R_LIST M_MTSModule X C_MTSModule:ects C_MTSModule:ects

Welche Module haben mehr als (0|1|2|3|4|5|6|7|8|9|10|11|12) ects
R_LIST M_MTSModule X Q_GT X C_MTSModule:ects C_MTSModule:ects

Welche Module haben weniger als (2|3|4|5|6|7|8|9|10|11|12) ects
R_LIST M_MTSModule X Q_LT X C_MTSModule:ects C_MTSModule:ects

Welche Module haben genau (0|1|2|3|4|5|6|7|8|9|10|11|12) ects
R_LIST M_MTSModule X X C_MTSModule:ects C_MTSModule:ects

Welche Veranstaltungen finden (Mo|Di|Mi|Do|Fr|Sa) statt
R_LIST M_Course X C_CourseDate:day X

Welche Veranstaltungen finden am (Mo|Di|Mi|Do|Fr|Sa) statt
R_LIST M_Course X X C_CourseDate:day X

Welche Veranstaltungen finden am (Mo|Di|Mi|Do|Fr|Sa) nach (7|8|9|10|11|12|13|14|15|16|17|18|19|20) Uhr statt
R_LIST M_Course X X C_CourseDate:day Q_GT C_CourseDate:startTime C_CourseDate:startTime X

Welche Veranstaltungen finden am (Mo|Di|Mi|Do|Fr|Sa) vor (8|9|10|11|12|13|14|15|16|17|18|19) Uhr statt
R_LIST M_Course X X C_CourseDate:day Q_LT C_CourseDate:startTime C_CourseDate:startTime X

Welche Veranstaltungen finden am (Mo|Di|Mi|Do|Fr|Sa) um (8|9|10|11|12|13|14|15|16|17|18|19) Uhr statt
R_LIST M_Course X X C_CourseDate:day X_CourseDate:startTime C_CourseDate:startTime
C_CourseDate:startTime X

Welche Veranstaltungen sind (Mo|Di|Mi|Do|Fr|Sa)
R_LIST M_Course X C_CourseDate:day

Welche Veranstaltungen sind am (Mo|Di|Mi|Do|Fr|Sa)
R_LIST M_Course X X C_CourseDate:day

Welche Veranstaltungen sind am (Mo|Di|Mi|Do|Fr|Sa) nach (7|8|9|10|11|12|13|14|15|16|17|18|19|20) Uhr
R_LIST M_Course X X C_CourseDate:day Q_GT C_CourseDate:startTime C_CourseDate:startTime

Welche Veranstaltungen sind am (Mo|Di|Mi|Do|Fr|Sa) vor (8|9|10|11|12|13|14|15|16|17|18|19) Uhr
R_LIST M_Course X X C_CourseDate:day Q_LT C_CourseDate:startTime C_CourseDate:startTime

Welche Veranstaltungen sind am (Mo|Di|Mi|Do|Fr|Sa) um (8|9|10|11|12|13|14|15|16|17|18|19) Uhr
R_LIST M_Course X X C_CourseDate:day X_CourseDate:startTime C_CourseDate:startTime
C_CourseDate:startTime

Welche Veranstaltung hält {Person:lastname} am (Mo|Di|Mi|Do|Fr|Sa)
R_LIST M_Course X C_Person:fullname X_CourseDate:day C_CourseDate:day

Welche Module mit mehr als (0|1|2|3|4|5|6|7|8|9|10|11|12) ects kann (ich|man) im {
CourseRegulation:group} belegen
R_LIST M_MTSModule X Q_GT X C_MTSModule:ects C_MTSModule:ects X X X C_CourseRegulation:group
X

Welche Module werden von Professor {Person:lastname} (unterrichtet|angeboten|gehalten)
R_LIST M_MTSModule X X X_Person C_Person:fullname X

A Appendix

Wer ist der Modulverantwortliche des Moduls {MTSMODULE:title}
M_PERSON X X X_PERSON X X_MTSMODULE:title C_MTSMODULE:title

Wer ist verantwortlich für das Moduls {MTSMODULE:title}
M_PERSON X X X X_X_MTSMODULE:title C_MTSMODULE:title

Bei wem findet das Moduls {MTSMODULE:title} statt
X M_PERSON X X X_X_MTSMODULE:title C_MTSMODULE:title X

Welche Kurse werden von Professor {PERSON:lastname} (unterrichtet|angeboten|gehalten)
R_LIST M_COURSE X X X_PERSON C_PERSON:fullname X

Welche Kurse (unterrichtet|bietet|hält) Professor {PERSON:lastname}
R_LIST M_COURSE X X_PERSON C_PERSON:fullname

Wie viele ects (hat|bringt) das Modul {MTSMODULE:title}
R_SINGLE X_count R_MTSMODULE:ects X X C_MTSMODULE:title C_MTSMODULE:title

(Wieviele|Wieviel) ects (hat|bringt) das Modul {MTSMODULE:title}
X_count R_MTSMODULE:ects X X C_MTSMODULE:title C_MTSMODULE:title

(Informationen|Details|Mehr) (zu|zum) Modul {MTSMODULE:title}
R_SINGLE X X_MTSMODULE:title C_MTSMODULE:title

Welche Modulkataloge gibt es (im|in) {PROGRAM:degree} {PROGRAM:name}
R_LIST M_COURSEREGULATION X X X_C_PROGRAM:degree C_PROGRAM:name

Zeige den Modulkatalog (im|in) {PROGRAM:degree} {PROGRAM:name}
R_LIST X M_COURSEREGULATION X C_PROGRAM:degree C_PROGRAM:name

Welche Module werden vom Fachgebiet {CHAIR:name} angeboten
R_LIST M_MTSMODULE X X X_CHAIR:name C_CHAIR:name X

Module vom Fachgebiet {CHAIR:name}
M_MTSMODULE X X_CHAIR:name C_CHAIR:name

Alle Module vom Fachgebiet {CHAIR:name}
R_LIST M_MTSMODULE X X_CHAIR:name C_CHAIR:name

Zeige mir alle Module vom Fachgebiet {CHAIR:name}
R_LIST X X M_MTSMODULE X X_CHAIR:name C_CHAIR:name

Welche Studiengänge von der Fakultät (1|2|3|4|5|6|7) gibt es
R_LIST M_PROGRAM X X X_INSTITUTE:faculty C_INSTITUTE:faculty X X

Module von der Fakultät (1|2|3|4|5|6|7)
M_MTSMODULE X X X_INSTITUTE:faculty C_INSTITUTE:faculty

Alle Module von der Fakultät (1|2|3|4|5|6|7)
R_LIST M_MTSMODULE X X X_INSTITUTE:faculty C_INSTITUTE:faculty

Zeige mir alle Module von der Fakultät (1|2|3|4|5|6|7)
R_LIST X X M_MTSMODULE X X X_INSTITUTE:faculty C_INSTITUTE:faculty

Welche Studiengänge von Fakultät (1|2|3|4|5|6|7) gibt es
R_LIST M_PROGRAM X X X_INSTITUTE:faculty C_INSTITUTE:faculty X X

Module von Fakultät (1|2|3|4|5|6|7)
M_MTSMODULE X X X_INSTITUTE:faculty C_INSTITUTE:faculty

Alle Module von Fakultät (1|2|3|4|5|6|7)
R_LIST M_MTSMODULE X X X_INSTITUTE:faculty C_INSTITUTE:faculty

A Appendix

Zeige mir alle Module von Fakultät (1|2|3|4|5|6|7)
R_LIST X X M_MTSModule X X_Institute:faculty C_Institute:faculty

Veranstaltungen mit {ExamElement:description} als Prüfung
M_Course X C_ExamElement:description X X_ExamElement

Welche Veranstaltungen haben die Prüfung {ExamElement:description}
R_LIST M_Course X X X_ExamElement C_ExamElement:description

Kurse die am (Mo|Di|Mi|Do|Fr|Sa) angeboten werden
M_Course X X C_CourseDate:day X X

Welche Veranstaltungen werden an einem (Mo|Di|Mi|Do|Fr|Sa) angeboten
R_LIST M_Course X X X C_CourseDate:day X

Wann ist das erste Treffen von {Course:title}
M_CourseDate X X R_FIRST X X C_Course:title

Wann ist die erste Veranstaltung von {Course:title}
M_CourseDate X X R_FIRST X X C_Course:title

Wann findet {Course:title} statt
M_CourseDate X C_Course:title X

In welchen Studiengängen gibt es das Modul {MTSModule:title}
X R_LIST M_Program X X X_MTSModule:title C_MTSModule:title

Welche Veranstaltungen des Fachgebiets {Chair:name} gibt es
R_LIST M_Course X X_Chair:name C_Chair:name X X

Module mit dem Titel {MTSModule:title}
M_MTSModule X X X_MTSModule:title C_MTSModule:title

Welche Veranstaltungen finden (im|in) Raum {CourseDate:room} statt
R_LIST M_CourseDate X X X_Room C_CourseDate:room X

Alle Veranstaltungen (im|in) Raum {CourseDate:room}
R_LIST M_CourseDate X X_Room C_CourseDate:room

Zeige mir alle Veranstaltungen (im|in) Raum {CourseDate:room}
X X R_LIST M_CourseDate X X_Room C_CourseDate:room

Welche Veranstaltungen werden {CourseDate:cycle} angeboten
R_LIST M_Course X C_CourseDate:cycle X

Welche Module haben eine Platzbeschränkung von mehr als (1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|
16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|40|45|50|55|60|65|70|75|80|85|
90|95|100|200|300) Teilnehmern
R_LIST M_MTSModule X X X_ParticipantLimitation X Q_GT X C_MTSModule:participantLimitation
X_ParticipantLimitation

Welche Module haben eine Platzbeschränkung von weniger als (1|2|3|4|5|6|7|8|9|10|11|12|13|14|
15|16|17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|40|45|50|55|60|65|70|75|80|
85|90|95|100|200|300) Teilnehmern
R_LIST M_MTSModule X X X_ParticipantLimitation X Q_LT X C_MTSModule:participantLimitation
X_ParticipantLimitation

Welche Module haben eine Platzbeschränkung von genau (1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|
17|18|19|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|40|45|50|55|60|65|70|75|80|85|90|
95|100|200|300) Teilnehmern

A Appendix

```
R_LIST M_MTSModule X X_X_ParticipantLimitation X Q_EQ C_MTSModule:participantLimitation _
X_ParticipantLimitation
```

```
Welche Module sind (beschränkt|begrenzt) auf (1|2|3|4|5|6|7|8|9|10|11|12|13|14|15|16|17|18|19|
|20|21|22|23|24|25|26|27|28|29|30|31|32|33|34|35|40|45|50|55|60|65|70|75|80|85|90|95|100|
200|300) (Teilnehmer|Studenten|Personen)
```

```
R_LIST M_MTSModule X X_X_ParticipantLimitation X C_MTSModule:participantLimitation _
X_ParticipantLimitation
```

```
Welche Personen bieten Module mit mehr als (1|2|3|4|5|6|7|8|9|10|11|12|15|30) ects an
R_LIST M_Person X R_MTSModule:title X Q_GT X C_MTSModule:ects C_MTSModule:ects X
```

```
Bitte nur welche die (ich|man) im Studiengang {Program:name} studieren kann
X_PLEASE X_ONLY X X X X_X_Program:name C_Program:name X X
```

```
Welche Kurse des Moduls {MTSModule:title} kann (ich|man) (Mo|Di|Mi|Do|Fr|Sa) belegen
R_LIST M_Course X X_MTSModule:title C_MTSModule:title X X C_CourseDate:day X
```

```
#####
#                               Wörter ohne Kontext                               #
#####
```

```
{MTSModule:title}
C_MTSModule:title
```

```
im master [100]
X C_Program:degree
```

```
im bachelor [100]
X C_Program:degree
```

```
nur [30]
X_ONLY
```

```
bitte [30]
X_PLEASE
```

```
hilfe [30]
X_HELP
```

```
hallo [30]
X_GREETING
```

```
Guten (Tag|Morgen|Abend|Nacht|Nachmittag) [5]
X_GOOD X_GREETING
```

```
Ich brauche hilfe [10]
X X_X_HELP
```

```
scheiße [30]
X_CURSE
```

```
ja [30]
X_YES
```

```
nein [30]
X_NO
```

```
zurück [30]
X_BACK
```

```
Leben
```

A Appendix

X_HITCH_LIFE

Universum

X_HITCH_UNIVERSE

Rest

X_HITCH_EVERYTHING

```
#####  
#                               Personal stuff                               #  
#####
```

(Was|Wie) ist dein Alter

X X X_PERSONAL X_AGE

(Was|Wie) ist dein Name

X X X_PERSONAL X_NAME

Wie heißt du

X X_NAME X_PERSONAL

Was bist du von Beruf

X X X_PERSONAL X_X_PROFESSION

Was ist dein (Beruf|Job)

X X X_PERSONAL X_PROFESSION

Was ist dein Auftrag

X X X_PERSONAL X_MISSION