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	Prof. 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 16. Mai 2018

Unterschrift

Abstract

The advisory artificial conversational agent ALEX was developed to answer questions about modules and courses at the Technische Universität Berlin. It utilizes part-of-speech tagging to process natural language and convert it into a database query. Therefore, 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 generated training templates that are filled with the data of 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, causing inaccurate tagging results and incorrect database queries on input sentences with an unknown structure or unknown words.

To overcome this challenge, this thesis presents two different neural network approaches for language modeling to assign POS tags to the input sentences for a chatbot like ALEX. A training corpus was generated based on training sentence templates, which were extended and improved using real user data. For the evaluation of the accuracy of the language models, two test sets were created containing known data (sentences from the training corpus) and unknown data.

It was shown, that the new training corpus proposed in this thesis significantly improves the HMM utilized by ALEX and that a neural network based tagger is able to outperform the HMM tagger.

This thesis thus demonstrates ways of improving tagging results and natural language output at the example of ALEX, which can be used for similar application areas in the field of NLP.

Zusammenfassung

Der Artificial Conversational Agent ALEX wurde entwickelt, um Fragen zu Modulen und Lehrveranstaltungen an der Technischen Universität Berlin zu beantworten. Es verwendet Part-of-speech Tagging, um natürliche Sprache zu verarbeiten und sie in eine Datenbankabfrage zu konvertieren. 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 führen zu vielen Trainingsdatensätzen mit gleicher Struktur, was zu ungenauen Tagging-Ergebnissen und inkorrekten Datenbankabfragen bei Eingabesätzen mit unbekannter Struktur oder unbekannten Wörtern führte.

Diese Arbeit stellt zwei verschiedene Ansätze neuronaler Netzwerke für die Sprachmodellierung der Eingabesätze für einen Chatbot wie ALEX vor. Basierend auf Trainingssatzvorlagen, die mit realen Benutzerdaten erweitert und verbessert wurden, wurde ein neuer Trainingskorpus erstellt. Zur Beurteilung der Genauigkeit der Sprachmodelle wurden zwei Tests entwickelt, die bekannte Daten (Sätze aus dem Trainingskorpus) und unbekannte Daten enthalten.

Es wurde gezeigt, dass der in dieser Arbeit vorgeschlagene neue Trainingskorpus das von ALEX verwendete HMM signifikant verbessert und dass ein auf neuronalen Netzwerken basierender Tagger in der Lage ist, den HMM Tagger bei bekannten Daten zu übertreffen.

Diese Arbeit demonstriert Verbesserungen eines Part-of-Speech Taggers am Beispiel von ALEX welche für ähnliche Anwendungsbereiche der natürlichen Sprachverarbeitung eingesetzt werden können.

Contents

List of Figures

List of Tables

Abbreviations

ACA	*Artificial Conversational Agent*
ADALINE	*Adaptive Linear Neuron*
ANN	*Artificial Neural Network*
AN	*Artificial Neuron*
ELU	*Exponential Linear Unit*
FNN	*Feed-forward Neural Network*
HMM	*Hidden Markov Model*
LSTM	*Long Short-Term Memory*
ML	*Machine Learning*
NLP	*Natural Language Processing*
NLTK	*Natural Language Toolkit*
POS	*Part-of-Speech*
RELU6	*Rectified Linear Unit with maximum 6*
RELU	*Rectified Linear Unit*
RNN	*Recurrent Neural Network*
SELU	*Scaled Exponential Linear Unit*
SGD	*Stochastic Gradient Descent*
TF	*TensorFlow*

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. Samuel[1] [23] 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 [16], 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

[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 [23].

natural language 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. It utilizes a statistical method (the *hidden Markov model*) to assign special parts of speech to the words of the input sentence (part-of-speech tagging). This enables ALEX to extract information that is necessary to create a database query and return the corresponding answer for the users question.

1.1 Scope of this Thesis

In general, this thesis aims to improve how ALEX the advisory Artificial Conversational Agent, understands and learns natural language using *artificial neural networks* (ANNs) for the task of assigning parts of speech to words. Therefore, this thesis focuses on developing a neural network based part-of-speech tagger for ALEX, the creation of a corpus of training data, 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 purpose of training both neural network implementations, an important part of this thesis is the generation of an improved training corpus of tagged language data 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 data[2] and unknown data[3] is created. Both neural network models and the HMM are compared to each other using this evaluation.

According to the evaluation results, the new tagger, which is proposed in this thesis, should provide the possibility to replace the initial HMM based

[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

part-of-speech 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.

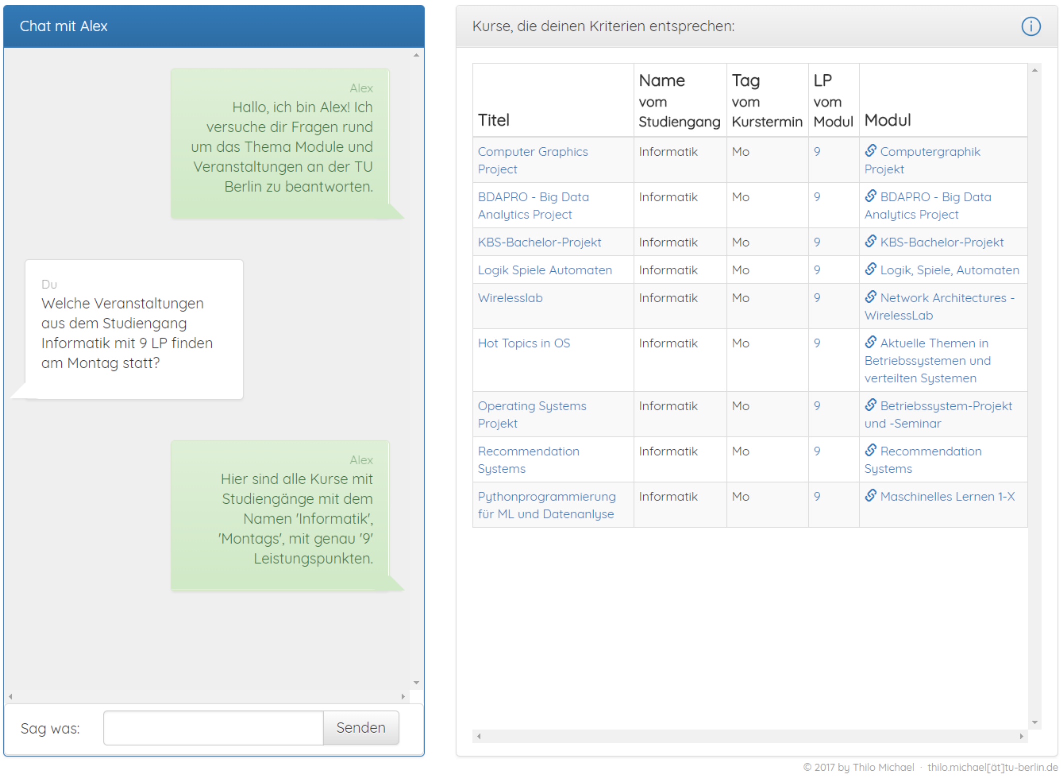
1.2 Related Work

This thesis is built upon the work of T. Michael [18], 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.

ALEX consists of several processing modules:

- 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.



Alex
Hallo, ich bin Alex! Ich versuche dir Fragen rund um das Thema Module und Veranstaltungen an der TU Berlin zu beantworten.

Du
Welche Veranstaltungen aus dem Studiengang Informatik mit 9 LP finden am Montag statt?

Alex
Hier sind alle Kurse mit Studiengänge mit dem Namen 'Informatik', 'Montags', mit genau '9' Leistungspunkten.

Titel	Name vom Studiengang	Tag vom Kurstermin	LP vom Modul	Modul
Computer Graphics Project	Informatik	Mo	9	🔗 Computergraphik Projekt
BDAPRO - Big Data Analytics Project	Informatik	Mo	9	🔗 BDAPRO - Big Data Analytics Project
KBS-Bachelor-Projekt	Informatik	Mo	9	🔗 KBS-Bachelor-Projekt
Logik Spiele Automaten	Informatik	Mo	9	🔗 Logik, Spiele, Automaten
Wirelesslab	Informatik	Mo	9	🔗 Network Architectures - WirelessLab
Hot Topics in OS	Informatik	Mo	9	🔗 Aktuelle Themen in Betriebssystemen und verteilten Systemen
Operating Systems Projekt	Informatik	Mo	9	🔗 Betriebssystem-Projekt und -Seminar
Recommendation Systems	Informatik	Mo	9	🔗 Recommendation Systems
Pythonprogrammierung für ML und Datenanlyse	Informatik	Mo	9	🔗 Maschinelles Lernen 1-X

© 2017 by Thilo Michael · thilo.michael[at]tu-berlin.de

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.

The focus of this thesis lies on the tagging module. As outlined in Chapter 1.1, the main objective is to replace the hidden Markov model by artificial neural networks. Both approaches with their historical context are introduced in the following.

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 [26], 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) [5] and part-of-speech tagging [13].

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 [17] and D. Hebb [25] 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)*[8] in 1951.

In the late 1950s, F. Rosenblatt developed the *Mark I Perceptron* computer and published a theorem of convergence of the perceptron[22] 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 neuron[4]. 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 ADALINE[5] model of a neural network, for which input weights could already be adjusted by the learning algorithm [27].

A publication of M. Minsky and S. Papert [19] 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.

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 [24] 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 [15].

[4] Chapter 3.1 and Chapter 3.2 explain the architecture of different neural network structures in detail

[5] ADALINE is an acronym for Adaptive Linear Neuron

[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).

1.3 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.

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

In conclusion, Chapter 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 [18] 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 (Chapter 2.2), the hidden Markov model tagger (Chapter 2.3) and the tagging interface (Chapter 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

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

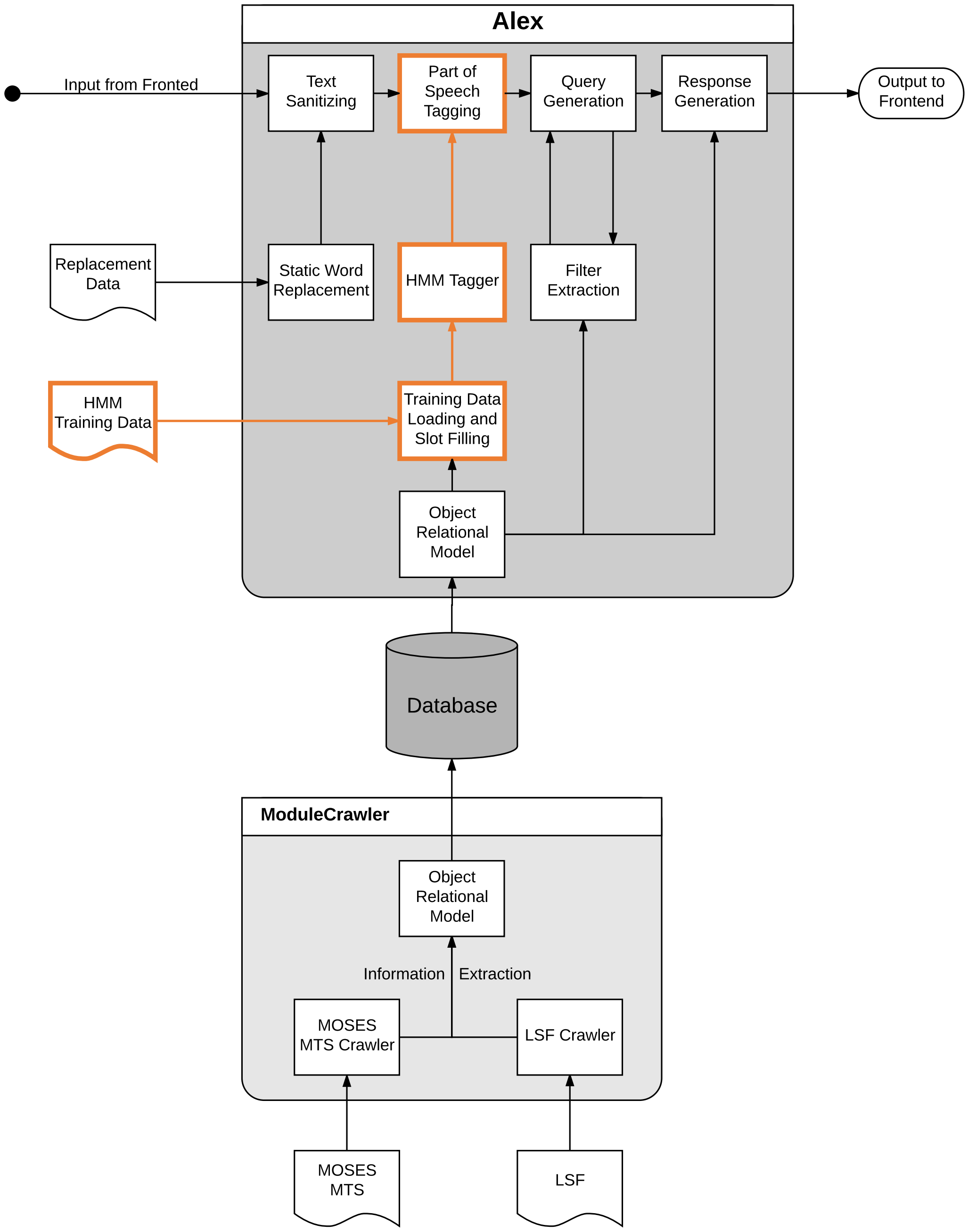


Figure 2.1: Overview of all components of Alex. The orange borders identify components that lie within the scope of this thesis and have been adapted or replaced. Original graphic by T. Michael [18]

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 sentence[1].

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 Chapter 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

[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* [18].

and determines as a result, which parts of speech most likely correspond to the words of the input sequence.

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.

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

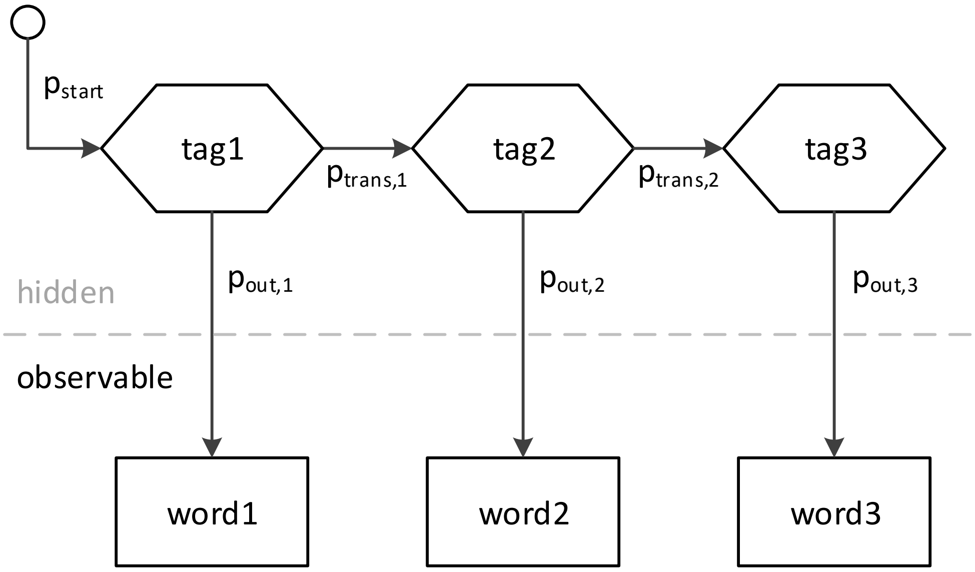


Figure 2.2: An example of a state sequence of three hidden states (tag1 – tag3) and an observed sequence of three words (word1 – word3) in a hidden Markov model. p_{start} denotes the starting probability, p_{trans} the transmission probabilities between hidden states and p_{out} the output probabilities 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 [18], 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 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.

	FORMATS	DESCRIPTION	EXAMPLE
R	*Return-tags*, describing data that is expected to be returned	R_LIST	*"**Which** modules…"*
		R_SINGLE	*"**Which** module…"*
		R_COUNT	*"**How many** modules…"*
M	*Model-tags*, describing the database model, e.g. M_MTSModule or M_Course	M_[MODEL]	*"Which **modules**…"*
			*"Which **courses**…"*
C	*Constraint-tags*, filtering the result set, given a database model and corresponding field, e.g. C_MTSModule:ects	C_[MODEL]:[FIELD]	*"Modules with **6** ects…"*
P	*Property-tags*, indicating to include fields in the result set, e.g. P_MTSModule:ects	P_[MODEL]:[FIELD]	*"Modules with 6 **ects**…"*
Q	*Comparison-tags*, describing an equal, greater than or less than constraint	Q_EQ	*"… with **exactly** 6 ects…"*
		Q_LT	*"… **less than** 6 ects…"*
		Q_GT	*"… **more than** 6 ects…"*
X	*Extra-tags*, describing words that are not relevant for the database query[2]	X	*"**and**", "**of**", "**is**"*
		X_[WORD]	*"I need **help**"*
		X_[MODEL]:[FIELD]	*"**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 [18] 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

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.

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 learning[1].

The network itself emerges from the interconnection of nodes (the artificial neurons), that are usually arranged in different layers. Each node represents a non-linear[2] 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

[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.

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* (SGD).

The POS tagging task requires the processing of words which are represented as character strings. For computational processing, 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

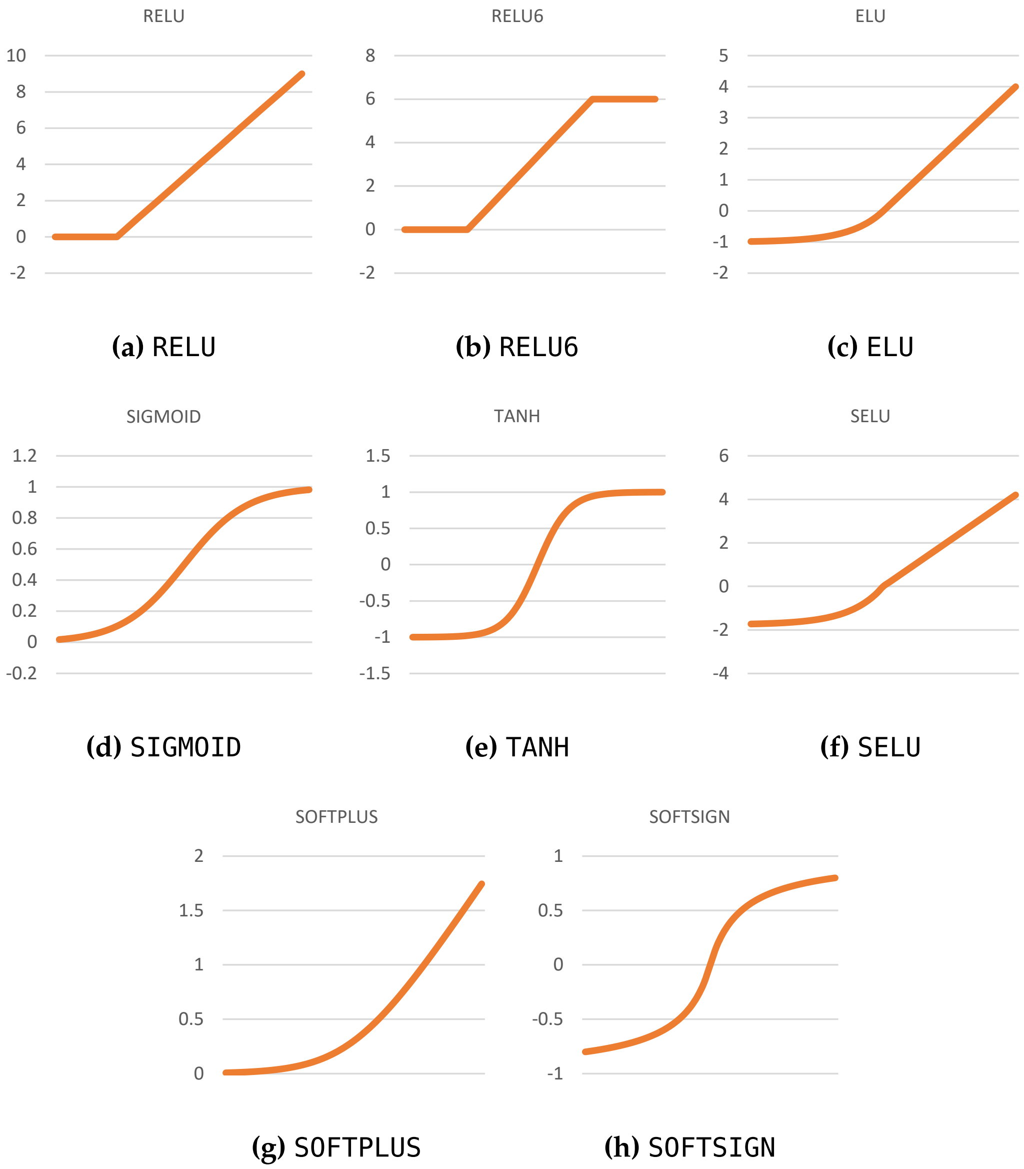
The FNN architecture that is used in this thesis is shown in Figure 3.2. The current word and the configured number of preceding words are each converted into word embeddings[3]. These embeddings are reshaped to one vector, called the feature vector, which serves as the input layer for the neural network. 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.

[3] Word embeddings are mathematical representations of words, usually presented as a vector of real numbers. While one-hot encoded word vectors have the size of the vocabulary, word embeddings maintain a dimensionality reduction to improve performance while maintaining feature information.

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.

The outputs of the hidden layer are typically computed with the rectified linear activation function (the *rectifier*). These nodes are therefore also called *rectified linear units* (RELUs). However, the effect of 8 different activation functions is examined in this thesis too. Figure 3.1 illustrates the qualitative plot of each function. The presented activation functions can be separated into two different types of nonlinearities: `RELU`[4], `RELU6` and `SELU`[5] are continuous functions but not everywhere differentiable, the other functions (`ELU`[6], `SIGMOID`[7], `TANH`, `SOFTPLUS`[8] and `SOFTSIGN`[9]) are continuously differentiable.

[4] *Rectified Linear Unit*, Nair et. al. [21]
[5] *Scaled Exponential Linear Units*, Klambauer et. al. [12]
[6] *Exponential Linear Units*, Clevert et. al. [6]
[7] Also known as Logistic function or SoftStep
[8] Glorot et. al. [9]
[9] Bergstra et. al. [4]



(a) RELU

(b) RELU6

(c) ELU

(d) SIGMOID

(e) TANH

(f) SELU

(g) SOFTPLUS

(h) SOFTSIGN

Figure 3.1: Qualitative illustration of 8 different activation functions that provide different types of nonlinearities. RELU, RELU6 and SELU are continuous but not everywhere differentiable functions, the other functions are continuously differentiable.

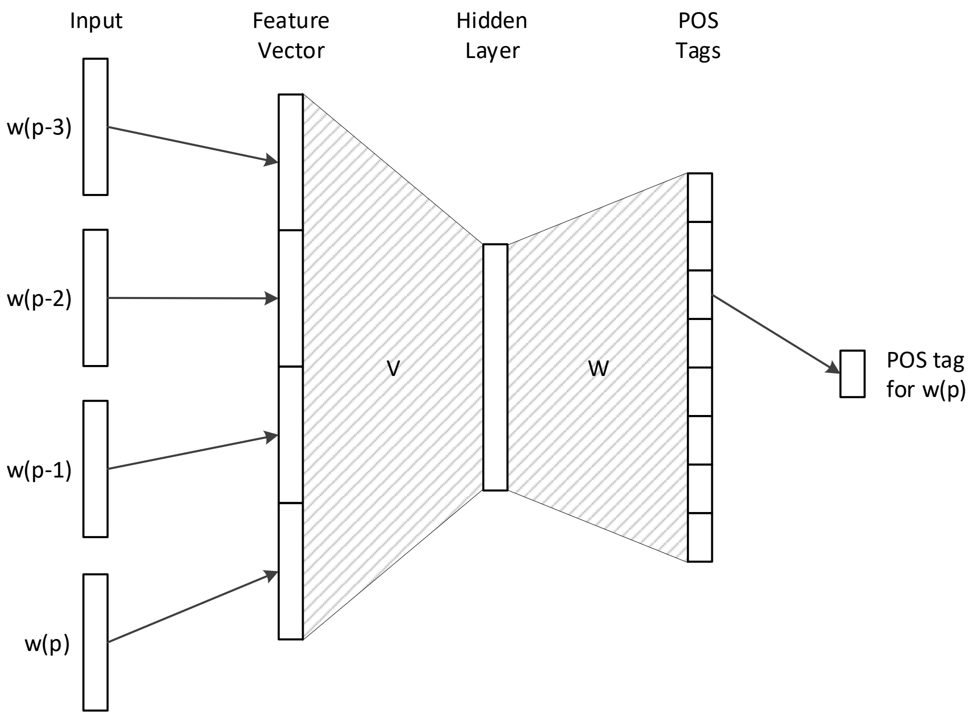


Figure 3.2: The structure of a feed-forward neural network. The feature vector is built by the embeddings of the corresponding input word at position p and by its 3 preceding words (the number of predecessors may vary, of course). V and W are weight matrices with respective matrix dimensions of feature vector size × hidden layer size for V and hidden layer size × number of POS tags for W.

3.1.2 Implementation

To implement the presented FNN architecture, the open source machine learning framework *TensorFlow*[10] 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 × embedding size is created and initially filled with random normal distributed

[10] 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`

values[11], 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 3.3 illustrates the creation of the feature vector.

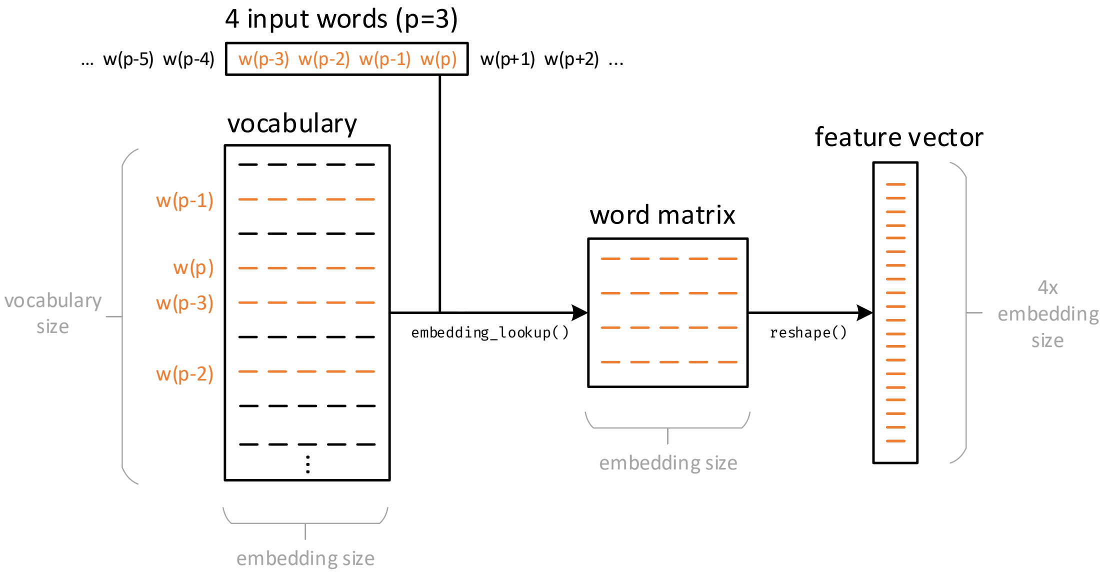


Figure 3.3: An example of the creation of a feature vector for the FNN using 3 preceding words and the current word.

The hidden layer nodes are also initialized with random normal distributed values. Their outputs are computed with one of the activation functions presented in Figure 3.1. Table 3.1 lists the activation functions with their respective computation formula and the corresponding TensorFlow function.

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.

[11] 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.

NAME	DEFINITION	TENSORFLOW
RELU	$a(x) = \max(x, 0)$	`tf.nn.relu()`
RELU6	$a(x) = \min(\max(x, 0), 6)$	`tf.nn.relu6()`
ELU	$f(x) = \begin{cases} e^x - 1, & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$	`tf.nn.elu()`
SIGMOID	$f(x) = \frac{1}{1+e^{-x}}$	`tf.nn.sigmoid()`
TANH	$f(x) = \tanh x$	`tf.nn.tanh()`
SELU	$f(x) = \lambda \begin{cases} \alpha(e^x - 1), & \text{if } x < 0 \\ x, & \text{otherwise} \end{cases}$	`tf.nn.selu()`
SOFTPLUS	$f(x) = \ln(e^x + 1)$	`tf.nn.softplus()`
SOFTSIGN	$f(x) = \frac{x}{\lvert x+1 \rvert}$	`tf.nn.softsign()`

Table 3.1: Definitions and TensorFlow Implementations of 8 different activation functions.

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()`[12] enables backpropagation with stochastic gradient descent to optimize the computations and results during training.

The implementation of the FNN POS tagger that was developed for this thesis is based on an existing TensorFlow POS tagger implementation by M. Rahtz[13].

[12] The Adam algorithm is an optimizer that was proposed by D. Kingma et. al. in 2014 [11]. It provides a computationally efficient gradient-based optimization of stochastic objective functions.

[13] The POS tagger is licensed under GNU GPL v3.0 and was retrieved on 15th January 2018. `https://github.com/mrahtz/tensorflow-pos-tagger`

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 3.4 illustrates the architecture of an RNN, which was implemented for this thesis. Like the FNN, the RNN uses word embeddings as input features. It contains one hidden layer, which is populated with the embedding of 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 3.4. The output layer represents the set of existing POS tags and is connected to the hidden layer with weight matrix W.

The word embedding of each word of every sentence in the training corpus represents the input feature vector, 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 weights and reduce the loss, the next training step is executed, taking the current training step into account.

The outputs of the hidden layer are calculated utilizing the same activation functions that were used for the FNN (see Figure 3.1).

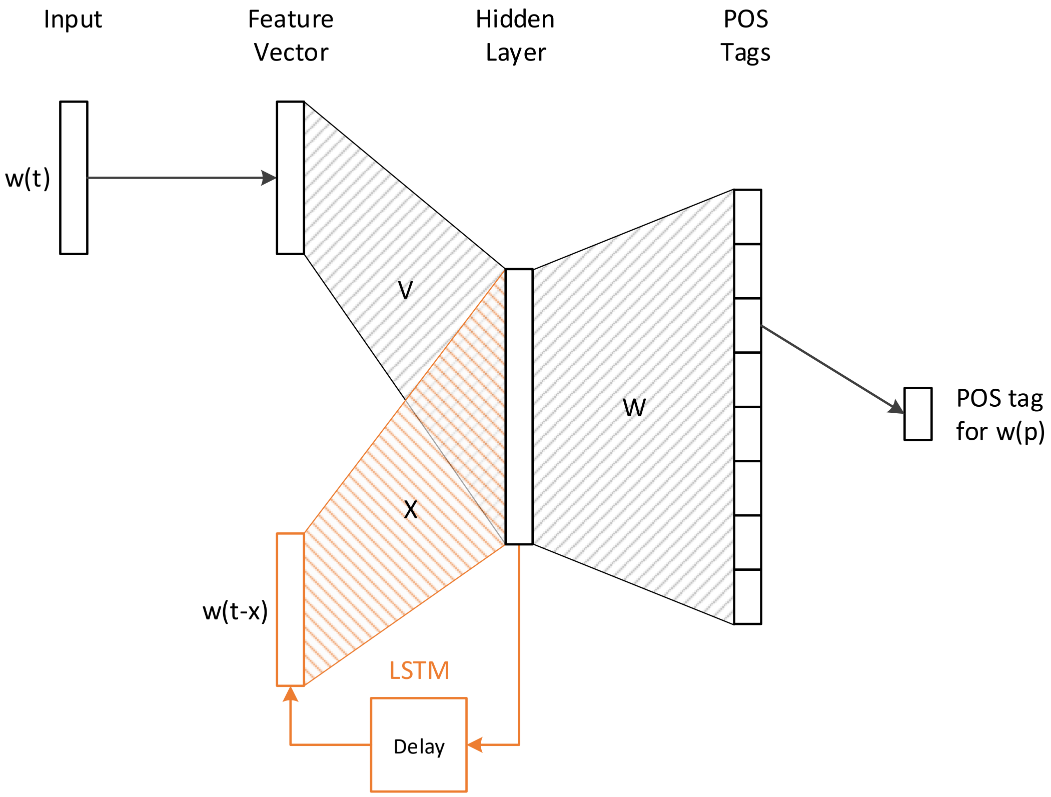


Figure 3.4: The structure of a recurrent neural network. The feature vector is the initial vector of the corresponding input word at time t. The output of the hidden layer from previously trained words (here at time $t - x$) is populated back into the same hidden layer for the current word. V, X and W are weight matrices with respective matrix dimensions of embedding size × hidden layer size for V, hidden layer size × hidden layer size for X and hidden layer size × 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 embedding of the current word, the feature vector now has an additional dimension describing the number of past training steps that are considered for the current training step.

The hidden layer is built using `tf.nn.rnn_cell.LSTMCell()`[14], a function that creates a long short-term memory (LSTM) cell for recurrent networks which

[14] This function uses an implementation proposed by S. Hochreiter et. al. in 1997 [10].

is then used in `tf.nn.dynamic_rnn()` to compute the logits for the output layer.

Loss, predictions and accuracy are implemented in the same way as in the FNN. In addition, the same optimizer (`tf.train.AdamOptimizer()`) is used.

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 data[1] 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 following[2]:

```
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>*

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 templates[3]:

```
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 only[4].
- 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. Table 4.1 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.

Previous Template	New Template	English Equivalent
Alle Module **vom** {Program:degree} {Program:name}	Alle Module **(vom | von | in | im)** {Program:degree} {Program:name}	*All modules* ***(of | in)*** *{Program:degree} {Program:name}*
Welche Module werden von Prof {Person:firstname} {Person:lastname} **angeboten**	Welche Module werden von Professor {Person:lastname} **(unterrichtet | angeboten | gehalten)**	*Which modules are* ***(taught | offered | held)*** *by Professor {Person:lastname}*
Wieviele LP **hat** das Modul {MTSModule:title}	**(Wieviele | Wieviel)** LP **(hat | bringt)** das Modul {MTSModule:title}	***How many*** *ects does the module {MTSModule:title}* ***(have | yield)***
Informationen zu Modul MTSModule:title	**(Informationen | Details | Mehr) (zu | zum)** Modul {MTSModule:title}	***(Information | details | more) of*** *the module {MTSModule:title}*

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 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. The gained knowledge from the evaluation of these models was subsequently used to train further models with tuned parameters.

For the training of the FNN models, the following 5 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***) constitutes a mathematical function that processes input information on each artificial neuron of the network to calculate a corresponding output

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 93.

Group	*p*	*e*	*s*	*n*	*a*	Models
1	**0–12**	50	100	1	relu	*13*
2	1	**1, 5, 10, 25, 50–700**[5]	100	1	relu	*18*
3	1	50	**10, 25, 50–1000**[5]	1	relu	*22*
4	1	50	100	**1, 5, 10, 20–140**[6]	relu	*10*
5	1	**200–600**[5]	350	1	relu	*9*
6	1	250	**200–600**[5]	1	relu	*9*
7	1	**50–500**[5]	**50–500**[5]	5	relu	*10*
8	1	250	350	5	**relu, ...**[7]	*8*

Table 4.2: The parameter configuration for training the FNN models. The rows represent the training groups with the corresponding variable parameter (in bold), 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 first column denotes the group number (for referencing training groups) whereas the last column shows the resulting number of models for each training group.

[5] With a step size of 50

[6] With a step size of 20

[7] The following 8 activation functions were considered: `RELU`, `RELU6`, `ELU`, `SIGMOID`, `TANH`, `SELU`, `SOFTPLUS` and `SOFTSIGN`. See Chapter 3.1.1

For the training of the RNN models, the following 5 parameters were considered:

- **Number of time steps** (***t***) specifies how many past training steps should be considered for the training of the current word
- **Embedding size** (***e***) describes the dimension of the word embeddings (as in the FNN)
- **Hidden layer size** (***s***) characterizes the dimension of the hidden layer (as in the FNN)
- **Number of training epochs** (***n***) indicates how often the whole training corpus is processed (as in the FNN)
- **Activation function** (***a***) constitutes a mathematical function (as in the FNN)

Based on these 5 parameters, Table 4.3 shows different parameter combinations. The emphasized cells show the respective parameter that is variable. The resulting number of models is 47; however, due to an overlap in configurations of the training groups, the total number of models to be trained with the RNN is 44.

All in all, a total of 140 distinct models are trained, given the parameter specification for the two neural network architectures above.

Group	t	e	s	n	a	Models
9	**1–10**	50	50	1	relu	*10*
10	1	**1, 5, 10, 20, 50–400**[8]	50	1	relu	*12*
11	1	50	**10, 25, 50–500**[8]	1	relu	*12*
12	1	50	50	**1, 5–20**[9]	relu	*5*
13	6	50	50	1	**relu, ...**[10]	*8*

Table 4.3: The parameter configuration for the training of RNN models. The rows represent the training groups with the corresponding variable parameter (in bold), while the columns represent the single training parameters (***t*** (number of time steps), ***e*** (embedding size), ***s*** (hidden layer size), ***n*** (number of training epochs), ***a*** - activation function). The first column denotes the group number (for referencing training groups) whereas the last column shows the resulting number of models for each training group.

[8] With a step size of 50

[9] With a step size of 5

[10] For the RNN the same 8 activation functions as for the FNN are used: `RELU`, `RELU6`, `ELU`, `SIGMOID`, `TANH`, `SELU`, `SOFTPLUS` and `SOFTSIGN`. See Chapter 3.1.1

5 Evaluation and Comparison

After giving a short description of how the evaluation tests were created, this chapter provides the evaluation results of all models of the 13 training groups that were trained using the variation of training parameters previously described. In addition to the results of each training group, 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 reasonably 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.

Table 5.1 and Table 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 1094 tagged test sentences, consisting of 7986 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 nevertheless tagged correctly.

TOPIC	SENTENCES	EXAMPLE
ECTS	44	*Which modules have 6 ECTS*
Time	72	*Which courses are on Tuesday*
Faculty	56	*All modules of faculty 4*
Participants	54	*Which courses are limited to 30 participants*
Persons	85	*Which courses are taught by Professor Martinelli*
Program	60	*I'm searching for all modules of the computer science program*
Modules	80	*Modules with the title Cognitive Algorithms*
Chair	50	*All modules of institute Media Science*
Exam	32	*Courses with Group Lecture as examination*
Courses	45	*When does Internet Security take place*
Locations	42	*All courses in room Mainbuilding H 1012*
	620	

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 620 tagged sentences, consisting of 4317 word-tag tuples.

The 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.

$$\kappa = \frac{p_o - p_c}{1 - p_c} \tag{5.1}$$

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 [7] and considers the possibility of agreement by chance.

Topic	Sentences	Example
ECTS	44	*All modules with 6 ECTS*
Time	55	*Courses that take place on Tuesday*
Faculty	56	*Show all modules of faculty 4 to me*
Participants	54	*List all courses that have a limitation of 30 participants*
Persons	58	*Show all courses that are taught by Professor Martinelli*
Program	10	*Show all modules of computer science*
Modules	60	*Show information about the module Cognitive Algorithms*
Chair	20	*All modules that the institute Media Science offers*
Exam	45	*Show me all courses that have Group Lecture as examination*
Courses	30	*The date of the first meeting of Internet Security*
Locations	42	*All courses in auditorium Mainbuilding H 1012*
	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. [14], which is used in this thesis.

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, based on a proposal of J. Landis et. al. [14].

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 present the evaluation results of the different architectures for each test set, including the corresponding accuracy values for each model. To improve readability in the evaluation sections themselves, the figures showing the results for Cohen's Kappa can be found in appendices A.2 and A.3.

5.2.1 Feed-Forward Neural Network Models

The **first training group** for the FNN refers to parameter p: the number of preceding words that are used to build the feature vector. Figure 5.1 presents the accuracy achieved for both tests.

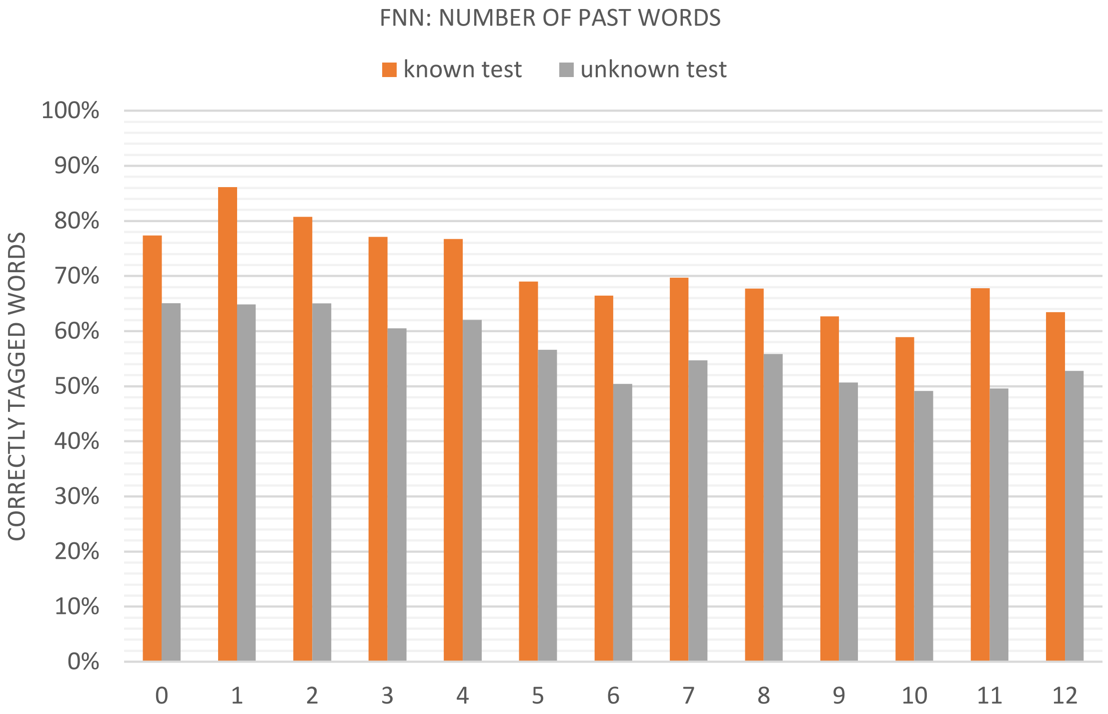


Figure 5.1: The evaluation results of the FNN for parameter p: the number of preceding words.

For the known test, the model with $p = 1$ clearly achieves the highest accuracy with 86.1% correctly tagged words. It also has the highest kappa score with $\kappa = 0.802$. With more than one preceding word, the accuracy decreases. The best result (65.1% accuracy and $\kappa = 0.588$) for the unknown test was reached by the model trained without any preceding words at all. It should be noted, however, that the results of the two following models with 1 or 2 preceding words are almost as precise as the first one. With more than 2 preceding words, the accuracy decreases.

Training group 2 included the evaluation based on 18 different values for the embedding size e of the input words for the FNN. The results are presented in Figure 5.2.

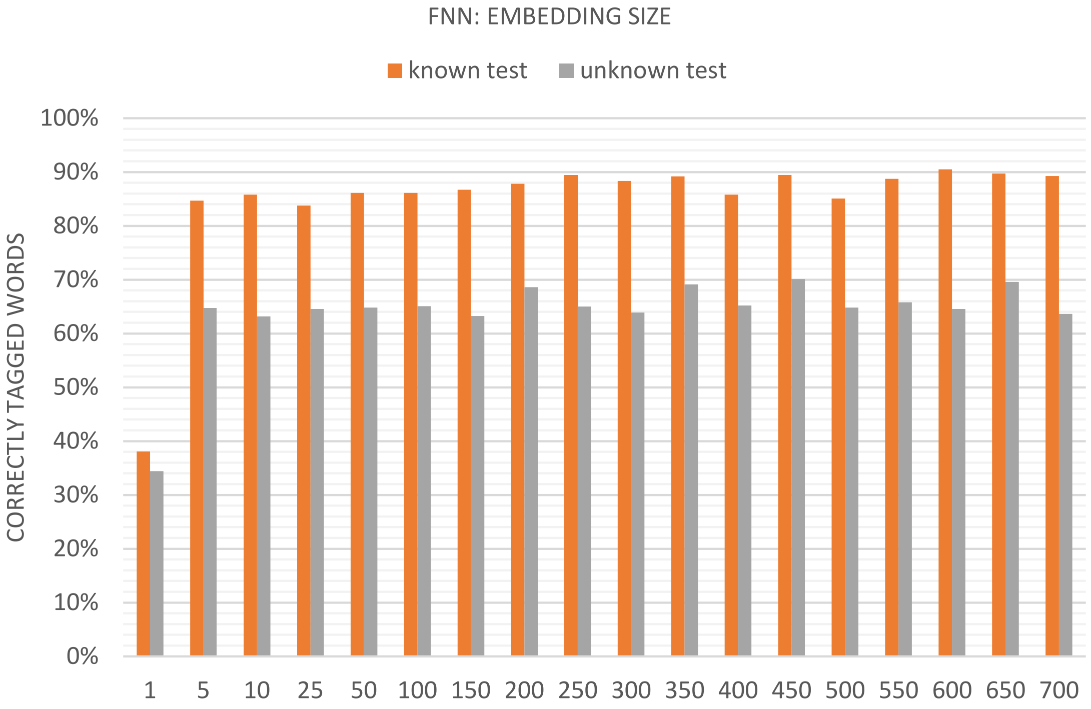


Figure 5.2: The evaluation results of the FNN for parameter e: the embedding size.

With a few exceptions, the accuracy for the known test increases slightly with larger embeddings, reaching a maximum of 90.5% (and $\kappa = 0.801$) on an embedding size of 600. The highest accuracy (70.1%, $\kappa = 0.639$) for the unknown test was reached by the model with an embedding size of 450.

The results of the evaluation of **training group 3**, which varied the size of the hidden layer s, is illustrated in Figure 5.3.

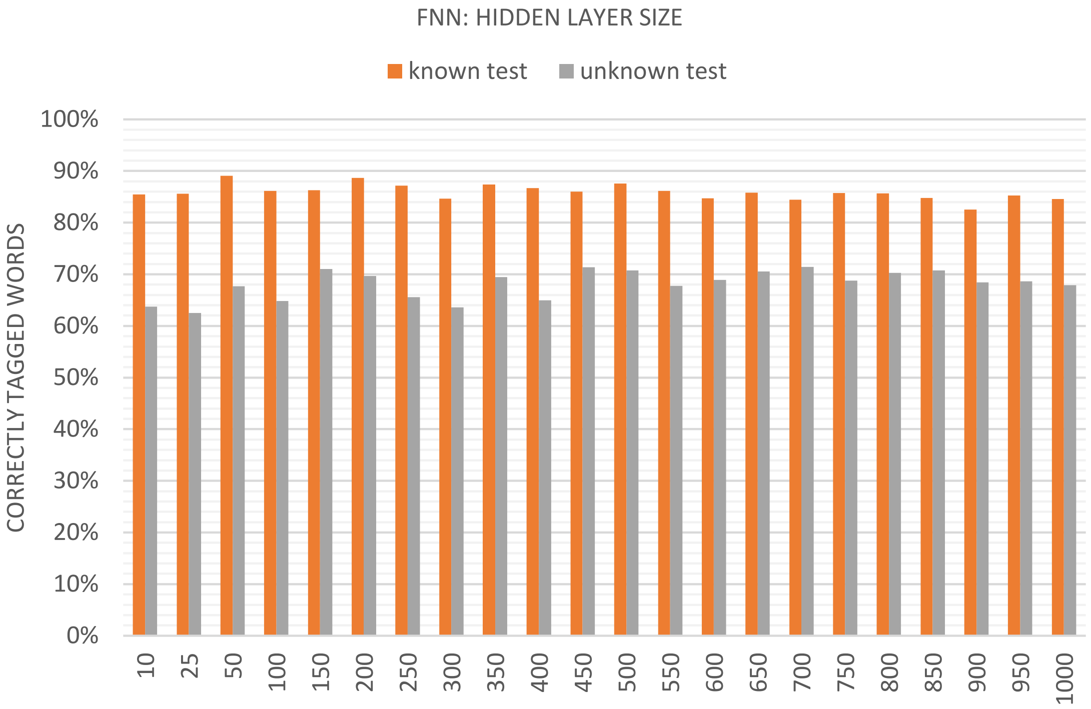


Figure 5.3: The evaluation results of the FNN for parameter s: the size of the hidden layer.

The highest accuracy of 89.1% was achieved on the known test with a hidden layer size of 50 neurons, however, the highest kappa value $\kappa = 0.813$ was assigned to a hidden layer size of 350. Although the accuracy of the models with $s < 500$ seems to be slightly higher, no clear trend can be observed. The case is similar for the unknown test, except that models with $s > 450$ have a higher, less fluctuating accuracy. The best result (71.5%, $\kappa = 0.643$) was reached by the model with $s = 700$, however, the highest kappa value $\kappa = 0.649$ was assigned to a hidden layer size of 450.

For **training group 4**, models were trained with different numbers of training epochs n. Figure 5.4 presents the results.

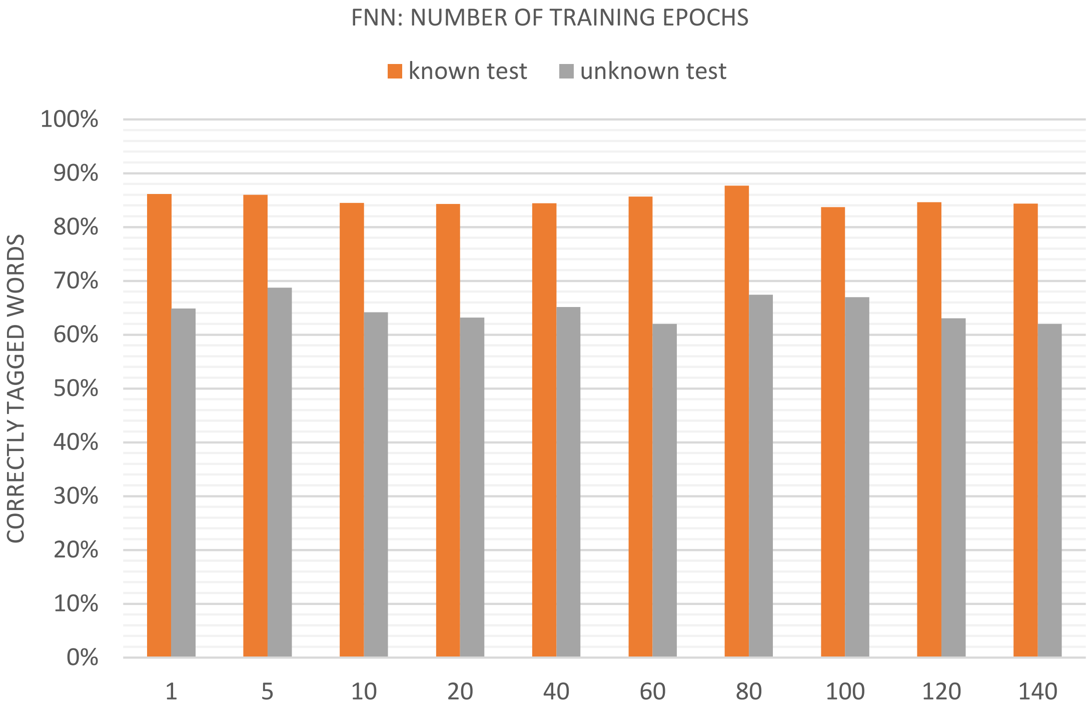


Figure 5.4: The evaluation results of the FNN for parameter n: the number of training epochs.

Overall, all models of the different training epochs achieved similar results on the known test without a discernible tendency. However, the model with $n = 80$ stands out slightly, with the highest accuracy of 87.7% and the second highest $\kappa = 0.808$. For the unknown test, the model with only 5 training epochs achieved a maximum accuracy of 68.8% and $\kappa = 0.622$.

Similar to training group 2, **training group 5** refers to the embedding size e, but utilizing a larger hidden layer size ($s = 350$). The results are presented in Figure 5.5.

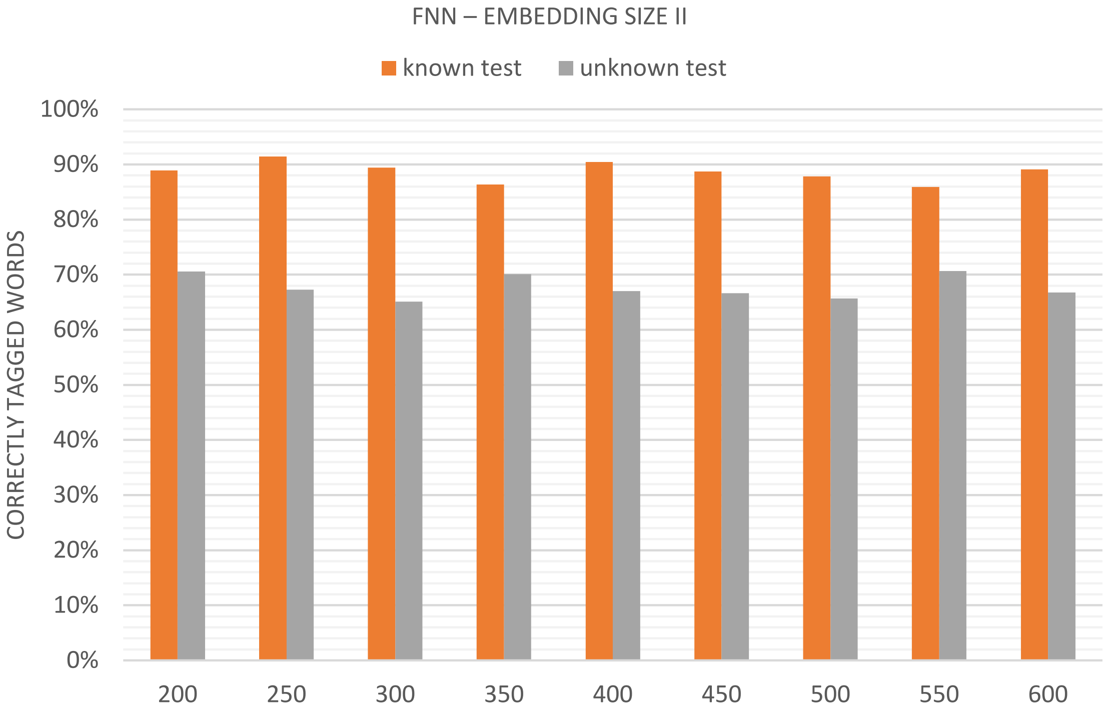


Figure 5.5: The evaluation results of the FNN for the embedding size e with a larger hidden layer size $s = 350$.

Compared to training group 2, the results were slightly better. Evaluating the known test, the highest accuracy of 91.4% ($\kappa = 0.807$) was reached by a model with embedding size $e = 250$. Nevertheless, the highest kappa value $\kappa = 0.818$ was assigned to the model with $e = 500$, which achieved a lower accuracy of 87.8%. For the unknown test, the model with $e = 550$ tagged the most words correctly (70.7%, $\kappa = 0.640$).

Similar to training group 3, **training group 6** refers to the hidden layer size *s*, with the difference of using a larger embedding size ($e = 250$). Figure 5.6 presents the results.

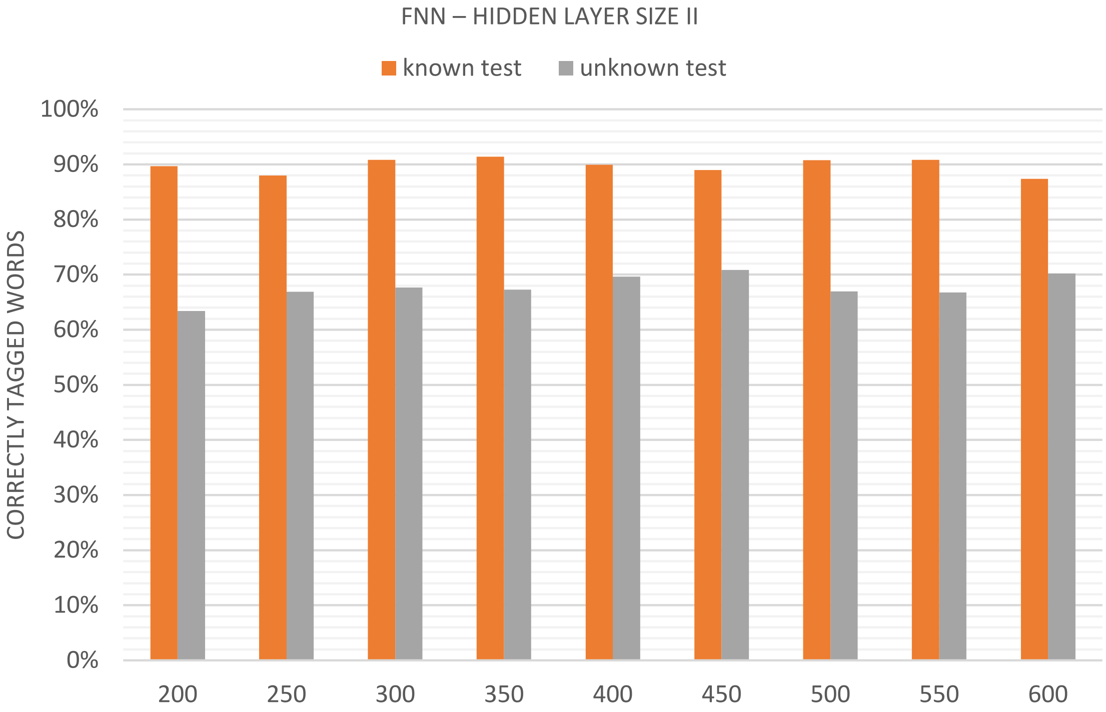


Figure 5.6: The evaluation results of the FNN for the hidden layer size *s* with a larger embedding size $e = 250$.

This parameter configuration performed slightly better for both tests comparing to training group 3. Evaluating the known test, the highest accuracy of 91.4% ($\kappa = 0.807$) was again reached by the model with hidden layer size $s = 350$. Nevertheless, the highest kappa value $\kappa = 0.821$ was assigned to the model with $s = 500$, which achieved a slightly lower accuracy of 90.8%. For the unknown test, the model with $s = 450$ tagged the most words correctly (70.9%, $\kappa = 0.643$).

Training group 7 increases the size of the whole network by enlarging the embedding size e and the hidden layer size s at the same time (combined to parameter es. The results are presented in Figure 5.7. The value on the horizontal axis represents both, the embedding size and the hidden layer size.

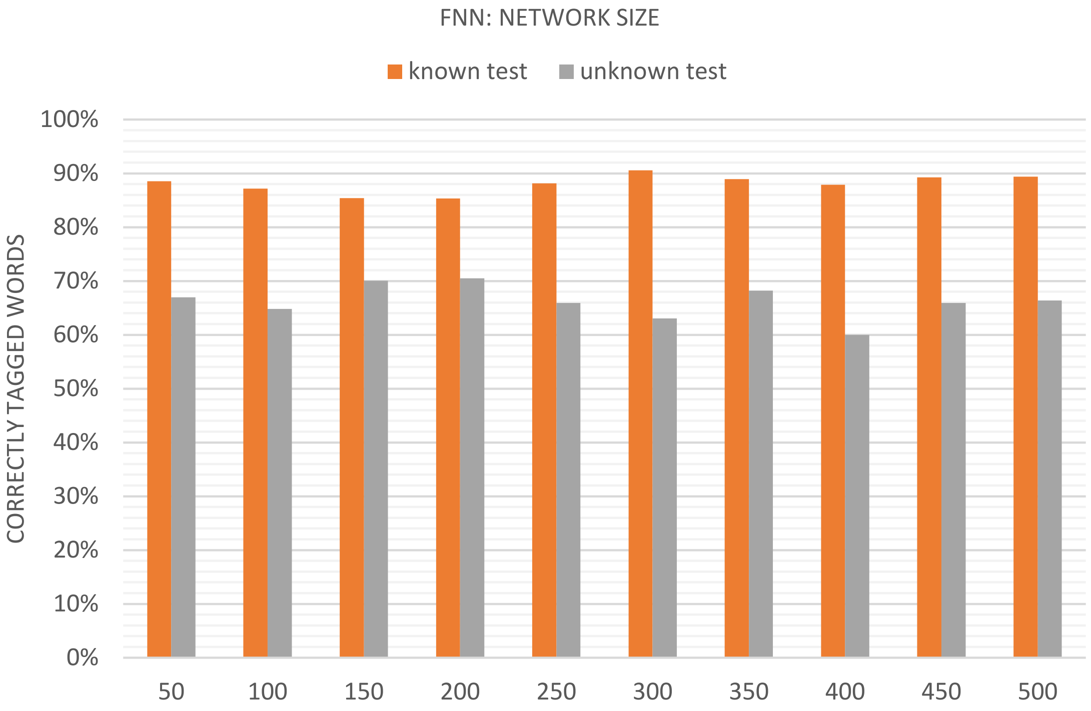


Figure 5.7: The evaluation results of the FNN for enlarging the whole network by increasing the embedding size e and the hidden layer size s at the same time.

The highest accuracy of 90.6% ($\kappa = 0.802$) was achieved by the model with $es = 300$ for the known test, however the highest kappa value $\kappa = 0.809$ was assigned to a dimension of $es = 500$. The best result for the unknown test was reached by $es = 200$ with an accuracy of 70.5% and $\kappa = 0.639$.

The last training group for the FNN, **training group 8**, refers to parameter a: the activation function (all activation functions used for training were introduced in Chapter 3.1.1). The accuracy results of 8 models that were trained with different activation function each is presented in Figure 5.8. The configuration of the other training parameters was based on the results of the preceding training groups, using $t = 1$, $e = 250$ and $s = 350$. The number of training epochs was chosen with respect to a reasonable training time.

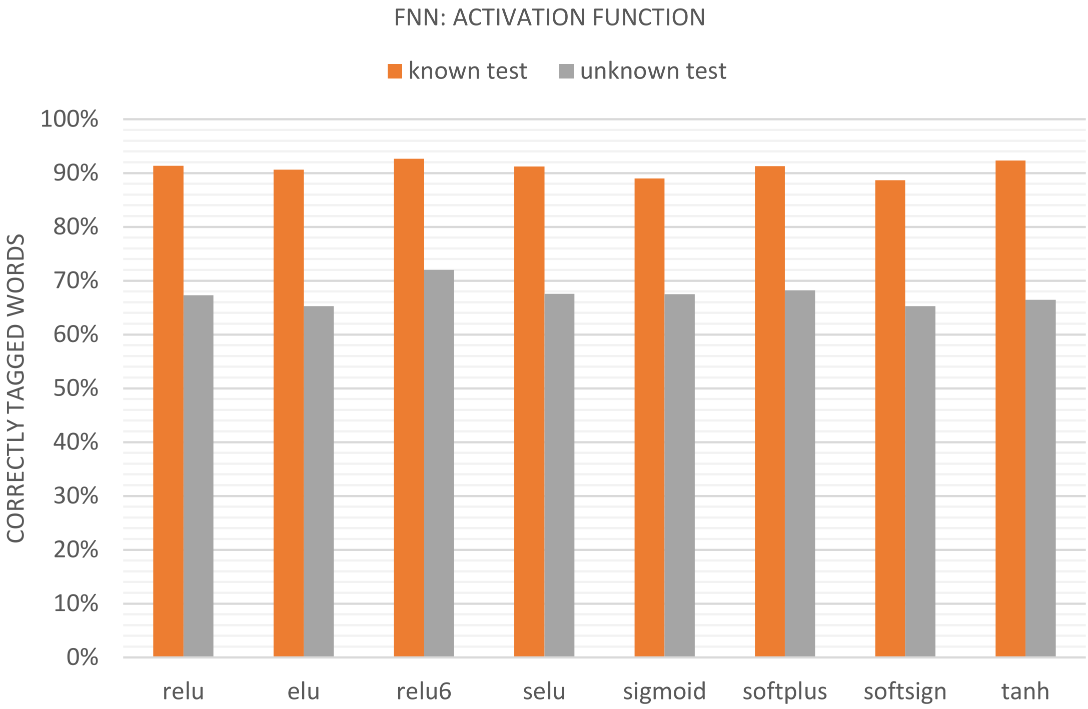


Figure 5.8: The evaluation results of the FNN for parameter a: the activation function.

For the known test, the model using RELU6 as activation function achieves the highest accuracy with 92.6% correctly tagged words and the highest kappa score with $\kappa = 0.818$. The same model also outperforms the other models for the unknown test, reaching an accuracy of 72.0% and $\kappa = 0.654$.

Considering the evaluation results of all 8 training groups for the FNN, the model with the following parameter configuration was found to achieve the highest accuracy, as well as the highest kappa value: one preceding word, embedding size of 250, hidden layer size of 350, 5 training epochs and `RELU5` as the activation function.

5.2.2 Recurrent Neural Network Models

The first training group for the RNN, **training group 9**, contains models trained with 10 different time steps t. The evaluation result is presented in Figure 5.9.

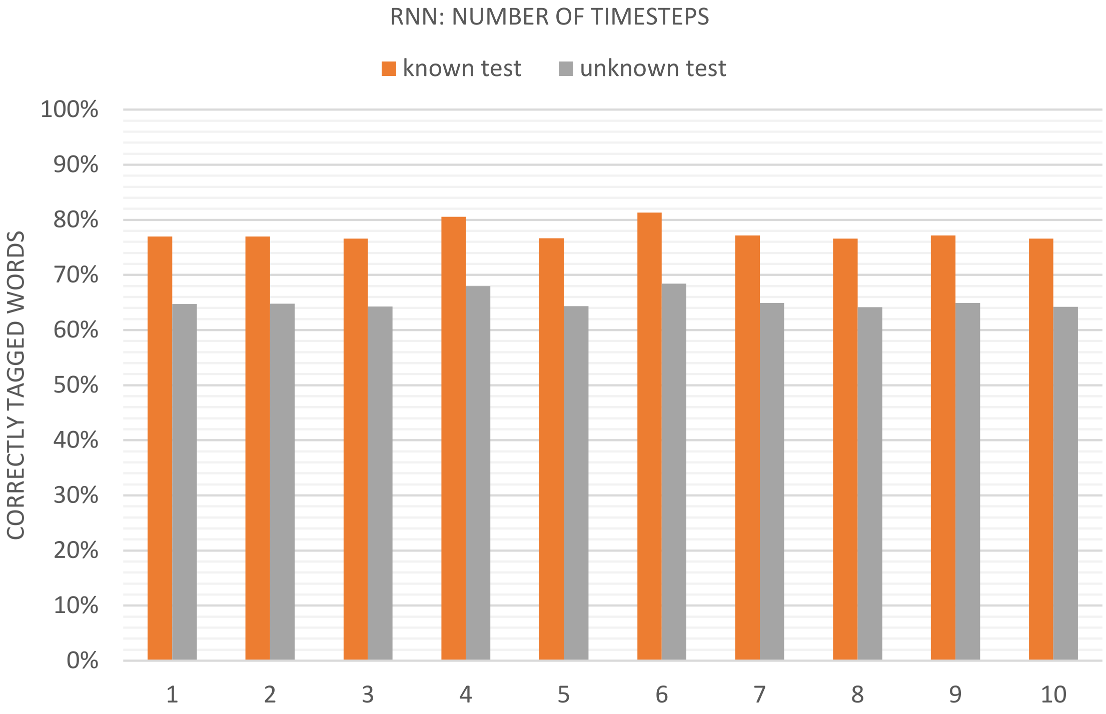


Figure 5.9: The evaluation results of the RNN: Cohen's Kappa for parameter t: the number of time steps.

Overall, all models based on the different time steps achieved similar results on both tests without a discernible tendency. However, for both tests, the models with $t = 4$ and $t = 6$ stands out with the highest accuracy of 81.3% and $\kappa = 0.728$ for the known test and 68.4% and $\kappa = 0.631$ for the unknown test for $t = 6$.

Training group 10 included the evaluation based on 12 different values for the embedding size e of the input words for the RNN. The results are presented in Figure 5.10.

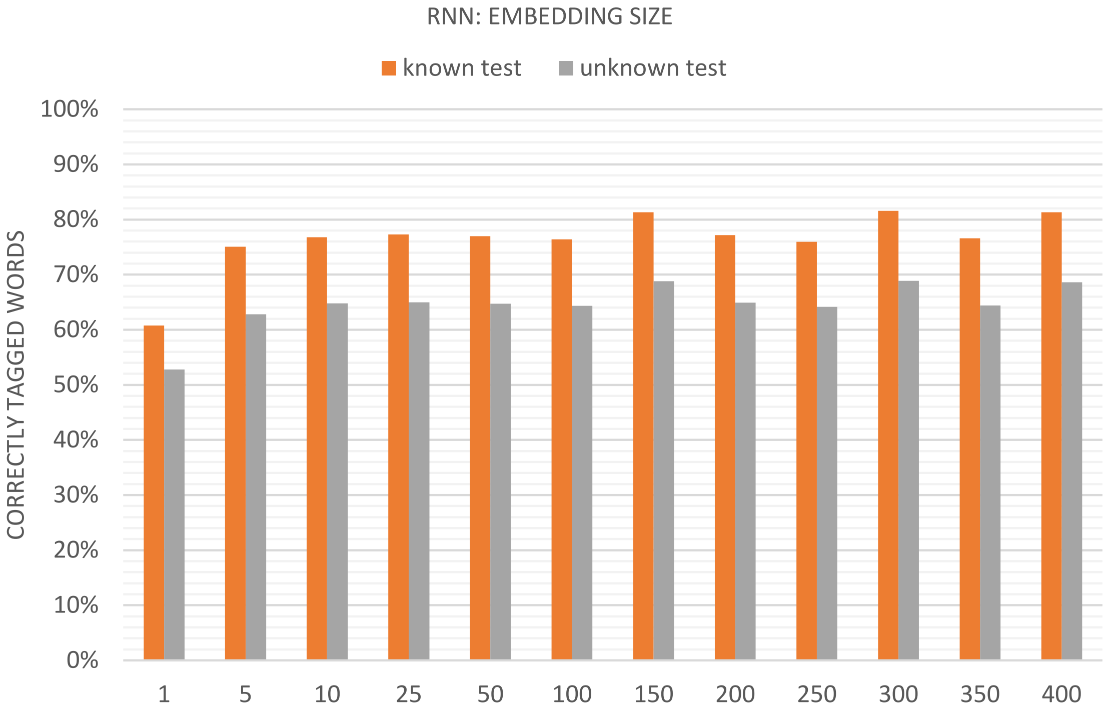


Figure 5.10: The evaluation results of the RNN for parameter e: the embedding size.

With a few exceptions, the accuracy for both tests increases slightly with larger embeddings, reaching a maximum of 81.6% ($\kappa = 0.723$) for the known test and 68.9% ($\kappa = 0.635$) for the unknown test utilizing an embedding size of 300. Models with embedding sizes of 150 and 400 achieved a similar but slightly lower accuracy.

The evaluation results of **training group 11**, which varied the size of the hidden layer s is illustrated in Figure 5.11.

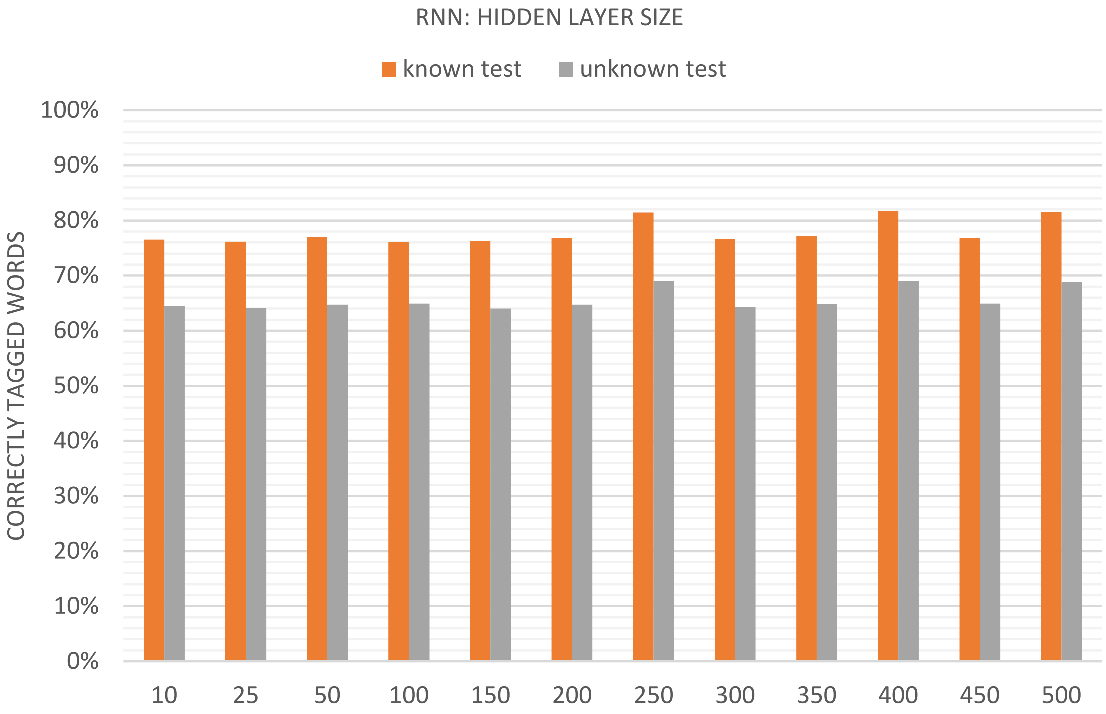


Figure 5.11: The evaluation results of the RNN: Cohen's Kappa for parameter s: the size of the hidden layer.

In general, models with a higher hidden layer size achieved a higher accuracy on both tests. Evaluating the known test, the highest accuracy of 81.8% ($\kappa = 0.726$) was reached by a model with hidden layer size $s = 400$. For the unknown test, the model with $s = 250$ tagged the most words correctly (69.1%, $\kappa = 0.632$). Nevertheless, the highest kappa value $\kappa = 0.636$ was assigned to the model with $s = 400$ again, which achieved a slightly lower accuracy of 68.98%.

Training group 12 included the evaluation based on 5 different numbers of training epochs e. The results are presented in Figure 5.12.

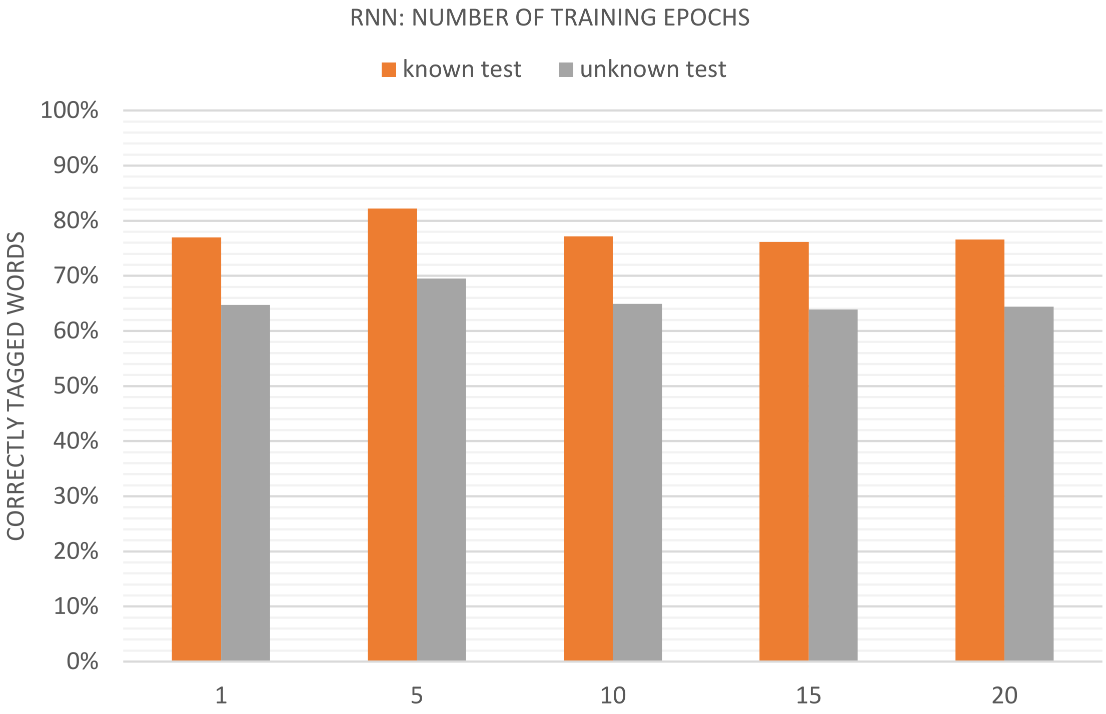


Figure 5.12: The evaluation results of the RNN: Cohen's Kappa for parameter n: the number of training epochs.

For both tests, the model with $n = 5$ achieved the highest number of correctly tagged words: 82.2% ($\kappa = 0.721$) for the known test and 69.5% ($\kappa = 0.642$) for the unknown test. For higher numbers of training epochs ($s > 5$), the accuracy decreases for more than 5%.

The last training group, **training group 13**, refers to parameter a: the activation function (see Chapter 3.1.1). The result for the accuracy of 8 models that were trained with a different activation function each is presented in Figure 5.13. The configuration of the time step parameter was based on the results of training group 9, using $t = 6$. The embedding size ($e = 50$) size of the hidden layer ($s = 50$) and the number of training epochs ($n = 1$) were chosen with respect to a reasonable training time.

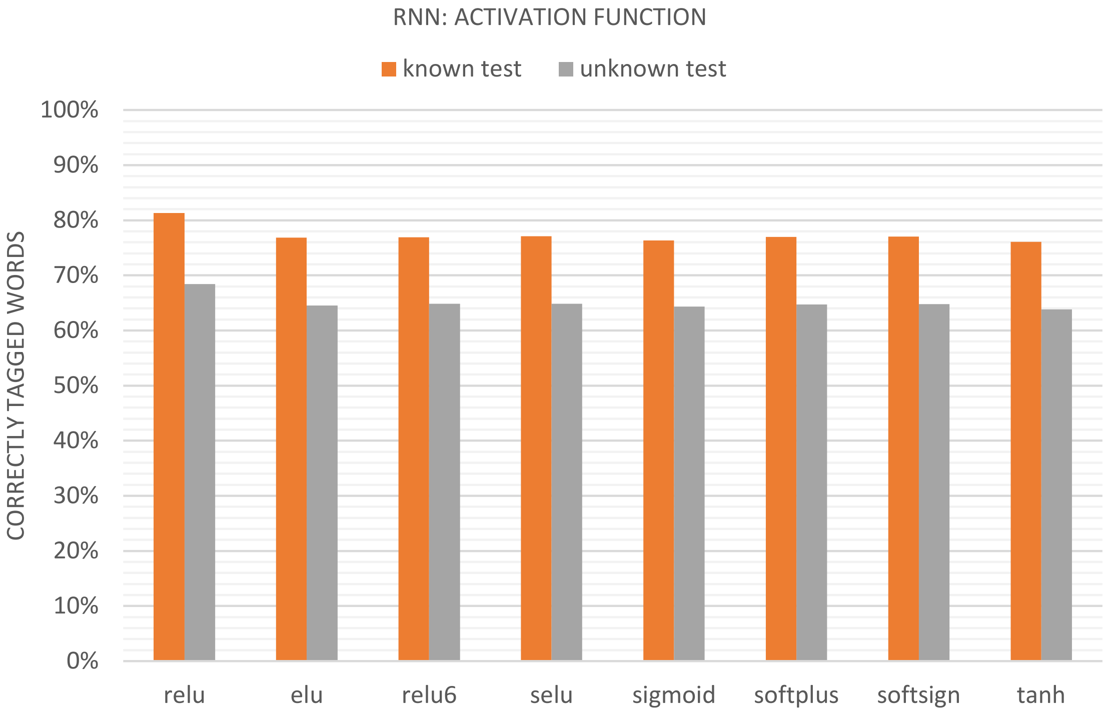


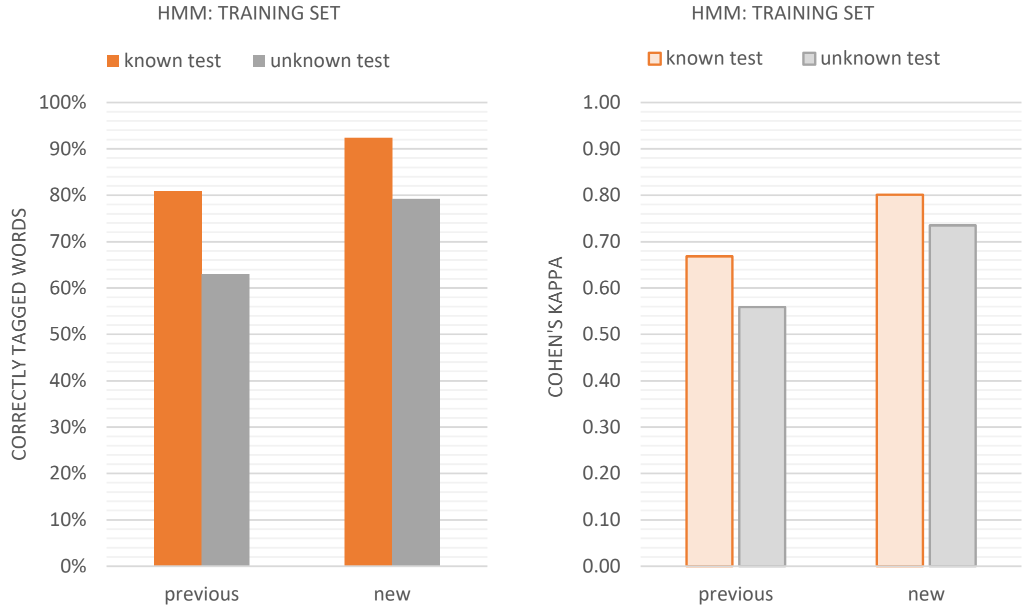
Figure 5.13: The evaluation results of the RNN: Cohen's Kappa for parameter a: the type of activation function.

The best evaluation results of this training group regarding accuracy were achieved by the model utilizing the RELU activation function. For the known test, an accuracy of 81.3% ($\kappa = 0.728$) and for the unknown test, an accuracy of 68.4% ($\kappa = 0.631$) was achieved. The other models of this training group reached an accuracy with a maximum of 77.1%.

Considering the evaluation results of all 5 training groups for the RNN, the model with the following parameter configuration was found to achieve the highest accuracy: one time step, embedding size of 50, hidden layer size of 50, 5 training epochs and RELU as activation function.

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 (also called *HMM1*) which was trained on the basis of the training data generated by T. Michael [18] is compared to a new HMM (*HMM2*) trained on the basis of the training data proposed in this thesis. The evaluation was carried out using the known and the unknown test developed in this thesis; the results are presented in Figure 5.14.



(a) Accuracy of correctly tagged words

(b) Cohen's Kappa

Figure 5.14: The evaluation results of the previous HMM trained by T. Michael [18] 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 known[2] 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.

5.3 Overall Comparison

After evaluating the neural network models and the hidden Markov models separately, the different architectures will now be compared to each other. For this purpose, the evaluation results of the best FNN model proposed in Chapter 5.2.1, the best RNN model found in Chapter 5.2.2 and the two HMM models evaluated in Chapter 5.2.3 are summarized in Table 5.4 and illustrated in Figure 5.15.

[2] As described in Chapter 5.1, the known test contains 620 sentences. 388 sentences are included in the corpus of the previous HMM, resulting in 37.4% of the sentences of the known test, that were unknown to the previous HMM.

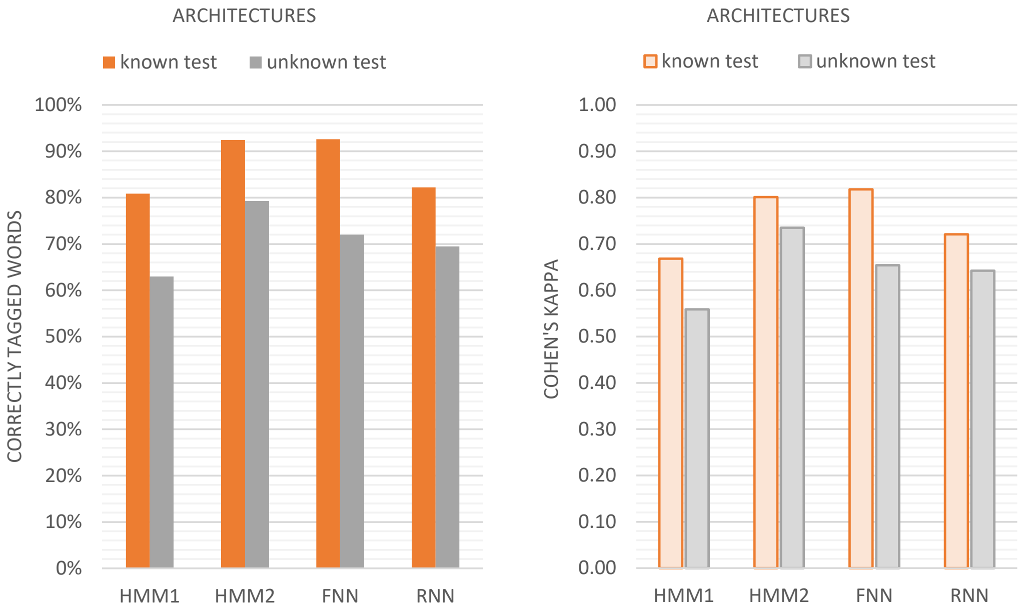
	Accuracy	Kappa
HMM1	80.87%	0.668
HMM2	92.43%	0.801
FNN	**92.63%**	**0.818**
RNN	82.21%	0.721

(a) Known Test

	Accuracy	Kappa
HMM1	62.99%	0.559
HMM2	**79.29%**	**0.735**
FNN	72.01%	0.654
RNN	69.47%	0.642

(b) Unknown Test

Table 5.4: A tabular overview of the evaluation results of HMM1 trained by T. Michael [18], HMM2 trained on the new training data proposed in this thesis and the FNN and RNN model with the highest accuracy according to the evaluation of the different training groups.



(a) Comparison of the accuracy

(b) Comparison of the kappa score

Figure 5.15: The evaluation results of the previous HMM model trained by T. Michael [18], the new HMM model trained on the new training data proposed in this thesis, the best FNN and the best RNN model according to the evaluation of the different training groups.

The FNN model achieved the highest accuracy as well as the highest kappa score for the known test, closely followed by the HMM2 model, which was only slightly less accurate. The difference between these two models is only 0.2% ($\Delta\kappa = 0.017$). With a kappa score of more than 0.8, both models offer a very good agreement with the labeled test data. However, the RNN and the HMM1 models still reached a good agreement, tagging more than 80% of the words correctly.

The result of the unknown test is a bit more distinctive. The highest accuracy as well as the highest kappa score was reached by the HMM2 model, followed by the FNN model with a difference of 7.28% ($\Delta\kappa = 0.084$). With a kappa score of more than 0.6, both models as well as the RNN model offer a good agreement with the labeled test data. Although the FNN model performed better than the RNN model, the differences between the results of both are very small (2.53%, $\Delta\kappa = 0.012$). The FNN1 model reaches a moderate agreement with the labeled test data, tagging less than two-thirds of the words correctly.

Overall, every model achieved a higher accuracy and a higher kappa score on the know test than on the unknown test. While this is an expected result, the differences between the accuracy of known and unknown test vary for each model. While this difference is smaller for the HMM2 and RNN model (13.14% and 12.74%), the HMM1 model shows a difference of 17.88%. The largest difference can be observed in the FNN model, which tagged an additional 20.63% of the words of the known test correctly, compared to those of the unknown test.

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.

The different modules of ALEX were analyzed to gain information about its language processing, the generation of training data and the hidden Markov model tagger and its interface.

Based on this knowledge, a tagger module was implemented featuring a feed-forward neural network as well as a recurrent neural network architecture. A training corpus containing tagged sentences was generated with the help of an improved set of sentence templates. With both neural network architectures, several 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, that were derived from user log data. Furthermore, both the hidden Markov model, which ALEX already utilizes for POS tagging and an HMM that was trained on the new training data, proposed in this thesis, were evaluated with these test sets.

After evaluating the trained language models, it could be shown that the HMM based on new training data significantly performed better than the previous HMM tagger, especially on unknown data. It was also shown that a tagger based on a feed-forward neural network was able to outperform this new HMM tagger on known evaluation data. Both, feed-forward and recurrent neural network models, achieved a higher accuracy than the HMM ALEX utilized before.

6.2 Discussion

According to the evaluation results, the FNN model achieved the highest accuracy for known data and would therefore be preferred for productive use. However, the HMM model based on the new training corpus (HMM2) performed significantly better on unknown data, showing the ability to generalize better. Given the fact that a chatbot has to deal with a lot of unknown input data due to users that do not know the limits of the chatbots knowledge and capabilities, the result of the unknown test is more important for the choice of the language model to use in ALEX.

When reviewing the comparison of the two hidden Markov models for the unknown test, it is apparent that the quality of the training data is of great importance. The improvements on the training corpus proposed in this thesis caused a higher accuracy in word tagging of more than 16%. However, the training data still has a fixed structure due to its origin in the sentence templates, which is a possible reason for the lower accuracy of the neural network approaches on the unknown data. The effect of the structure of the training data is especially evident in the results for the recurrent neural network, which created less accurate language models compared to the FNN.

Another important factor, which should be taken into account, especially when the database is changed or replaced, is the training time of neural network models. Depending on the parameter configuration, the training of a model could take several hours. However, in this thesis, the FNN model that performed best had a training time of 30 minutes, which should still be a feasible solution when the database is only updated every semester (as it is the case for ALEX).

6.3 Future work

Although a high accuracy of the language models was achieved with an improved training corpus and the neural network approach used in this thesis, further research could be done in certain areas to fine-tune and improve the training templates as well as the tagging results.

Furthermore, 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 superfluous 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 number of time steps, the size of the hidden layer, the size of the word embeddings, the number of training epochs and the type of activation function 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 and the training optimizer. The use of multiple LSTM cells for the RNN could be interesting too, as well as the training 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 [20].

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 demonstrated a way of implementing, training and evaluating a neural network based 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] James Bergstra, Guillaume Desjardins, Pascal Lamblin, and Yoshua Bengio. Quadratic polynomials learn better image features. *Technical report, 1337*, 2009.

[5] M.J. Bishop and E.A. Thompson. Maximum likelihood alignment of DNA sequences. *Journal of Molecular Biology*, 190(2):159 – 165, 1986.

[6] Djork-Arné Clevert, Thomas Unterthiner, and Sepp Hochreiter. Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs). *CoRR*, abs/1511.07289, 2015.

[7] Jacob Cohen. A Coefficient of Agreement for Nominal Scales. *Educational and Psychological Measurement*, 20(1):37–46, 1960.

[8] Daniel Crevier. AI: The Tumultuous Search for Artificial Intelligence. 1993.

[9] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Deep sparse rectifier neural networks. In *Proceedings of the Fourteenth International Conference on Artificial Intelligence and Statistics*, pages 315–323, 2011.

[10] Sepp Hochreiter and Jürgen Schmidhuber. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780, 1997.

[11] Diederik P. Kingma and Jimmy Ba. Adam: A Method for Stochastic Optimization. *CoRR*, abs/1412.6980, 2014.

[12] Günter Klambauer, Thomas Unterthiner, Andreas Mayr, and Sepp Hochreiter. Self-Normalizing Neural Networks. *CoRR*, abs/1706.02515, 2017.

[13] Julian Kupiec. Robust part-of-speech tagging using a hidden Markov model. *Computer Speech & Language*, 6(3):225 – 242, 1992.

[14] J. Richard Landis and Gary G. Koch. The Measurement of Observer Agreement for Categorical Data. *Biometrics*, 33(1):159–174, 1977.

[15] Qing Ma, Kiyotaka Uchimoto, Masaki Murata, and Hitoshi Isahara. Elastic Neural Networks for Part of Speech Tagging. 5, January 2000.

[16] W. S. Mcculloch. The brain computing machine. *Electrical Engineering*, 68(6):492–497, June 1949.

[17] 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.

[18] Thilo Michael. Design and Implementation of an Advisory Artificial Conversational Agent, October 2016.

[19] Marvin Minsky and Seymour Papert. *Perceptrons: An Introduction to Computational Geometry*. The MIT Press, 1969.

[20] Andreas Müller. Analyse von Wort-Vektoren deutscher Textkorpora, June 2015.

[21] Vinod Nair and Geoffrey E. Hinton. Rectified Linear Units Improve Restricted Boltzmann Machines. In *Proceedings of the 27th International Conference on International Conference on Machine Learning*, ICML'10, pages 807–814, USA, 2010. Omnipress.

[22] F. Rosenblatt. The Perceptron: A Probabilistic Model for Information Storage and Organization in The Brain. *Psychological Review*, pages 65–386, 1958.

[23] 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.

[24] Helmut Schmid. Part-of-Speech Tagging with Neural Networks. October 1994.

[25] 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.

[26] R. L. Stratonovich. Conditional Markov Processes. *Theory of Probability & Its Applications*, 5(2):156–178, 1960.

[27] 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
```

```
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_CurseDate: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_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_Institute:faculty C_Institute:faculty X X

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

Alle Module von Fakultät (1|2|3|4|5|6|7)
R_LIST M_MTSModule 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 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
```

```
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_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
```

A.2 FNN Evaluation: Cohen's Kappa

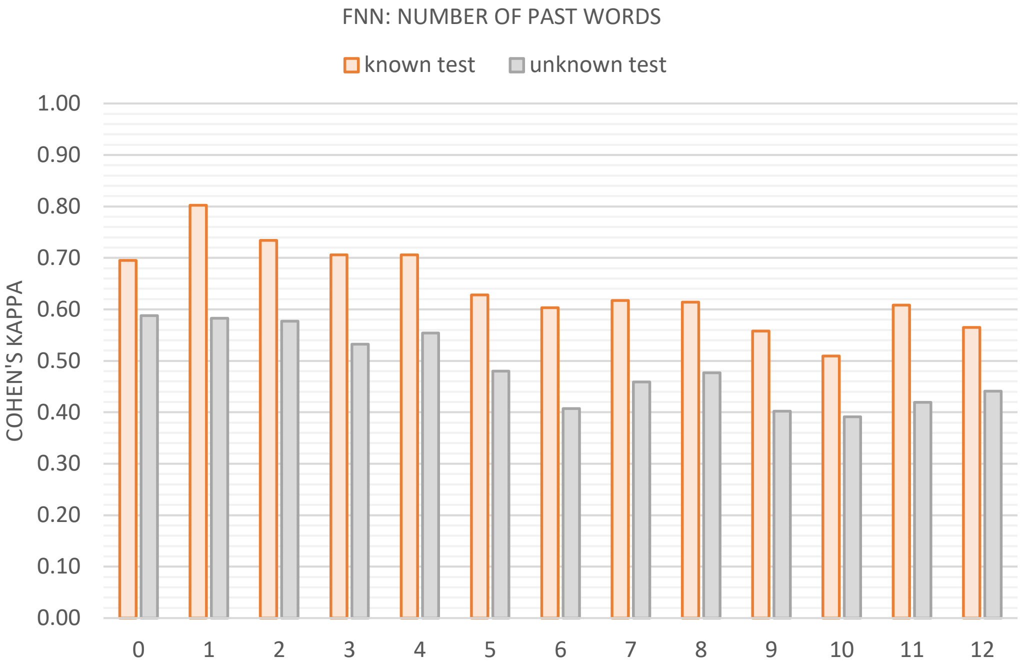


Figure A.1: The evaluation results of the FNN: Cohen's Kappa for parameter p: the number of preceding words.

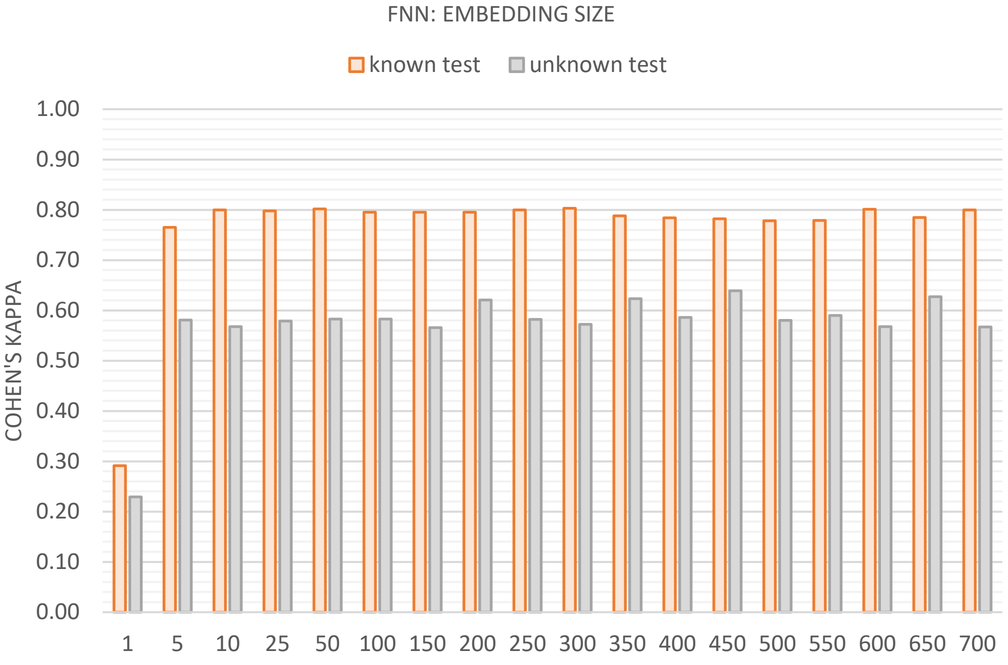


Figure A.2: The evaluation results of the FNN: Cohen's Kappa for parameter e: the embedding size.

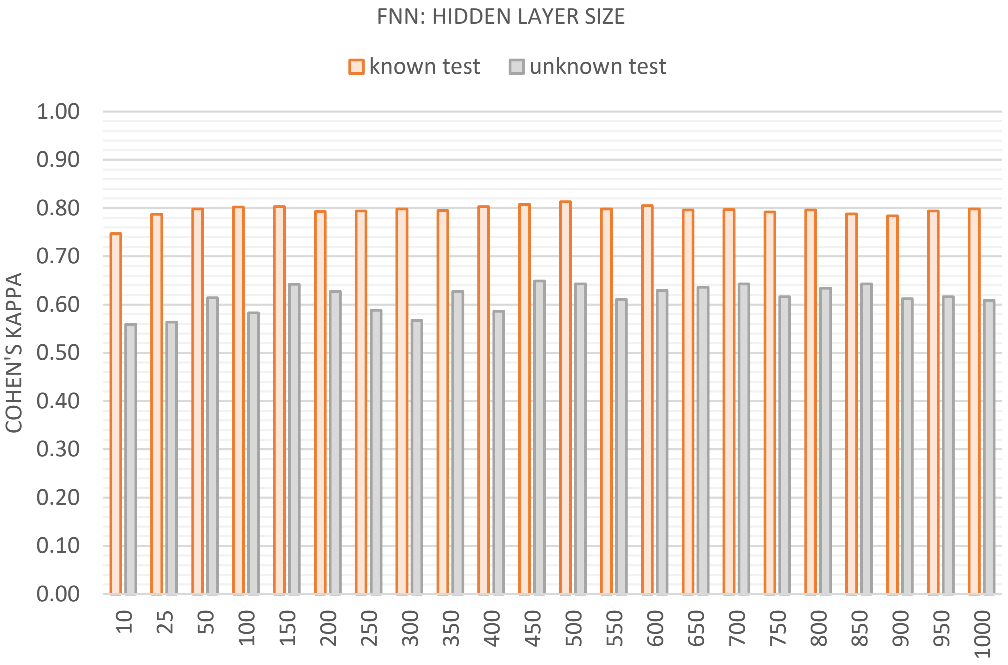


Figure A.3: The evaluation results of the FNN: Cohen's Kappa for parameter s: the size of the hidden layer.

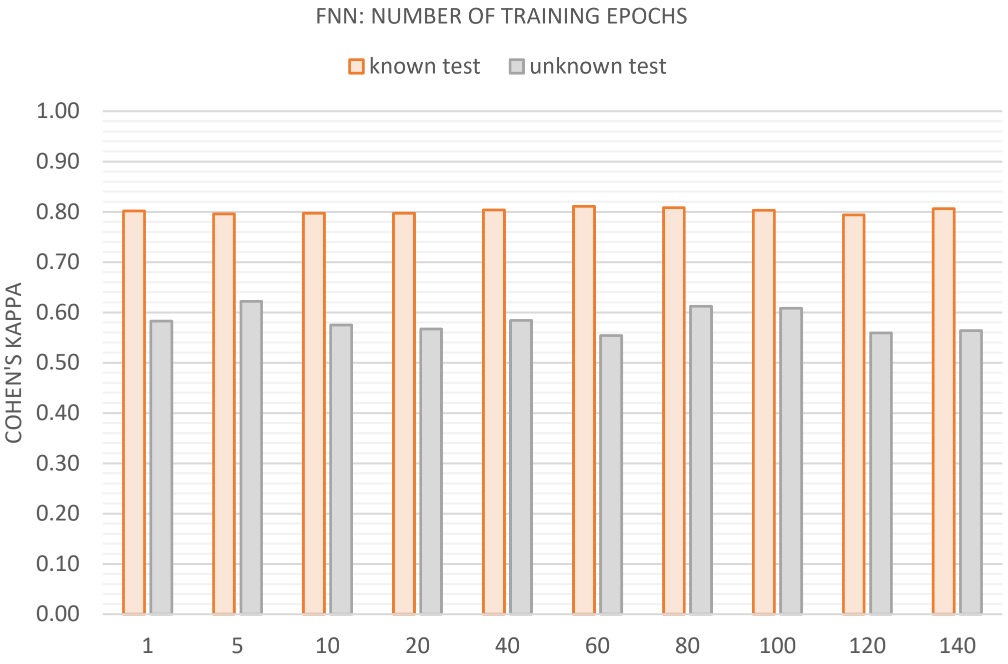


Figure A.4: The evaluation results of the FNN: Cohen's Kappa for parameter n: the number of training epochs.

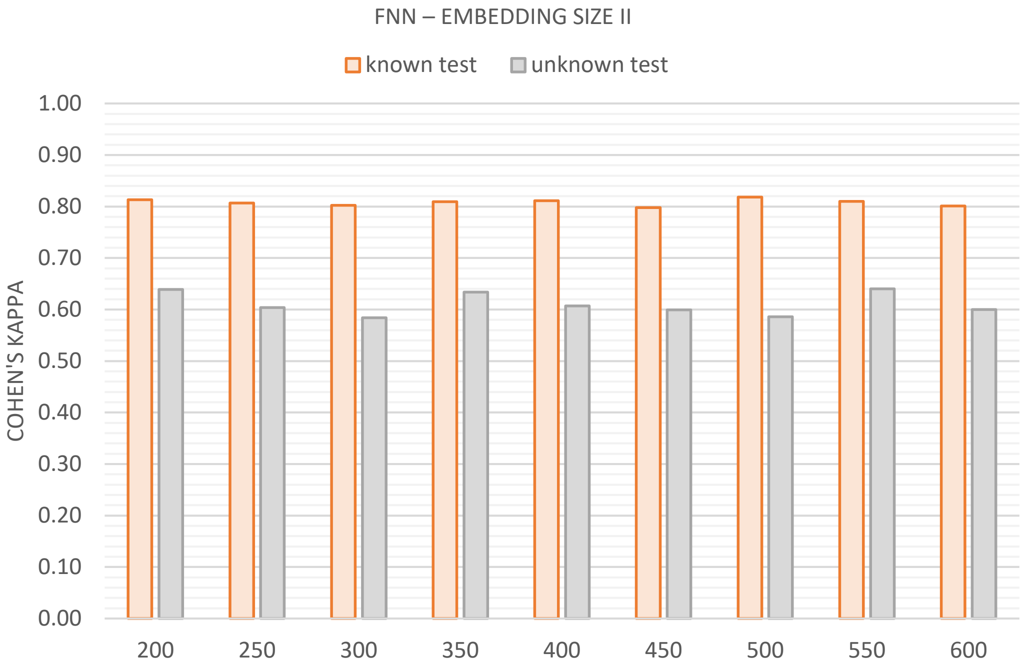


Figure A.5: The evaluation results of the FNN: Cohen's Kappa for parameter *e*: the embedding size with a higher hidden layer $s = 350$.

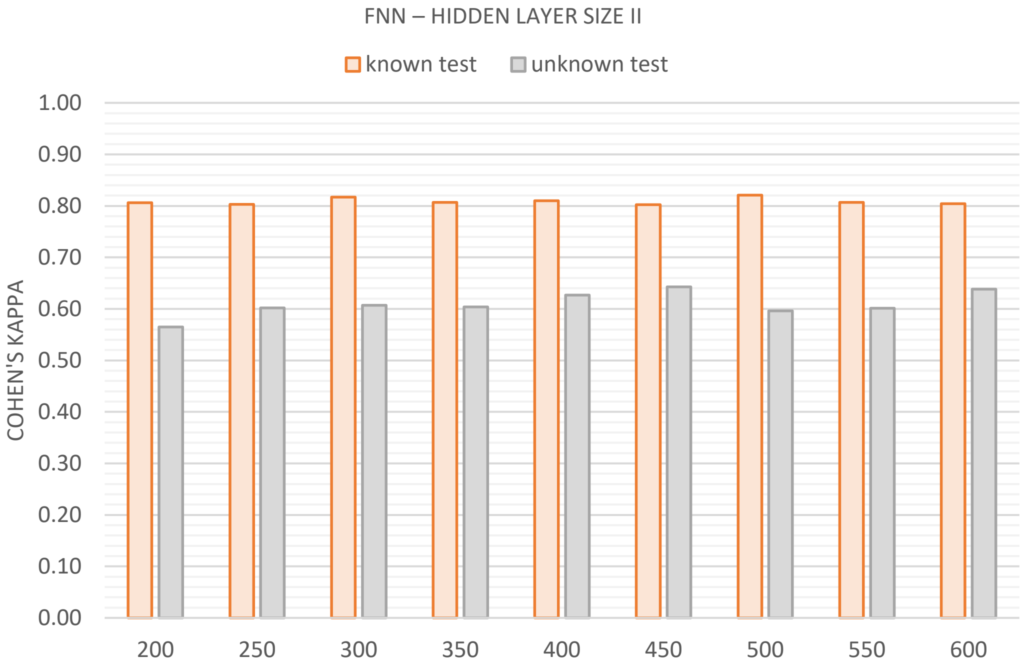


Figure A.6: The evaluation results of the FNN: Cohen's Kappa for parameter *s*: the hidden layer size for larger embeddings $e = 250$.

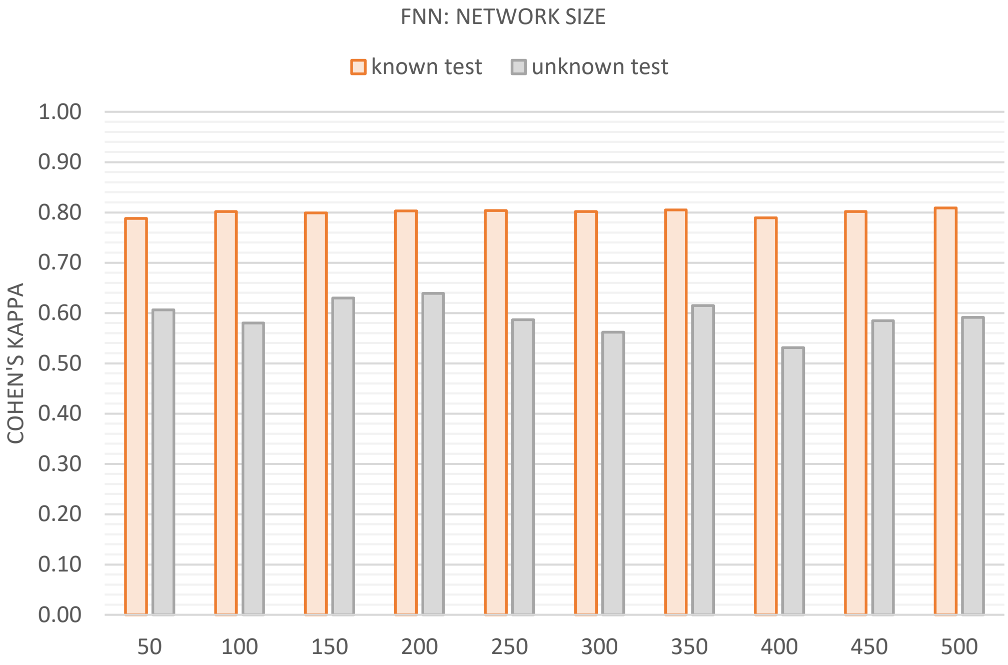


Figure A.7: The evaluation results of the FNN: Cohen's Kappa for uniform scaling of parameters e ad s, increasing the whole network size.

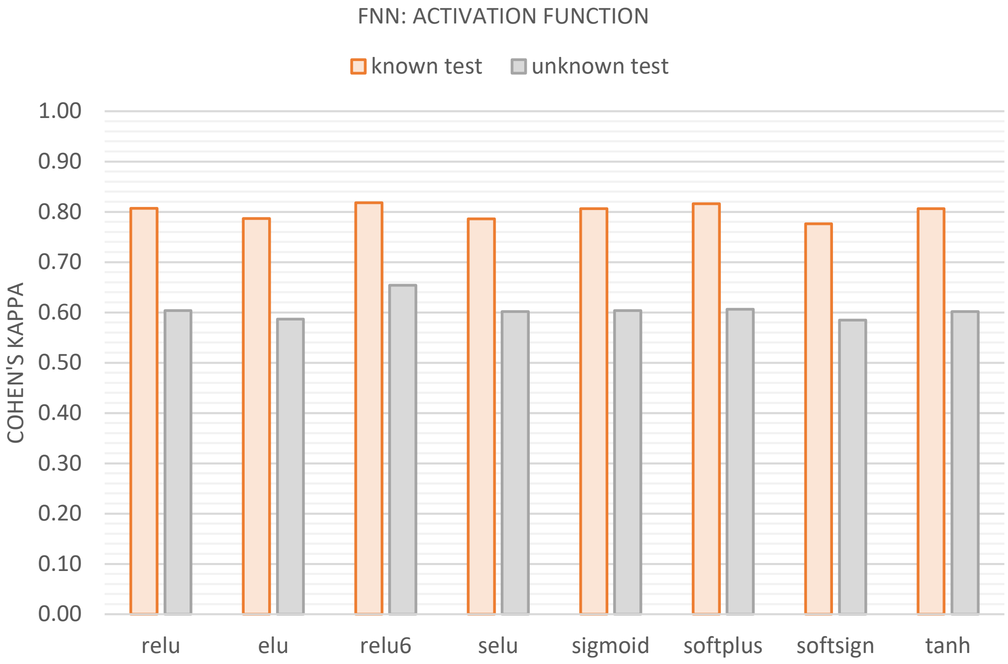


Figure A.8: The evaluation results of the FNN: Cohen's Kappa for parameter a: the activation function.

A.3 RNN Evaluation: Cohen's Kappa

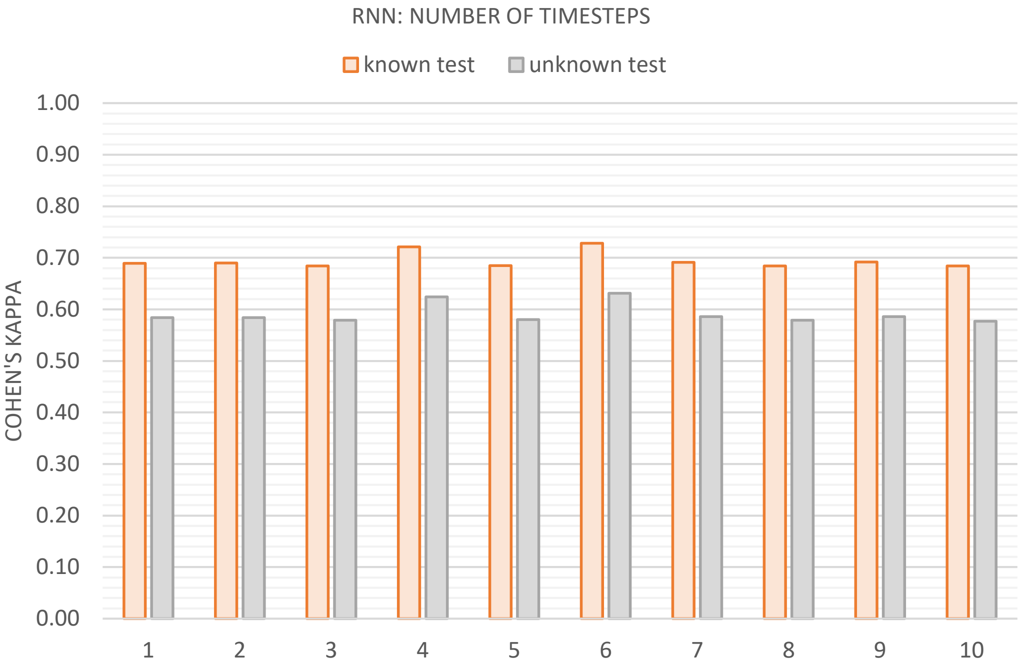


Figure A.9: The evaluation results of the RNN: Cohen's Kappa for parameter t: the number of time steps.

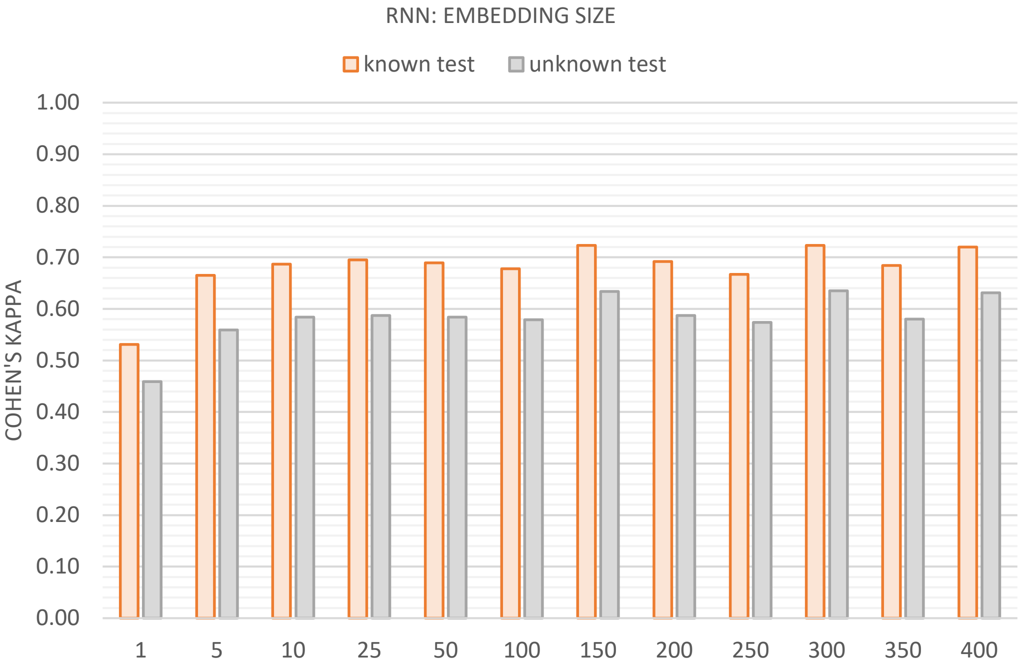


Figure A.10: The evaluation results of the RNN: Cohen's Kappa for parameter e: the size of the word embeddings.

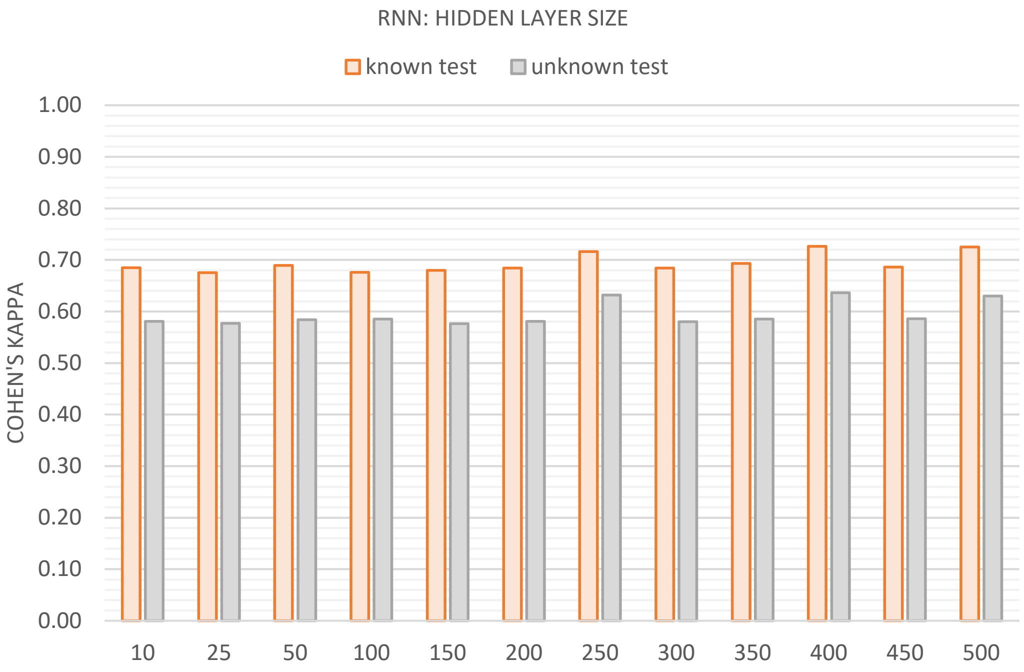


Figure A.11: The evaluation results of the RNN: Cohen's Kappa for parameter s: the size of the hidden layer.

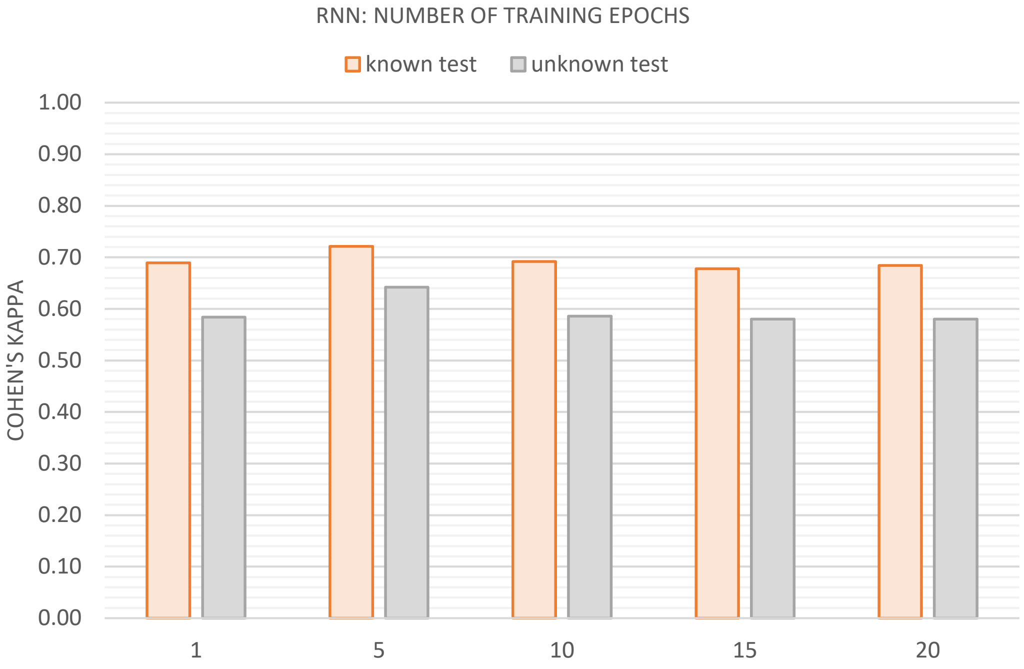


Figure A.12: The evaluation results of the RNN: Cohen's Kappa for parameter n: the number of training epochs.

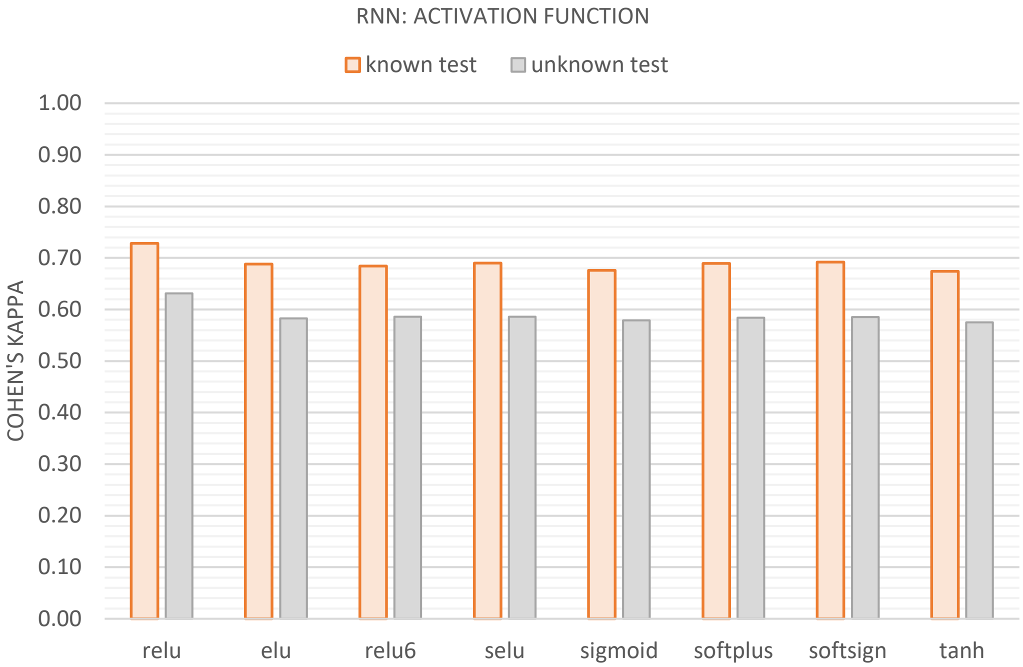


Figure A.13: The evaluation results of the RNN: Cohen's Kappa for parameter *a*: the type of activation function.