Electronic version of an article published in
International Journal of Semantic Computing
Vol. 14, No. 2 (2020) 249–272
https://doi.org/10.1142/S1793351X20400097
© World Scientific Publishing Company
https://www.worldscientific.com/worldscinet/ijsc

Towards Programming in Natural Language: Learning New Functions from Spoken Utterances

Sebastian Weigelt, Vanessa Steurer, Tobias Hey, Walter F. Tichy
Karlsruhe Institute of Technology
Institute for Program Structures and Data Organization
Karlsruhe, Germany
weigelt@kit.edu, vanessa_steurer@web.de, hey@kit.edu, tichy@kit.edu
http://ps.ipd.kit.edu

Systems with conversational interfaces are rather popular nowadays. However, their full potential is not yet exploited. For the time being, users are restricted to calling predefined functions. Soon, users will expect to customize systems to their needs and create own functions using nothing but spoken instructions. Thus, future systems must understand how laypersons teach new functionality to intelligent systems. The understanding of natural language teaching sequences is a first step towards comprehensive end-user programming in natural language.

We propose to analyze the semantics of spoken teaching sequences with a hierarchical classification approach. First, we classify whether an utterance constitutes an effort to teach a new function or not. Afterwards, a second classifier locates the distinct semantic parts of teaching efforts: declaration of a new function, specification of intermediate steps, and superfluous information. For both tasks we implement a broad range of machine learning techniques: classical approaches, such as Naïve Bayes, and neural network configurations of various types and architectures, such as bidirectional LSTMs. Additionally, we introduce two heuristic-based adaptations that are tailored to the task of understanding teaching sequences. As data basis we use 3168 descriptions gathered in a user study. For the first task convolutional neural networks obtain the best results (accuracy: 96.6%); bidirectional LSTMs excel in the second (accuracy: 98.8%). The adaptations improve the first-level classification considerably (plus 2.2 percentage points).

Keywords: Programming in Natural Language; Natural Language Understanding; End-User Programming; Conversational Interfaces; Spoken Language Understanding; Natural Language Processing; Computational Linguistics; Naturalistic Programming; Machine Learning; Neural Networks; Intelligent Systems; Artificial Intelligence.

1. Introduction

Intelligent systems with conversational interfaces became rather smart lately. One can literally communicate with virtual assistants such as Apple's Siri or Google Assistant and easily arrange meetings or check for new messages. Other voicecontrolled systems, such as humanoid robots or home automation systems, are on the rise. However, such systems still suffer from a limited range of functions. For the time being, users can access built-in functionality only; new features are added by developers solely. To eradicate this limitation, future systems must enable users to customize systems themselves, preferably using nothing but spoken instructions (because it seems most natural to enhance conversational interfaces by means of conversation). Understanding how laypersons teach new functionality will be a big step towards comprehensive end-user programming.

As for today, this task is not well studied. Therefore, we carried out a user study to analyze the semantics of teaching sequences. The participants were supposed to teach a humanoid robot in four different scenarios. We gathered 3168 descriptions from 870 participants.

Based on this dataset and the findings from the study, we develop a hierarchical classification approach. We observed that about a third of the descriptions are rather sequences of instructions than teaching efforts. Thus, the first classification task is to determine, whether a description constitutes an explicitly verbalized teaching intent, such as “we gonna learn how to [...]” or “to [do A] you have to [...]”. All teaching efforts are passed to the second-level classifier, that analyzes the semantic structure. We found that teaching efforts are usually composed of three distinct semantic parts. The first is a verbalization of the intent to teach a new function, e.g. “preparing a cup of coffee *means* [...]”. The second includes all actions to be performed, i.e. intermediate steps, to learn the function, e.g. “[...] put a coffee mug under the dispenser and then press the red button on the coffee machine [...]”. Moreover, many descriptions contain superfluous phrases (in the context of teaching sequences), such as greetings or remarks, e.g. “Hello” or “coffee is a beverage that people like to drink”. For the first classification task we implemented five classical machine learning techniques and three different types of neural networks: ANNs, CNNs, and RNNs. We tested various architectures of neural networks (e.g. LSTMs), added further layers (e.g. GMax), and systematically altered the hyper-parameters. For the second task we narrowed down to neural networks. Additionally, we implemented heuristics to improve the performance of our approach; they are tailored to the task but dataset-agnostic. The work presented here is part of the project *PARSE* [1], in which we study the opportunities of end-user programming in (spoken) natural language.

The remainder is structured as follows. First, we provide a task definition in Section 2 before we introduce the project *PARSE* in Section 3. Afterwards we detail the user study and the resulting dataset in Section 4. In Section 5 we compare the performance of the different machine learning approaches (and configurations) for the hierarchical classification task; we also present our adaptations there. Then, we discuss related work from the field of programming with natural language in Section 6. Finally, we conclude our work in Section 7 and discuss future work.

This article is an extended version of the paper published in the proceedings of the 2020 IEEE 14th International Conference on Semantic Computing (ICSC) [2].

2. Task Definition

The objective of our approach is to understand how laypersons teach new functions to intelligent systems. From a preliminary study we learned that utterances containing teaching sequences are usually composed of three semantic parts:

- Declaration: a declaration comprises an explicitly stated teaching intent, a name for the skill that is to be learned, and potentially parameters. Example: "[In order to]$_{intent}$ [set the table]$_{name}$ [for two]$_{parameter}$".
- Specification: a specification is the description of intermediate steps to realize the new functionality. Example: "[go to the cupboard]$_{action1}$ [open it]$_{action2}$ [and take out two plates]$_{action3}$".
- Miscellaneous: Any other types of statements that are irrelevant to understand the teaching effort. These include (but are not limited to) greetings, teaching of common sense knowledge or the environment, and observations. Example: "setting the table is important".

The individual parts may appear anywhere in the utterances. Furthermore, declarative parts might be split up or repeated (often with different wordings). The specification of intermediate steps is of variable length and non-contiguous in some cases. But most importantly, we observed that humans often struggle to express a teaching intent. Thus, many descriptions we examined can hardly be interpreted as a teaching effort; they instead merely state a sequence of actions.

Based on these observations, we define a two-level hierarchical classification task consisting of (see also Figure 1):

(1) First level (binary): classify whether an utterance contains a teaching intent and can thus be interpreted as an effort to teach a new function or not. Labels: *Teaching* and *Non-Teaching*, attached to entire descriptions.
(2) Second level (ternary): classify the semantic parts of a teaching sequence as defined above (only for utterances with a teaching intent). Labels: *Declaration*, *Specification*, and *Miscellaneous*, attached to each word in the description.

An alternative approach we considered was to drop the first classification level. In this case the absence of *declaration* labels would have indicated a missing teaching intent. However, since a single word in an utterance is misclassified easily, this would have produced many false positives. Thus, we expect a better overall classification performance with the hierarchical approach. Moreover, others argued in favor of hierarchical classification for similar tasks, e.g. Cohen et al. [3].

For both classification tasks we use machine learning approaches. Since the first classification task is a sequence-to-single-label task, classical machine learning approaches and neural networks are suitable. The second task is a typical sequence-to-sequence task. Thus, we focus on neural networks with an LSTM-like architecture, which have proven appropriate in tasks of that type.

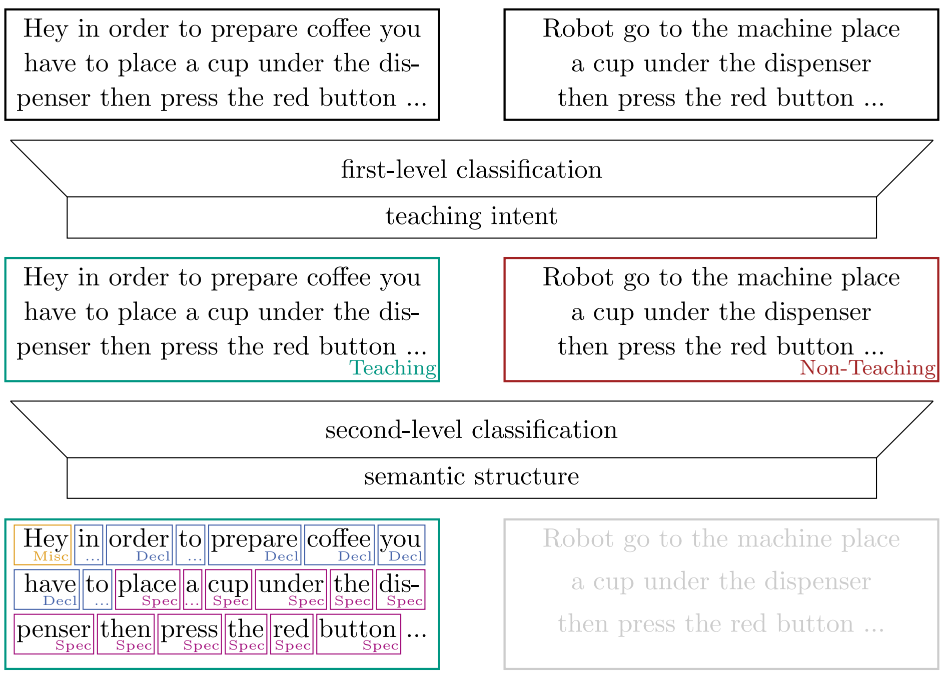

Hey in order to prepare coffee you have to place a cup under the dispenser then press the red button ...
Robot go to the machine place a cup under the dispenser then press the red button ...
first-level classification
teaching intent
Teaching
Non-Teaching
second-level classification
semantic structure
Misc
Decl
Spec

Fig. 1. Schematic overview of the two-leveled hierarchical classification task.

3. The project PARSE

The objective of the project *PARSE* (Programming ARchitecture for Spoken Explanations) [1] is to enable layperson to program intelligent systems using nothing but spoken natural language. To synthesize source code from spoken explanations, *PARSE* needs to interpret natural language. Thus, *PARSE* is a system for spoken language understanding (SLU), in the first place. Unlike most approaches, *PARSE* employs an agent-based architecture instead of a natural language processing (NLP) pipeline.

Agents analyze the natural language input concurrently and store their results in a shared (graph-based) knowledge representation. They perform tasks such as coreference analysis, control structure detection [4, 5], context analysis [6] or topic modeling [7]. Due to the parallel execution, agents may benefit from (intermediate) results of the other agents. For instance, the coreference agent resolves some references. Then the context agent use these to construct an initial model. The context model enables the coreference agent to resolve more references and so on. Agents can implement either probabilistic, knowledge- or rule-based approaches, depending on the task at hand. *PARSE*'s architecture is depicted in Figure 2. Besides the agent-based language understanding component, *PARSE* makes use of pre- and a post-processing pipelines. The first performs common NLP tasks and creates an ini-

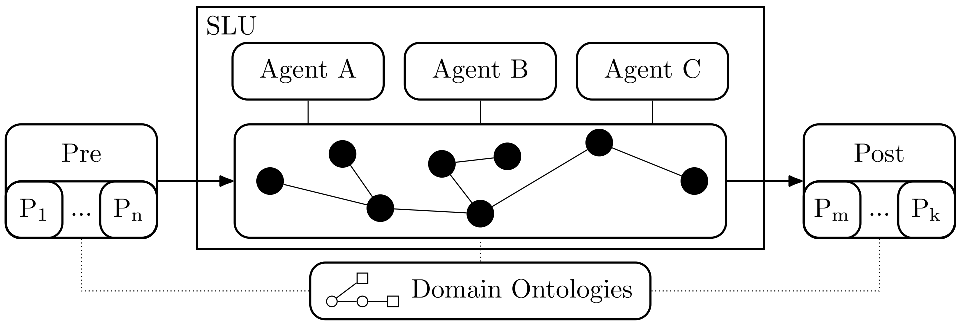


Fig. 2. The general architecture of *PARSE*.

tial knowledge graph. Since spoken language entails disfluencies and ungrammatical wording, *PARSE*'s pre-processing has to be error-tolerant. For this reason *PARSE* uses robust NLP techniques, such as shallow parsing.

To represent target systems, i.e. APIs, and system environments, *PARSE* makes use of ontologies. This approach makes *PARSE* (mostly) domain agnostic; systems and environments can be replaced with ease (without adjustments to other components, such as agents and pipelines). Actual source code is synthesized as follows. Elements of the enriched graph are mapped onto ontology elements, which generates pseudo code fragments. Afterwards, the graph is transformed into an AST. Based on this AST, *PARSE* is able to synthesize source code for most common programming languages, including Java, Python, and C. However, for the time being, *PARSE* generates scripts for one-time execution only. The approach presented in this article is a first step towards comprehensive end-user programming in natural language. Proceeding from analyzing teaching sequences, we aim to synthesize methods from spoken utterances.

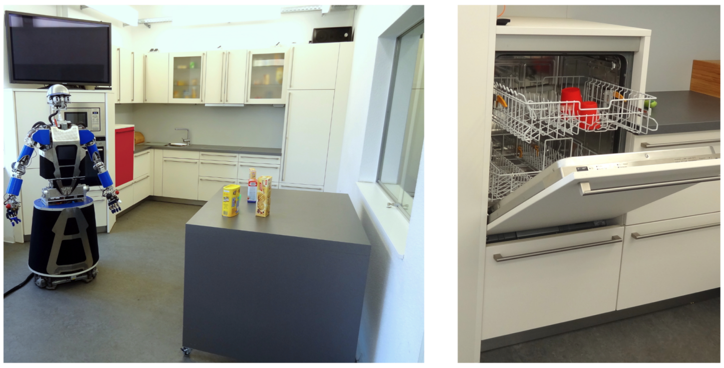
4. Dataset

The dataset we use to train, validate, and test the classifiers originates from a preliminary study. In this study, we examined how laypersons teach intelligent systems new functions by means of natural language instructions; we analyzed the language and structure used by subjects when describing new functionality. More precisely, we investigated, whether laypersons always clearly state that they wish to add new functionality and if so, whether the wish for extension (and the name of the new function) can be clearly separated from the actions that are to be performed. Furthermore, we studied the wording, e.g. the use of particular phrases to declare certain intentions and the presence of non-descriptive or meaningless statements. We used the online micro-tasking platform *Prolific*[a] to collect the data. Subjects

[a]Prolific: `https://www.prolific.co/`

New skill: Start the dishwasher
Intermediate steps:

- The dishwasher can be started by pressing the red button
- It is located in the kitchen
- It needs to be closed before starting it



Some exemplary instructions to teach this new skill:

- "Hi Armar, starting the dishwasher means you have to go to the dishwasher, close it and press the red button."
- "You have to close the dishwasher and press its red button for turning it on. That's how you start the dishwasher."

Fig. 3. The example scenario from the study including solutions.

were supposed to teach a humanoid robot new skills in four different scenarios:

(1) greeting someone (*greet*)
(2) preparing coffee (*coffee*)
(3) serving drinks (*drinks*)
(4) setting a table for two (*table*)

All of them take place in a kitchen setting but involve different objects and actions. For each scenario, we provided the subjects with a short name (of the function to be taught), a list of possible intermediate steps, and pictures depicting the setting. To familiarize the subjects with the task, we designed a short introduction including an exemplary scenario (*starting the dishwasher*). As shown in Figure 3 the exemplary scenario includes potential, valid solutions and emphasizes the components: a name for the new function (in blue), the explicit expression of the wish to teach something (in red) and intermediate steps (in green).

870 subjects participated in the study. We gathered 3168 descriptions, i.e. teaching sequences[b]. Table 1 depicts six exemplary descriptions. Besides the descriptions, we also gathered some personal information about the participants. First of all, women and men participated almost equally (49% females and 51% males). Most of

[b]Note that we have gathered textual submission for the sake of simplicity but encouraged the subjects to respond spontaneously.

Table 1. Six exemplary descriptions taken from different scenarios.

ID	scen.	description
302	1	Look directly at the person. Wave your hand. Say 'hello'.
1000	2	You have to place the cup under the dispenser and press the red button to make coffee.
1346	2	Making coffee means you have to press the red button, put a cup underneath the hole and then pouring the coffee that comes out into your cup
2180	3	To ring a beverage, open the fridge and select one of te beverages inside, pour it into one of the glasses on the kitchen counter and hand the glass over to the person.
2511	4	collect cutlery from cupboard, bring them to the table and place down neatly
2577	4	To set the table for two, Go to the cupboard and take two of each; plates, glasses, knives, and forks. Take them to the kitchen table and set two individual places.

Table 2. The number of descriptions, words used in total, and uniquely used words per scenario and in the entire dataset.

	descriptions	words (total)	words (unique)
scenario 1 (*greet*)	795	18205	566
scenario 2 (*coffee*)	794	26005	625
scenario 3 (*drinks*)	794	33001	693
scenario 4 (*table*)	785	31797	685
dataset	3168	109008	1469

them are native English speakers (60%). The majority of the participants are UK (31%) or US citizens (15%); the remainder is from European countries mainly. 70% have no programming experience at all. The age of the participants (at the time of participation) ranges from 18 to 76; the majority was 30 or younger (59%).

The 3168 teaching sequences we gathered in the preliminary study contain more than 109,000 words in sum. The participants used the most words to describe the third scenario (33001) and the least for the first (18205). In the mean, the subjects used nearly 71 unique words to shape their instructions (min: 2, max: 227). The dataset contains 1469 unique words altogether and about 642 unique words per scenario. This suggests, that there is not much overlap between the scenarios, which indicates a varying diction. Table 2 summarizes these dataset statistics.

The descriptions are of varying quality. A notable share contains syntactical flaws, e.g. typos, and grammar mistakes (see Table 1 for examples). For instance, description 2180 contains typos ("ring some beverage [...] te beverages inside").

They also vary in terms of descriptiveness and style; the latter ranges from full sentences to notes. Most of the participants slipped into the role of a teacher; some subjects however shifted their perspective to a naive end-user. Only a few used a rather technical language, as from a developer's perspective. The descriptions show the following general structure in the most cases:

- a brief greeting (e.g. "Hey robot [...]" or "Hi Armar [...]"),
- the *declaration* of the new function (e.g. "[...] to prepare coffee [...]"),
- and the *specification* of intermediate steps (e.g. "[...] you have to place a cup under the dispenser [...]").

However, the order of the semantically distinct parts varies in some descriptions Often the *declarative* part is uttered at the end (or even in between the *specification* of intermediate steps). Some subjects also repeated the declaration after specifying the steps, e.g. "[...] to prepare coffee [...] that's how you make some coffee").

Regarding the language, the descriptions show highly differing wording and syntax (even for the same scenario). This fact also entails different levels of abstraction among the descriptions. Some subjects detailed the intermediate steps up to the movement of the robot's arms, others gave rather abstract instructions. However, we observed that many of the subjects used the same language constructs to structure their descriptions. For instance, to verbalize a teaching intent, subjects often used gerunds (e.g. "preparing coffee") and to-infinitives (e.g. "to prepare coffee") in combination with particular phrases, such as "you have to" or "means". Other phrases that were regularly used to shape a wish to extend the functionality are:

- "[...] means you have to [...] (see description 1346 in Table 1)"
- "if you want to [...] you need to [...]"
- "we are going to learn how to [...]"
- "in order to [...] you have to [...]"

To quantify these observations, we extracted the most frequent n-grams ($n = [2; 4]$) from the dataset. In Figure 4 we depict the results for trigrams, which appeared to be most informative. Expectedly, most of the trigrams are domain specific, i.e. they include objects and events that can be attributed to the respective scenario. However, some phrases are generally valid; they are used to structure the description. For instance, the trigrams *you have to* and *you need to* separate the *declaration* from the *specification* (see submission 1000 and 1346 in Table 1). The first one is even the most frequent trigram in the dataset, the latter can be found at rank nine. Either one of these two phrases was used in 1327 descriptions (42%). This shows that certain wordings were commonly used by the subjects for this task.

We use the dataset as basis to solve the classification tasks. Therefore, we must label each description according to the scheme described in Section 2. The labeling was performed jointly by the first and the second author. In a first step, we attached the binary labels (*Teaching* and *Non-Teaching*). To attach labels for the ternary classification task, we only considered submissions that were labeled as

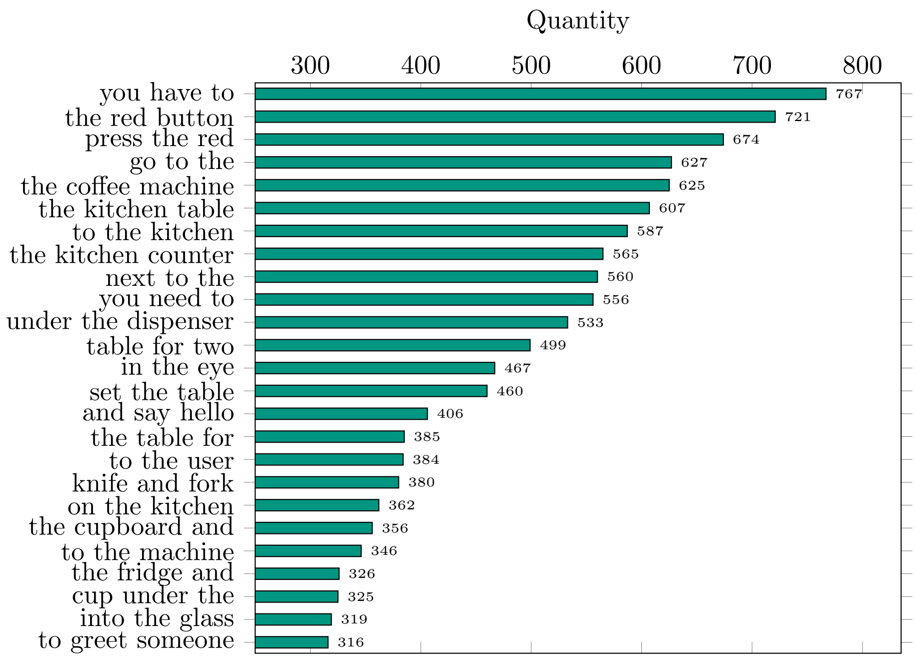


Fig. 4. Distribution of the 25 most frequent trigrams in the dataset.

Table 3. The distribution of the binary and ternary labels in the dataset.

	binary			ternary			
	Teach	*Non-Teach*	total	*Decl*	*Spec*	*Misc*	total
amount	1998	1170	3168	15559	57156	2219	74934
share	.63	.37	1.00	.21	.76	.03	1.00

Teaching previously. For each of them, we first determined the declarative parts (label *Declaration*). Then we separated the specification (label *Specification*) from all superfluous information (label *Miscellaneous*), e.g. greetings and common-sense teaching sequences. Table 3 depicts the total amount and share of the labels.

The analysis of the dataset revealed that more than one third (37%) of the descriptions do not contain an explicitly stated teaching intent (label *Non-Teaching*). Regarding the semantic structure of teaching sequences, the majority of words (76%) can be attributed to the specification of intermediate steps; more than a fifth (21%) account for verbalizations of teaching intents (wish for extension plus the name of the function). A negligible number of words are superfluous in our context (3%).

Both label sets are unequally distributed, which may affect the quality of the machine learning models. A one-sided shift often leads to over-fitted models that

Table 4. Statistics on the words per description.

				quantiles		
min.	max.	mean	st. dev.	.990	.995	.999
1	312	35.43	22.48	117	135	232

favor the dominating label, since this approach optimizes accuracy on the dataset. This threat concerns primarily the ternary classification task, in which the label *Specification* strongly dominates the other labels (76%).

Another factor that affects the machine learning approaches is the length of the natural language descriptions. In the study, we set no length restrictions. The responses of the subjects in the dataset consist of one to 312 words with a mean of 35.48 (see Table 4). Thus, the majority of descriptions is rather short; even the responses within the .995 quantile are no longer than 135 words[c]. The complexity of most machine learning models increases with maximum input length. Therefore, it might be beneficial to limit the input length. Since neural networks can only deal with input of fixed length, we have to define a maximum length anyways[d].

The dataset, including scenario descriptions, raw data, labeled data, and metadata is publicly available: `http://dx.doi.org/10.21227/zecn-6c61`

5. Understanding How Laypersons Teach New Functions

We aim to grasp the semantics of teaching sequences given by laypersons. In Section 2 we have defined a hierarchical classification task. To implement it, we first generate training instances (see Subsection 5.1). This involves pre-processing the dataset as well as extracting and pre-processing instances. Then, we describe the general approach to the classification task (see Subsection 5.2). Our approach is hierarchic. On the first level, we classify descriptions in terms of the existence of an explicitly stated teaching intent (see Subsection 5.3). The second classification task addresses the semantic structure of teaching sequences (see Subsection 5.4). Finally, we apply adaptations to improve the results (see Subsection 5.5).

5.1. *Generation of Training Instances*

According to Mihalcea [8] the generation of training instances involves three consecutive steps:

(1) Gathering and pre-processing the dataset
(2) Extraction of training instances
(3) Pre-processing of training instances

[c]Note that an input length of 135 words is still a lot compared to the state of the art. Most related approaches are limited to single instruction that hardly exceed ten words (see Section 6).
[d]For neural networks the maximum length of the input determines the size of the input layer.

Concerning the first step, we have already gathered the dataset (see Section 4). However, we must pre-process the data to meet the requirements of the machine learning toolkit and to maximize the overall quality. We perform the following actions during dataset optimization:

- Conversion to lower case, e.g. *Hello* $\rightarrow$ *hello*
- Recovering contractions, e.g. *don't* $\rightarrow$ *do not*
- Conversion of (cardinal) numbers, e.g. *1*st $\rightarrow$ *first*
- Deletion of enumerations, punctuation, and disfluencies
- Correction of typographical errors (but not grammatical mistakes), e.g. *thng* $\rightarrow$ *thing*

To extract the training instances, we can simply use all labeled descriptions from the dataset (see Section 4). Note that the pre-processing of the dataset has no effect on the number of training instances.

The pre-processing of the training instances primarily concerns the second-level instances. We create lemmatized and tokenized versions of the instances. Additionally, we prepare datasets with and without stop words. Finally, we map the instances and output labels to numeric values. The labels are simply mapped to one-hot vectors, while we transform the words to bag-of-words vectors and *fastText* word embeddings [9, 10]. We use two types of word embeddings: pre-trained embeddings generated by *Facebook Research* [11] on the Common Crawl dataset[e] and self-trained embeddings learned from the dataset. For the latter we tested three lengths: 50, 100, and 300. However, we used the last option only, since it produced the best results (at reasonable processing expense). Furthermore, the test results are comparable, since the pre-trained embeddings also have 300 dimensions. Since neural networks can process input with a fixed length only, we had to set a reasonable value. We limit the input length to 135 tokens as 99.5% of all descriptions in our dataset consist of 135 tokens or fewer (see Table 4).

5.2. *General Approach*

We used the *Python* libraries *scikit learn*, *keras*, and *tensorflow* to implement the classifiers. For our experiments we used two hardware configurations: a MacBook Pro with an Intel Core i5 (2.9 GHz) and 16 GB RAM and a PC with an Intel Core i7 (3.5 GHz) and 32 GB RAM.

For the first-level classification task, which is a sequence-to-single-label task, we decided to implement classical machine learning approaches and neural networks. We used the following classical classification approaches: Decision Tree, Random Forest, Support Vector Machine, Naïve Bayes, and Logistic Regression. The neural networks we implemented are of three different types: (basic) Artificial Neural Networks (ANN), Convolutional Neural Networks (CNN), and Recurrent Neural

[e]Common Crawl: `https://commoncrawl.org/`

Table 5. Types and architectures overview of neural networks used in both classification tasks.

types	architectures	additional layers
ANN		Flatten (Flat) Global max pooling 1D (GMax), Dense (D), Dropout(DO)
CNN		Max pooling 1D (Max), Global max pooling 1D (GMax), Dense (D), Dropout(DO)
RNN	LSTM GRU BiLSTM BiGRU	Dense (D), Dropout (DO)

Table 6. Overview of the hyper-parameters ranges tested for both classification tasks.

hyper-parameter	binary	ternary
epochs	300, 500, 1000	50, 100, 300
batch sizes	50, 100, 300, 400	32, 64, 100, 256, 300
number of units	10, 20, 32, 40, 50, 64, 100, 128, 150, 250, 256, 512	
dropout values	0.1, 0.2, 0.3	
learning rates	0.001, 0.0005	

Networks (RNN). We also implemented different architectures (e.g. LSTMs and GRUs), added further layers (e.g. dense and dropout layers), and varied the hyper-parameters (e.g. number of units and epochs). On the second level, we only implemented neural network approaches, since the problem is a typical sequence-to-sequence task. We used the same types and architectures, but varied the hyper-parameters. Table 5 depicts the neural network types and architectures we used for the first- and second-level classification task. Additionally, Table 6 lists the hyper-parameters we tested in the process. Note that CNNs take another parameter besides the number of units, the *convolution factor* for which we tested the values 3, 5, and 7.

We divided our dataset into train, validation, and test set. To split the data, we used two strategies: a random split and a scenario-based split. For the random split, we use the entire dataset and randomly divide it into training (80%) and test set (20%). We further divide the training set into training and validation set; again, we use an 80-20 split. The second split strategy selects one of the scenarios

Table 7. First-level classification accuracy achieved by the classical machine learning techniques on validation (in parenthesis) and test set.

	Random	Scenario
Decision Tree	(.893) .903	(.861) .719
Random Forest	(.917) .909	(.893) .374
Support Vector Machines	(.848) .861	(.870) .426
Naïve Bayes	(.771) .801	(.765) .300
Logistic Regression	(.927) .947	(.891) .719
Baseline (*ZeroR*)	.573	.547

(see Section 4) as test set; the remaining are used for training and validation, again with an 80-20 split. The rationale behind the scenario-based is as follows. If we use a whole scenario for testing, we can determine how the classifiers behave on unseen data that is conceptually different. All descriptions for a single scenario involve more or less the same actions and objects. However, they vary between the scenarios. Thus, with the scenario-based split we are able to measure how well a classifier learns teaching intent verbalizations and the general structure of teaching sequences.

5.3. *First-level Classification: Teaching Intent*

On the first level of our hierarchical classification task, we determine whether a description contains a teaching intent or not. The preliminary study has shown that subjects verbalize teaching intents quite differently. Often the intent is implicitly indicated or expressed by a single word only, e.g. "[do A] and [B] *to* prepare coffee". Therefore, the classification task is anything but straight forward.

As mentioned before, we implemented classical machine learning and neural network approaches. We present results for both and discuss the differences between the random and scenario-based dataset splits.

5.3.1. *Classical Machine Learning Techniques*

The input features for the classifiers are bag-of-words vectors and trigrams or quadrigrams. We used the tokenized and lemmatized dataset for training, validation, and test. However, all classifiers perform best on the lemmatized set. The same applies to stop words; their exclusion degrades results in all cases. Thus, we only report the results for the lemmatized set including stop words in Table 7. For all classifiers we show the accuracy on the validation set in parenthesis and the final results (test set) without. To provide a baseline, we depict the numbers of the so-called Zero-Rule classifier (*ZeroR*); it always predicts the majority class of the training set.

As expected, the baseline is rather similar for the random and scenario-based split. This indicates that our data is uniformly distributed. The results for the

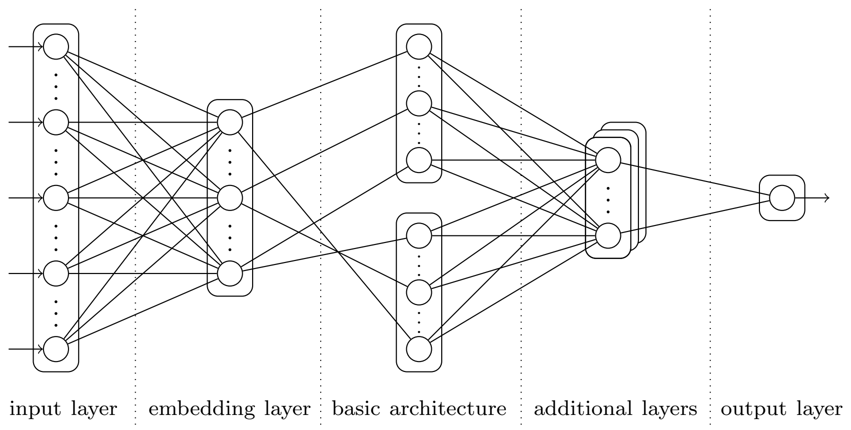


Fig. 5. A schematic illustration of the general network architecture.

classifiers vary greatly for the different splits. For the random split the accuracy on the validation and test set are similar. Not surprisingly, the elaborate approaches outperform the simple ones. The classifier that uses logistic regression achieves the best results. An accuracy of 94.7% on the test set is a surprisingly good result. However, the performance of all classifiers drastically declines if we use the scenario-based split. Three of five fall behind the baseline; the Naïve Bayes classifier labels only 30% of the instances correctly. The results for the Random Forest classifier show the problem plainly. It works well for the random split and is the best classifier on the validation set for the scenario-based split (89.3%). However, on the according test set its accuracy drops to 37.4%. Solely Decision Trees and Logistic Regression achieve acceptable accuracies (71.9%).

The results clearly show that classical machine learning approaches are insufficient for this task, since they oversimplify the classification problem and are unable to generalize to unseen data that is conceptually different.

5.3.2. *Neural Network Approaches*

For the neural networks we use word embeddings as input, either self-trained or pre-trained fastText embeddings (see Subsection 5.1). The general network structure as depicted in Figure 5 is composed of an input and an embedding layer, followed by the basic network architecture (e.g. LSTM), additional layers (e.g. Dense or Dropout layers) and an output layer.

We tested different batch sizes (see Table 6). However, no matter how we set the other hyper-parameters, we obtained the best results with a batch size of 100. The same applies to the question whether to use lemmatized or just tokenized input and stop words; in all cases the lemmatized dataset including stop words produced better results again.

Table 8. First-level classification accuracy achieved by neural networks on the *random* split; the results on the validation set are depicted in parenthesis and without for the test set.

Name	Configuration	self-trained	pre-trained
$\mathrm{ANN}_{1.0}$	Flat, D(10)	(.907) .911	(.874) .887
$\mathrm{ANN}_{1.1}$	Flat, D(100)	(.916) .914	(.846) .867
$\mathrm{ANN}_{2.0}$	GMax, D(10)	(.876) .887	(.872) .902
$\mathrm{ANN}_{2.1}$	GMax, D(100)	(.899) .896	(.879) .896
$\mathrm{CNN}_{1.0}$	C(128, 3), GMax, D(10)	(.947) .966	(.954) .963
$\mathrm{CNN}_{1.1}$	C(128, 5), GMax, D(10)	(.947) .971	(.930) .965
$\mathrm{CNN}_{1.2}$	C(128, 7), GMax, D(10)	(.952) .966	(.943) .962
$\mathrm{CNN}_{2.0}$	C(128, 3), Max(2), C(64, 3), GMax, D(10)	(.952) .959	(.952) .971
$\mathrm{CNN}_{2.1}$	C(128, 5), Max(2), C(64, 5), GMax, D(10)	(.949) .972	(.952) .966
$\mathrm{CNN}_{2.2}$	C(128, 5), Max(2), C(128, 5), GMax, D(10)	(.952) .964	(.954) .966
$\mathrm{CNN}_{2.3}$	C(128, 5), Max(5), C(128, 5), GMax, D(10)	(.956) .958	(.952) .959
$\mathrm{RNN}_{1.0}$	GRU(128)	(.560) .625	(.562) .625
$\mathrm{RNN}_{1.1}$	GRU(128), D(100)	(.562) .625	(.562) .625
$\mathrm{RNN}_{2.0}$	BiGRU(32), DO(0.2), D(64), DO(0.2)	(.947) .944	(.952) .959
$\mathrm{RNN}_{3.0}$	LSTM(64)	(.566) .631	(.568) .638
$\mathrm{RNN}_{3.1}$	LSTM(128)	(.570) .625	(.654) .738
$\mathrm{RNN}_{3.2}$	LSTM(128), D(100)	(.562) .625	(.562) .625
$\mathrm{RNN}_{4.0}$	BiLSTM(64), DO(0.2), D(64), DO(0.2)	(.947) .955	(.949) .955
$\mathrm{RNN}_{4.1}$	BiLSTM(64), DO(0.3), D(200), D(100)	(.941) .947	(.947) .949
$\mathrm{RNN}_{5.0}$	BiLSTM(128), D(64)	(.951) .955	(.956) .959
$\mathrm{RNN}_{5.1}$	BiLSTM(128), D(64), D(32)	(.945) .962	(.947) .955
$\mathrm{RNN}_{5.2}$	BiLSTM(128), D(100), DO(0.3), D(50)	(.936) .937	(.945) .941
$\mathrm{RNN}_{6.0}$	BiLSTM(256), D(128)	(.952) .944	(.945) .952
LogReg	–	(.927) .947	

For all other hyper-parameters we tested all possible combinations (as depicted in Table 5 and Table 6). However, in Table 8 and Table 9 we only present the best configurations per type, respective architecture. The first table depicts the results on the *random* data split, while the second considers the *scenario-based* split. Concerning the number of epochs, we observed that the best results are achieved at different points. Usually the networks need a few epochs only (less than 10) to converge. Also, the convergence can be predicted by means of the validation loss. We interrupt the training process when the validation loss stops to decrease, which is usually referred to as *early stopping.* Figure 6 shows the effect for RNN_4; the validation loss optimum is reached after epoch five.

The configurations in Table 8 and Table 9 can be read as follows. For the second last recurrent neural network ($\mathrm{RNN}_{5.2}$) we use a bidirectional LSTM (BiLSTM)

Table 9. First-level classification accuracy achieved by neural networks on the *scenario-based* split; the results on the validation set are depicted in parenthesis and without for the test set.

Name	Configuration	self-trained	pre-trained
$ANN_{1.0}$	Flat, D(10)	(.918) .759	(.897) .722
$ANN_{1.1}$	Flat, D(100)	(.905) .781	(.874) .715
$ANN_{2.0}$	GMax, D(10)	(.907) .766	(.905) .542
$ANN_{2.1}$	GMax, D(100)	(.893) .668	(.918) .674
$CNN_{1.0}$	C(128, 3), GMax, D(10)	(.962) .765	(.966) .854
$CNN_{1.1}$	C(128, 5), GMax, D(10)	(.973) .743	(.973) .776
$CNN_{1.2}$	C(128, 7), GMax, D(10)	(.973) .775	(.970) .897
$CNN_{2.0}$	C(128, 3), Max(2), C(64, 3), GMax, D(10)	(.968) .855	(.962) .874
$CNN_{2.1}$	C(128, 5), Max(2), C(64, 5), GMax, D(10)	(.969) .850	(.975) .859
$CNN_{2.2}$	C(128, 5), Max(2), C(128, 5), GMax, D(10)	(.973) .862	(.977) .862
$CNN_{2.3}$	C(128, 5), Max(5), C(128, 5), GMax, D(10)	(.962) .901	(.973) .801
$RNN_{1.0}$	GRU(128)	(.477) .299	(.519) .702
$RNN_{1.1}$	GRU(128), D(100)	(.519) .702	(.519) .702
$RNN_{2.0}$	BiGRU(32), DO(0.2), D(64), DO(0.2)	(.954) .911	(.958) .932
$RNN_{3.0}$	LSTM(64)	(.519) .702	(.519) .702
$RNN_{3.1}$	LSTM(128)	(.519) .702	(.519) .702
$RNN_{3.2}$	LSTM(128), D(100)	(.519) .702	(.519) .702
$RNN_{4.0}$	BiLSTM(64), DO(0.2), D(64), DO(0.2)	(.956) .896	(.962) .916
$RNN_{4.1}$	BiLSTM(64), DO(0.3), D(200), D(100)	(.947) .884	(.956) .911
$RNN_{5.0}$	BiLSTM(128), D(64)	(.960) .927	(.962) .919
$RNN_{5.1}$	BiLSTM(128), D(64), D(32)	(.950) .919	(.966) .898
$RNN_{5.2}$	BiLSTM(128), D(100), DO(0.3), D(50)	(.937) .922	(.954) .917
$RNN_{6.0}$	BiLSTM(256), D(128)	(.954) .843	(.962) .912
LogReg	–	(.891) .719	

architecture with 128 units. Additionally, the network is composed of further layers: Dense (with 100 units), Dropout (with a dropout value of 0.3), and another Dense.

For the random split (see Table 8) most neural networks achieve sufficient results. Only *unidirectional* LSTMs and GRUs fall behind considerably and seem insufficient for the task ($RNN_{1.*}$ and $RNN_{3.*}$). However, their *bidirectional* counterparts obtain excellent results; $RNN_{5.0}$ is even the best regarding the validation accuracy using pre-trained embeddings. On the test set, CNNs outperform all others; $CNN_{2.1}$ and $CNN_{2.2}$ show the overall best performance (96.6% using pre-trained embeddings). The majority of neural network configurations work best with pre-trained embeddings (except for the ANNs). A comparison to the baseline reveals that only CNNs and bidirectional RNNs (plus $ANN_{1.0}$) outperform the Logistic Regression classifier. The results for the random split suggest, that neural network

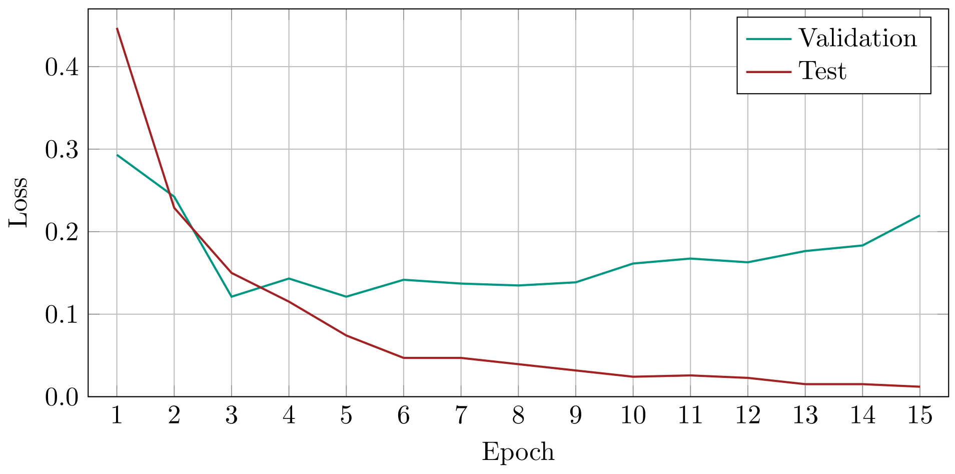


Fig. 6. The validation and training loss for $\mathrm{RNN}_{5.0}$ (BiLSTM(128), D(64)).

approaches are the most suitable for the task, but must be configured with care.

Either way, the consideration of the scenario-based split is more informative (see Table 9), since it models the realistic deployment of a classifier[f]. The results show that the accuracy of all neural networks decreases on the test set. This outcome was to be expected, since the subjects used a different vocabulary and wordings in the test scenario. However, the magnitude of decline differs considerably. The ANNs show the heaviest decline. That indicates that simple ANNs tend to over-fit to the training instances, i.e. they solely memorize a previously seen wording. The observation that ANNs perform worse using pre-trained embeddings for the random split speaks for this assumption, too. The CNNs (which showed the best accuracy on the random split) also deteriorate sharply (despite outstanding accuracies on the validation sets). The bidirectional RNNs show the best performances; $\mathrm{RNN}_{2.0}$ reaches test set accuracy levels (over 93%). It also performs best on pre-trained embeddings, which is to be expected, since the test set comprises previously unseen vocabulary. Contrary to expectations, the other bidirectional RNNs show their best results on self-trained embeddings. A possible cause may be that the advanced network architectures actually focus on the wordings that constitute a teaching intent, e.g. "... means you have to ...".

Unfortunately, the validation accuracy is hardly a good predictor for the test accuracy. The classifier with the best accuracy on the test set ($\mathrm{RNN}_{2.0}$) shows only a mediocre validation accuracy; the neural network with the best validation accuracy ($\mathrm{CNN}_{2.2}$) is ranked in the bottom half regarding test set results.

[f]Usually, classifiers are trained on existing datasets and then used for new and potentially differing input. With a hold-out scenario these conditions are reasonably simulated.

Table 10. Second-level classification accuracy achieved by neural networks on the *random* split; the results on the validation set are depicted in parenthesis and without for the test set.

Name	Configuration	self-trained	pre-trained
$ANN_{1.0}$	-	(.851) .855	(.851) .856
$ANN_{2.0}$	D(10)	(.848) .857	(.852) .849
$ANN_{2.1}$	D(100)	(.853) .856	(.853) .848
$RNN_{1.0}$	LSTM(64)	(.977) .976	(.979) .978
$RNN_{1.1}$	LSTM(128)	(.974) .976	(.978) .977
$RNN_{2.0}$	LSTM(128), DO(0.2)	(.976) .977	(.977) .977
$RNN_{2.1}$	LSTM(128), DO(0.4)	(.976) .977	(.979) .979
$RNN_{2.2}$	LSTM(128), D(64)	(.973) .972	(.977) .976
$RNN_{3.1}$	BiLSTM(128)	(.986) .983	(.987) .985
$RNN_{3.2}$	BiLSTM(128), D(64)	(.980) .983	(.985) .984
$RNN_{3.3}$	BiLSTM(128), D(100), DO(0.3), D(50)	(.982) .982	(.982) .985
$RNN_{3.4}$	BiLSTM(128), DO(0.2)	(.985) .984	(.988) .988
$RNN_{3.5}$	BiLSTM(128), DO(0.4)	(.985) .986	(.986) .986
$RNN_{4.0}$	BiLSTM(256), DO(0.2)	(.986) .984	(.987) .985
$RNN_{5.0}$	BiGRU(128)	(.984) .984	(.985) .985
ZeroR	–	.759	

5.4. *Second-level Classification: Semantic Structure*

On the second level of our hierarchical classification task, we determine the semantic structure of teaching sequences. We assume that they are composed of three parts: a declarative part that expresses the teaching intent and the name of the new skill, a specifying part that comprises the intermediate steps, and miscellaneous parts that are irrelevant for the task. The preliminary study has shown that these parts occur anywhere in an utterance and are potentially non-sequential.

For this task we waive the classical machine learning approaches. We assumed that the sequence-to-sequence labeling task is too complex for the classical approaches and pre-tests confirmed this assumption.

The input (word embeddings) and general network layouts are the same as for the first-level classification. The tested hyper-parameters differ slightly (see Table 5), due to the changed boundary conditions of this task (see Section 2). For this task a batch size of 32 proved to be best performing. Also, the results are best for the tokenized (non-lemmatized) dataset. However, we still do not exclude stop words. The unbalanced dataset poses a challenge; the class *Specification* clearly dominates (see Table 3). In return, the *Zero-Rule* classifier becomes a strong baseline.

In Table 10 we report the results of the best performing neural networks for the *random* dataset split; Table 11 shows the results for the *scenario-based* split. We again distinguish results using self-trained versus pre-trained fastText embeddings. Overall, the results are promising. All approaches outperform the baseline clearly.

Table 11. Second-level classification accuracy achieved by neural networks on the *scenario-based* split; the results on the validation set are depicted in parenthesis and without for the test set.

Name	Configuration	self-trained	pre-trained
$ANN_{1.0}$	-	(.850) .779	(.851) .826
$ANN_{2.0}$	D(10)	(.850) .825	(.851) .826
$ANN_{2.1}$	D(100)	(.851) .822	(.851) .827
$RNN_{1.0}$	LSTM(64)	(.971) .960	(.975) .966
$RNN_{1.1}$	LSTM(128)	(.973) .960	(.973) .964
$RNN_{2.0}$	LSTM(128), DO(0.2)	(.970) .960	(.973) .966
$RNN_{2.1}$	LSTM(128), DO(0.4)	(.971) .959	(.974) .967
$RNN_{2.2}$	LSTM(128), D(64)	(.970) .955	(.971) .963
$RNN_{3.1}$	BiLSTM(128)	(.983) .960	(.981) .976
$RNN_{3.2}$	BiLSTM(128), D(64)	(.973) .960	(.979) .965
$RNN_{3.3}$	BiLSTM(128), D(100), DO(0.3), D(50)	(.978) .955	(.981) .968
$RNN_{3.4}$	BiLSTM(128), DO(0.2)	(.982) .958	(.981) .975
$RNN_{3.5}$	BiLSTM(128), DO(0.4)	(.980) .961	(.980) .973
$RNN_{4.0}$	BiLSTM(256), DO(0.2)	(.982) .964	(.982) .975
$RNN_{5.0}$	BiGRU(128)	(.976) .955	(.982) .968
ZeroR	–	.757	

However, no CNN is among the best fifteen and ANNs performs considerably worse (more than 10%) than the remaining; the RNNs dominate this task. In particular, the bidirectional RNNs obtain surprisingly good results. The classification accuracy of the best configuration ($RNN_{3.4}$) for the random split using pre-trained embeddings is 98.8%. Encouragingly, the results for the scenario split are almost on the same level. Four RNNs exceed 97% using pre-trained embeddings; $RNN_{3.1}$ performs best with an accuracy of 97.6%. However, there are only small differences between the configurations. Thus, bidirectional RNNs seem to be suitable for this task in general.

5.5. *Adaptations*

We implemented two task-based adaptations to improve the classification results heuristically; the first concerns the binary and the second the ternary classification. For both we use the best neural network configuration as basis (based on the mean results): $RNN_{2.0}$ for the first task and $RNN_{3.1}$ for the second.

The first adaptation works as follows. We perform the first-level classification as usual. However, we observed that the binary classifiers struggle to separate the classes from time to time. Therefore, we adjust the class allocation. Originally, the classifiers assign the label *Non-Teaching* to all values in the range [0;0.5] and *Teaching* to (0.5;1]. We experimented with alternative separation values; we tested all in the range [0,0.5] (steps: 0.05). The results are illustrated in Figure 7; the baseline

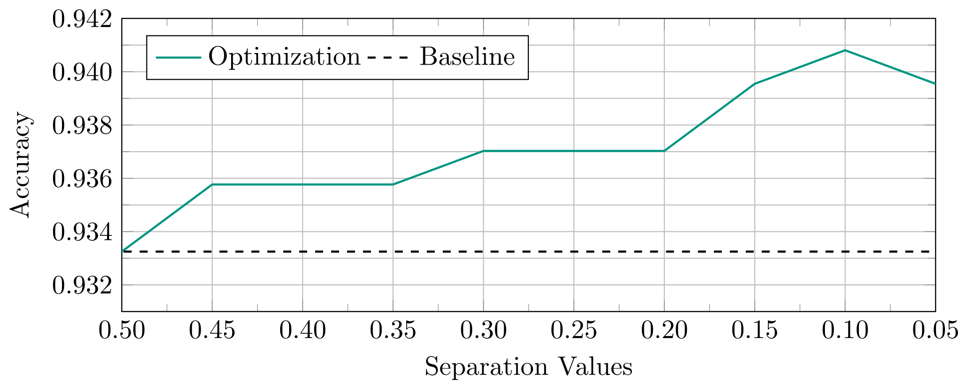


Fig. 7. Accuracies (on the test set) obtained for different (optimized) separation values in the output layer of $\text{RNN}_{2.0}$ for the first-level classification (binary).

is $\text{RNN}_{2.0}$ with an unchanged separation value (0.5). We obtain the best results with a separation value of 0.1 (accuracy: 94.1%, plus 0.8% for the scenario-based split). In a second step we use the ternary classification. We apply it to all descriptions (not only those labeled as *Teaching* on the first level). Then, we review the binary result and alter the class of all instances to *Teaching* that have a classification value in the range of $[m, 0.1)$ and at least n *Declaration*-labels. The rationale behind this approach is as follows. The presence of *Declaration*-labels suggests that the description is a teaching effort (first-level classification label *Teaching*). Again we tested different values for m (range: [0.001; 0.1], steps: 0.001) and n (range [0; 6]); the results are depicted in Figure 8[g]. Two configurations obtain the best result: $(m = 0.008; n = 2)$, the green graph, and $(m = 0.006; n = 5)$, the purple graph. Using one of them, the accuracy of the binary classification increases to 95.5% for the scenario-based split (plus 2.2 percentage points).

The second adaptation uses linguistic information to generate continuous semantic parts (second-level classification). For our heuristic we employ the semantic role labeling tool SENNA [12]; Figure 9 illustrates the approach. We interpret the roles as chunks (and ignore their semantics) and merge these chunks with the output of the second-level classifier as follows. For most cases we use a simple majority decision. This means, the heuristic attaches the dominating label to all words of the chunk. If there is a draw, we take the first word left of the chunk into account and if there is no left word we consider the first to the right. Whenever there is neither a word left nor right and the chunk contains *Specification*-labels, we attach this label to all words. We evaluated the heuristic using $\text{RNN}_{3.1}$ and the scenario-based data split. Unfortunately, the classification accuracy decreases (minus 0.86%, relative). However, the loss is small. In return, the adapted classification ensures that all se-

[g]We only show the results for $m = [0.001; 0.03]$, since accuracy values are best in this range.

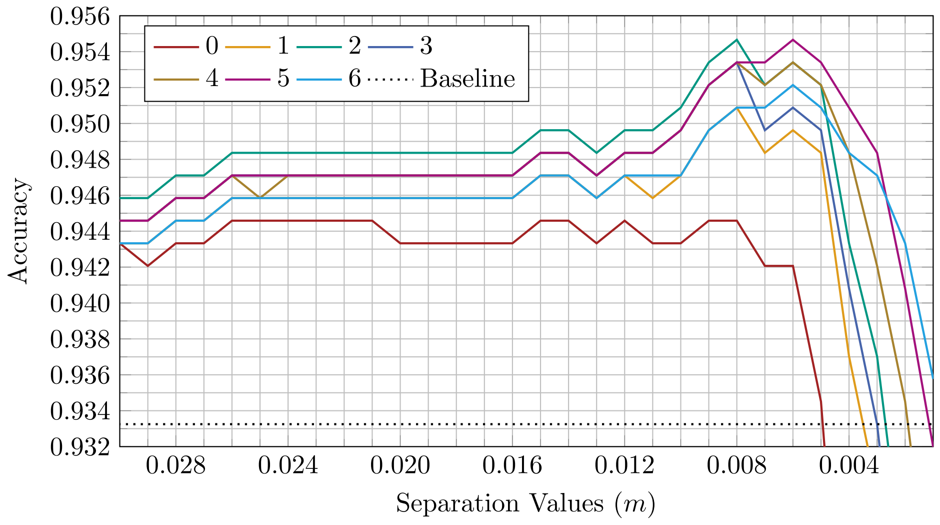


Fig. 8. Accuracies (on the test set) obtained for different adaption configurations for the first-level classification (binary). A configuration consists of a separation value (m) and a minimum number of *Declaration* labels ($n = [0; 6]$).

Utterance:	*to*	*prepare*	*coffee*	*place*	*an*	*empty*	*cup*	...
Prim. Pred.:	DECL	DECL	DECL	SPEC	SPEC	ELSE	SPEC	
SRL Labels:	0	V	A1	V	A1	A1	A1	
Adap. Pred.:	DECL	DECL	DECL	SPEC	SPEC	SPEC	SPEC	

Fig. 9. Schematic illustration of the adaptation of the second-level classification.

mantic parts are continuous. For instance, in the utterance "take a cup that is next to a machine", RNN$_{3.1}$ attaches the label *Specification* to all words but *is* and *to*. These two receive the label *Declaration* (which is incorrect here). The adaptation alters both labels to *Specification*.

Continuous semantic parts are an essential precondition for most applications, such as the synthesis of methods. At the same time, the misclassifications introduced by the adaptations are negligible, since they mainly concern words of little relevance, such as conjunctions.

6. Related Work

Over the years, the objective of programming with natural language has been viewed from different perspectives: Some approaches think of it as code dictation, others try to naturalize programming languages. Interactive systems rely on user feedback to solve the task, while others employ semantic parsing. For research in the field of humanoid robotics, programming with natural language is of particular importance. Each perspective focuses on different aspects and addresses the task of teaching new skills differently.

Approaches for code dictation are basically natural language interfaces to code editors. Developers dictate code and the text (or speech) is literally converted into code. Thus, no semantic transformation or mapping is necessary. However, the respective parsers (and automatic speech recognition systems) are tailored to preferably recognize code-like terms. Natural Java by Price et al. uses case frame grammars for Java source code dictation [13]. They use information retrieval techniques to fill the roles in the frames. Begel and Graham present Spoken Java, a voice based code dictation interface for Java [14, 15]. According to the authors it is supposed to be used by developers that can not use their hands due to injuries, e.g. repetitive strain injuries. VoiceCode by Désilets et al. allows dictating different programming languages [16]. With all approaches new methods can be dictated just like anything else. However, users have to dictate proper source code.

The approach to naturalize programming by Wang et al. is set in a voxel world called Voxelurn [17]. Users may define new aliases for API methods to naturalize the vocabulary used. The approach also offers the composition of calls. The aliases of composed calls constitute newly learned functions.

Other approaches are interactive; they synthesize source code in dialog with the user. They are designed for laypersons or programming novices. Most of them make use of mixed or user initiative dialog to clarify ambiguous or unclear input. Metafor by Liu and Lieberman constructs program skeletons from English prose [18]. They use a specialized parser that creates code-like subject-verb-object-object structures. The results are classes, attributes, method signatures, but no runnable code. The follow-up work by Mihalcea et al. is able to create runnable code including control structures; they also detect comments [19]. Landhäußer et al. additionally reconstruct timelines [20]. However, their tool NLCI provides marginal user feedback only. Le et al. enable users to create short scripts for smartphones with SmartSynth [21]. The scripts are synthesized with the help of heuristics on syntactical features. The input is limited to the following structure: a condition followed by a sequence of actions. SmartSynth uses type inference to fill gaps in method calls, e.g. missing parameters. If a script is invalid the user is queried for clarification.

Another perspective was recently introduced by the semantic parsing community. Semantic parsing denotes the task of mapping natural language to logical forms. Recently, source code is considered as one logical forms. Even though scripts can be synthesized, integrating new functions are not considered so far. Guu et al. use reinforcement learning in combination with the maximal marginal likelihood method to map natural language to code [22]. Rabinovich et al. use an AST-like structured BiLSTM to infer ASTs from textual descriptions [23] and Chen et al. use recurrent neural networks to learn so-called action embeddings [24]. Dong and Lapata use a two-tiered approach; first producing a light-weight, coarse meaning representation and then using a BiLSTM to fill in missing details [25].

Teaching new functionality to intelligent systems is of particular interest in the robotic domain. The robotic systems of the future are supposed to act like humans. Thus, they have to be able to understand task descriptions for humans. Most approaches aim at synthesizing actions plans or new functions (composed of single actions). Lincoln and Verres use a planning approach to model the shared goals and intents of users and machines [26]. New functionality can be taught but the used language is rather technical. The approach by She et al. allows the usage of everyday language to teach a robotic system new functionality [27]. For the transformation the approach uses semantic parsing. Even though the approach does not expect technical terms, the vocabulary and wordings are restricted. Markievicz et al. use descriptions that were originally created to teach humans [28]. They use dependency parsing and specialized semantic role labeling to map the natural language input to robotic instructions. Their approach assumes that the input consists of known instructions and thus is unable to cope with newly introduced functionalities.

7. Conclusion & Future Work

Natural language will be the key to effortless end-user programming. To make programming in natural language a truly creative process, users must be empowered to create new functions using spoken instructions. As a first step towards this goal, we have presented a hierarchical classification task to grasp the semantics of natural language teaching sequences. The first classification level determines whether an utterance constitutes an effort to teach a new function. The second analyzes the semantic structure of teaching efforts and divides them into three distinct parts: a declarative part that contains the teaching intent with a name for the new function, a specifying part that states the intermediate steps, and superfluous information.

For both tasks we implemented a broad range of machine learning approaches. However, neural networks outperform the classical approaches in almost all cases. In particular, bidirectional RNNs excel in both tasks. Even if we expose them to input that is conceptually different to the training instances, they are highly accurate. In this setting (scenario-based data split), the best classifier for the first task, a BiGRU, obtains an accuracy of 93.2%; for the second it is even 97.6% (BiLSTM).

Additionally, we implemented two heuristic improvements. With the first, we overrule the first-level classification if the second classifier disagrees and the first was uncertain; this heuristic improves the accuracy by 2.2%. With the help of the second heuristic we make sure that semantic parts are continuous.

As the next step we plan to synthesize actual methods based on the classification results. We will construct method signatures from *specifying* phrases and bodies from *declarative* parts. In order to implement that, we may have to refine the label set of the second-level task, e.g. define a label for the name of the function or parameters. Furthermore, we plan to evaluate our approach on other datasets; we may use open-access corpora or even carry out another online study.

References

[1] S. Weigelt and W. F. Tichy, ProNat: An Agent-based System Design for Programming in Spoken Natural Language, in *Proceedings of the 37th International Conference on Software Engineering - Volume 2 ICSE '15*, (IEEE Press, Piscataway, NJ, USA, 2015), pp. 819–820.

[2] S. Weigelt, V. Steurer, T. Hey and W. F. Tichy, Roger that! Learning How Laypersons Teach New Functions to Intelligent Systems, in *2020 IEEE 14th International Conference on Semantic Computing (ICSC)* February 2020, pp. 93–100.

[3] S. Cohen, L. Rokach and O. Maimon, Decision-tree Instance-space Decomposition with Grouped Gain-ratio, *Inf. Sci.* **177** 3592–3612 (September 2007).

[4] S. Weigelt, T. Hey and V. Steurer, Detection of Conditionals in Spoken Utterances, in *2018 IEEE 12th International Conference on Semantic Computing (ICSC)* January 2018, pp. 85–92.

[5] S. Weigelt, T. Hey and V. Steurer, Detection of Control Structures in Spoken Utterances, *International Journal of Semantic Computing* **12** 335–360 (September 2018).

[6] S. Weigelt, T. Hey and W. F. Tichy, Context Model Acquisition from Spoken Utterances, in *Proceedings of The 29th International Conference on Software Engineering & Knowledge Engineering* (Pittsburgh, PA, 2017), pp. 201–206.

[7] S. Weigelt, J. Keim, T. Hey and W. F. Tichy, Unsupervised Multi-Topic Labeling for Spoken Utterances, in *2019 IEEE International Conference on Humanized Computing and Communication (HCC)* September 2019, pp. 38–45.

[8] R. Mihalcea, Using Wikipedia for Automatic Word Sense Disambiguation, in *Human Language Technologies 2007: The Conference of the North American Chapter of the Association for Computational Linguistics; Proceedings of the Main Conference* (Association for Computational Linguistics, Rochester, New York, April 2007), pp. 196–203.

[9] P. Bojanowski, E. Grave, A. Joulin and T. Mikolov, Enriching Word Vectors with Subword Information, *Transactions of the Association for Computational Linguistics* **5** 135–146 (December 2017).

[10] A. Joulin, E. Grave, P. Bojanowski and T. Mikolov, Bag of Tricks for Efficient Text Classification, in *Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics: Volume 2, Short Papers* (Association for Computational Linguistics, Valencia, Spain, April 2017), pp. 427–431.

[11] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch and A. Joulin, Advances in Pre-Training Distributed Word Representations, in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)* (European Language Resources Association (ELRA), Miyazaki, Japan, May 2018).

[12] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu and P. Kuksa, Natural Language Processing (Almost) from Scratch, *J. Mach. Learn. Res.* **12** 2493–2537 (November 2011).

[13] D. Price, E. Riloff, J. Zachary and B. Harvey, NaturalJava: A Natural Language Interface for Programming in Java, in *Proceedings of the 5th International Conference on Intelligent User Interfaces IUI '00*, (ACM, New Orleans, Louisiana, USA, 2000), pp. 207–211.

[14] A. Begel, Spoken Language Support for Software Development, in *Proceedings of the 2004 IEEE Symposium on Visual Languages - Human Centric Computing VLHCC '04*, (IEEE Computer Society, USA, September 2004), pp. 271–272.

[15] A. Begel and S. L. Graham, Spoken Programs, in *Proceedings of the 2005 IEEE Symposium on Visual Languages and Human-Centric Computing VLHCC '05*, (IEEE Computer Society, USA, September 2005), pp. 99–106.

[16] A. Désilets, D. C. Fox and S. Norton, VoiceCode: An Innovative Speech Interface for Programming-by-voice, in *CHI '06 Extended Abstracts on Human Factors in Computing Systems CHI EA '06*, (ACM, New York, NY, USA, 2006), pp. 239–242.

[17] S. I. Wang, S. Ginn, P. Liang and C. D. Manning, Naturalizing a Programming Language via Interactive Learning, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Association for Computational Linguistics, Vancouver, Canada, July 2017), pp. 929–938.

[18] H. Liu and H. Lieberman, Metafor: Visualizing Stories as Code, in *IUI '05: Proceedings of the 10th International Conference on Intelligent User Interfaces* (ACM, 2005), pp. 305–307.

[19] R. Mihalcea, H. Liu and H. Lieberman, NLP (Natural Language Processing) for NLP (Natural Language Programming), in *Proceedings of the 7th International Conference on Computational Linguistics and Intelligent Text Processing CICLing'06*, (Springer-Verlag, Berlin, Heidelberg, 2006), pp. 319–330.

[20] M. Landhäußer, S. Weigelt and W. F. Tichy, NLCI: A Natural Language Command Interpreter, *Automated Software Engineering* **24** 839–861 (December 2017).

[21] V. Le, S. Gulwani and Z. Su, SmartSynth: Synthesizing smartphone automation scripts from natural language, in *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services - MobiSys '13* (ACM Press, Taipei, Taiwan, 2013), p. 193.

[22] K. Guu, P. Pasupat, E. Liu and P. Liang, From Language to Programs: Bridging Reinforcement Learning and Maximum Marginal Likelihood, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Association for Computational Linguistics, Vancouver, Canada, July 2017), pp. 1051–1062.

[23] M. Rabinovich, M. Stern and D. Klein, Abstract Syntax Networks for Code Generation and Semantic Parsing, in *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics* (Association for Computational Linguistics, 2017), pp. 1139–1149.

[24] B. Chen, L. Sun and X. Han, Sequence-to-Action: End-to-End Semantic Graph Generation for Semantic Parsing, in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Association for Computational Linguistics, Melbourne, Australia, July 2018), pp. 766–777.

[25] L. Dong and M. Lapata, Coarse-to-Fine Decoding for Neural Semantic Parsing, in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)* (Association for Computational Linguistics, Melbourne, Australia, July 2018), pp. 731–742.

[26] N. K. Lincoln and S. M. Veres, Natural Language Programming of Complex Robotic BDI Agents, *Journal of Intelligent & Robotic Systems* **71** 211–230 (September 2012).

[27] L. She, Y. Cheng, J. Y. Chai, Y. Jia, S. Yang and N. Xi, Teaching Robots New Actions through Natural Language Instructions, in *The 23rd IEEE International Symposium on Robot and Human Interactive Communication* (IEEE, Edinburgh, UK, August 2014), pp. 868–873.

[28] I. Markievicz, M. Tamosiunaite, D. Vitkute-Adzgauskiene, J. Kapociute-Dzikiene, R. Valteryte and T. Krilavicius, Reading Comprehension of Natural Language Instructions by Robots, in *Beyond Databases, Architectures and Structures. Towards Efficient Solutions for Data Analysis and Knowledge Representation* (Springer, May 2017), pp. 288–301.