Roger that! Learning How Laypersons Teach New Functions to Intelligent Systems.

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

***Abstract*—Intelligent systems are rather smart today but still limited to built-in functionality. To break through this barrier, future systems must allow users to easily adapt the system by themselves. For humans the most natural way to communicate is talking. But what if users want to extend the systems' functionality with nothing but natural language? Then intelligent systems must understand how laypersons teach new skills.**

To grasp the semantics of such teaching sequences, we have defined a hierarchical classification task. On the first level, we consider the existence of a teaching intent in an utterance; on the second, we classify the distinct semantic parts of teaching sequences: declaration of a new function, specification of intermediate steps, and superfluous information.

We evaluate twelve machine learning techniques with multiple configurations tailored to this task ranging from classical approaches such as naïve-bayes to modern techniques such as bidirectional LSTMs and task-oriented adaptations. On the first level convolutional neural networks achieve the best accuracy (96.6%). For the second task, bidirectional LSTMs are the most accurate (98.8%). With the additional adaptations we are able to improve both classifications distinctly (up to 1.8%).

I. Introduction

Intelligent systems are everyday companions nowadays. Users easily arrange appointments or check their emails with virtual assistants such as Apple's Siri or Google Assistant. Humanoid robots or home automation systems provide conversational interfaces. However, the full potential of intelligent systems is not yet exploited. For the time being, users can merely access built-in functionality. All too soon, users will not only expect to *use* an intelligent system but *extend* its functionality. They will expect to implement new functionality with little effort, ideally using nothing but natural language. Thus, future intelligent systems must understand how laypersons teach new functionality.

Up to now, this task is not well studied; it is unclear how to grasp the semantics of teaching sequences. To better understand how people verbalize teaching sequences we ran a preliminary study in which subjects were supposed to teach new skills to a humanoid robot. Each participant gave natural language descriptions for four scenarios. We were able to gather 3168 descriptions from 870 participants.

Based on the findings of the preliminary study we developed the following approach. We propose to decompose the task of understanding teaching sequences. The first objective is to understand, whether an utterance contains a teaching intent at all. If an utterance is a teaching effort, the second step is to extract the distinct semantic parts. We found that teaching efforts are usually composed of three semantic structures. The first is the verbalization of the teaching intent, e.g. "preparing a cup of coffee means [...]". Second, most teaching efforts include a description of intermediate steps to realize the new skill, e.g. "[...] put a coffee mug under the dispenser and then press the red button on the coffee machine [...]". Finally, a noteable subset contains statements that are irrelevant for teaching sequences, e.g. "Hello" or "coffee is a beverage that people like to drink".

We implemented a hierarchical classification that on the first level discovers utterances with teaching intent (binary). Secondly, it determines the semantic structures (ternary). For the binary classification task we implemented five basic machine learning approaches and three different types of neural networks (e.g. RNNs) with different architectures (e.g. GRUs, LSTMs, etc.); for the latter we tested a broad range of hyper-parameters. On the second level we also implemented three different types of neural networks; again we used multiple architectures and different hyper-parameters. Finally, we added a set of heuristics for the binary and the ternary classification task to improve the performance of our approach; they are tailored to the task but dataset-agnostic.

The remainder is structured as follows. First, we define the task in Section II before we introduce the dataset in Section III. In Section IV 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 V. Finally, we conclude our work in Section VI and discuss future work.

II. Task Definition

The objective of our approach is to understand how laypersons teach new functions to intelligent systems. From a pre-

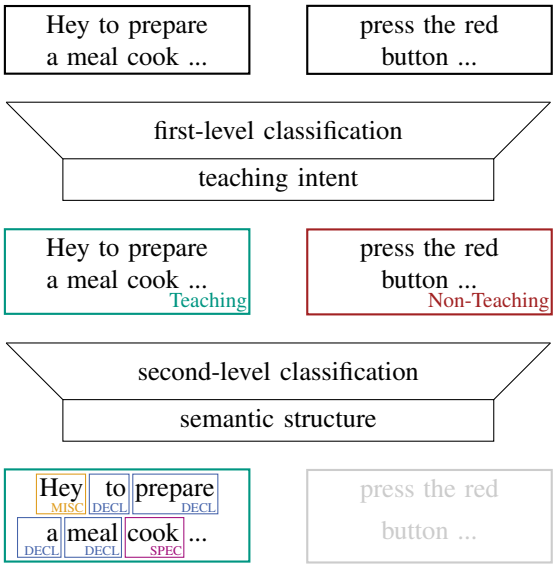

Fig. 1. Schematic overview of the two-leveled hierarchical classification task.

liminary 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.

TABLE I
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	795	18205	566
scenario 2	794	26005	625
scenario 3	794	33001	693
scenario 4	785	31797	685
dataset	3168	109008	1469

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 miss-classified 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. [1].

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.

III. DATASET

The dataset we use to train, validate, and test the classifiers originates from a preliminary study. A detailed discussion on the study and the dataset may be found in [2]. We used the online micro-tasking platform *Prolific*[1] for collecting the data. Subjects were supposed to teach a humanoid robot new skills in four different scenarios, such as greeting someone or preparing coffee. All of them take place in a kitchen setting but involve different objects and actions. 870 subjects participated in the study. We gathered 3168 teaching sequences with more than 109,000 words in sum. The subjects used 1469 unique words in their descriptions. In the mean, the subjects used 642 unique words per scenario. Thus, there is not much overlap between the scenarios, which indicates a varying diction. Table I summarizes these dataset statistics.

Besides the descriptions we also gathered some personal information about the participants. Most of them are native English speakers (60%) and 70% have no programming experience. Women and men participated almost equally (359 females and 366 males); their age (at the time of participation) ranges from 18 to 76. However, nearly 60% of the participants were 30 or younger.

The dataset is labeled according to the scheme described in Section II. 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*). Apart from that, the semantic parts can be clearly separated in almost all cases. Thus, the ternary labels for the second-level classification can

[1]Prolific: https://www.prolific.co/

TABLE II
THE DISTRIBUTION OF THE BINARY AND TERNARY LABELS IN THE DATASET.

		amount	share
binary	Teaching	1998	.63
	Non-Teaching	1170	.37
	Total	3168	1.00
ternary	Declaration	15559	.21
	Specification	57156	.76
	Miscellaneous	2219	.03
	Total	74934	1.00

TABLE III
STATISTICS ON THE WORDS PER DESCRIPTION.

				quantiles		
min.	max.	mean	st. dev.	.990	.995	.999
1.0	312	35.43	22.48	117	135	232

be attached unambiguously. Table II depicts the total amount and share of the labels.

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 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 III). Thus, the majority of descriptions is rather short; even the responses within the .995 quantile are not longer than 135 words. The complexity of most machine learning models increases with maximum input length[2]. 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.

IV. LEARNING HOW LAYPERSONS TEACH NEW FUNCTIONS TO INTELLIGENT SYSTEMS

We aim to grasp the semantics of teaching sequences given by laypersons using nothing but natural language. In Section II we have defined a hierarchical classification task. To implement it, we first generate training instances (see Subsection IV-A). 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 IV-B). Our approach is hierarchic. On the first level we classify whole descriptions in terms of the existence of an explicitly stated teaching intent (see Subsection IV-C). The second classification task addresses the semantic structure of teaching sequences (see Subsection IV-D). Finally, we apply some adaptations to improve the results (see Subsection IV-E).

[2]In particular neural network architectures are problematic as the maximum length of the input determines the size of the input layer.

A. Generation of Training Instances

According to Mihalcea [3] 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

Concerning the first step, we have already gathered the dataset (see Section III). 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 III). 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 stopwords. 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 word embeddings. We use two types of word embeddings: Facebook's fastText embeddings [4] 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 fastText 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 less (see Table III).

B. 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),

TABLE IV
OVERVIEW OF THE TYPES, ARCHITECTURES, AND HYPER-PARAMETERS OF NEURAL NETWORKS USED IN THE TWO CLASSIFICATION TASKS.

types	architectures	additional layers	number of units	epochs	batch sizes	dropout values	learning rates
ANN		Flatten (Flat), Global max pooling 1D (GMax), Dense (D), Dropout(DO)	10, 20, 32, 40, 50, 64, 100, 128, 150, 250 256, 512	binary: 300, 500, 1000	binary: 50, 100, 300, 400	0.1, 0.2, 0.3	0.001, 0.0005
CNN		Max pooling 1D (Max), Global max pooling 1D (GMax), Dense (D), Dropout(DO)		ternary: 50, 100 300	ternary: 32, 64, 100, 256, 300		
RNN	LSTM GRU BiLSTM BiGRU	Dense (D), Dropout (DO)					

Convolutional Neural Networks (CNN), and Recurrent Neural 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 IV depicts the neural network configurations we used for the first- and second-level classification task. 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 a 80-20 split.

The second split strategy selects one of the scenarios (see Section III) as test set; the remaining are used for training and validation, again with a 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.

C. 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.

TABLE V
FIRST-LEVEL CLASSIFICATION ACCURACY ACHIEVED BY THE CLASSICAL MACHINE LEARNING TECHNIQUES ON VALIDATION (IN BRACKETS) AND TEST SET. THE BEST RESULTS ARE PRINTED IN BOLD TYPE.

	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 (MFL)	.573	.547

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. Therefore, we only report the results for the lemmatized set including stop words in Table V. For all classifiers we show the accuracy on the validation set in brackets and the final (test set) results without brackets. The best results are printed in bold type. To provide a baseline we additionally depict the numbers of a classifier that always classifies the most frequent label (MFL) for each instance (usually referred to as *Zero-Rule* classifier).

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 classifiers vary greatly for the different splits. For the random split the accuracy on the validation and test set are similar. Not suprisingly, 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 senario-based split

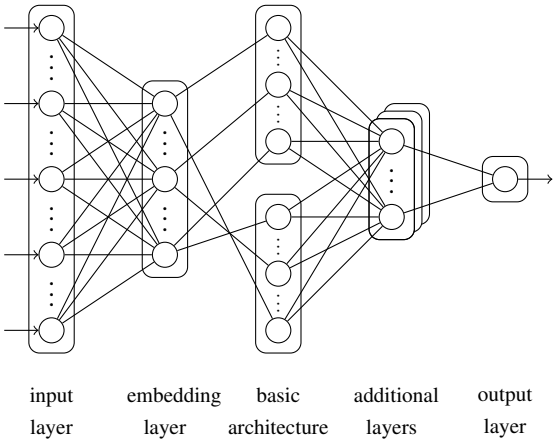

Fig. 2. A schematic illustration of the general network architecture.

Fig. 3. The validation and training loss for RNN_4 (BiLSTM(128), D(64)).

(89.3%). However, on the according test set its accuracy drops to 37.4%. Solely the classifiers based on Decision Trees and Logistic Regression achieve acceptable accuracies (71.9%).

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

2) Neural Network Approaches: For the neural networks we use word embeddings as input, either self-trained or fastText embeddings (see Subsection IV-A). The general network structure as depicted in Figure 2 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 IV). 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 of whether to use lemmatized or just tokenized input and stop words; in all cases the lemmatized dataset including stop words produced better results again.

For all other hyper-parameters we tested all possible combinations (as depicted in Table IV). However, in Table VI we only present the best configurations (regarding the validation results). 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 3 shows the effect for RNN_4; the validation loss optimum is reached after epoch five.

The configurations in Table VI can be read as follows. For the fifth recurrent neural network (RNN_5) we use a bidirectional LSTM (BiLSTM) 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 (with 50 units).

For the random split most neural networks achieve sufficient results. Only RNN_1 and RNN_3 fall behind considerably; *unidirectional* LSTMs and GRUs seem insufficient for the task. However, their *bidirectional* counterparts obtain excellent results; RNN_4 is even the best regarding the validation accuracy using fastText. CNN_1 shows the overall best performance including a test accuracy of 96.6% using fastText. Overall, the majority of neural network configurations work best with fastText embeddings (except for the ANNs). However, only the top three outperform the baseline (Logistic Regression) on the test set: CNN_1, RNN_2, and RNN_4. The results for the random split suggest, that neural network approaches are the most suitable for the task, but must be configured with care.

Either way, the consideration of the sceario-based split is more informative, since it models the realistic deployment of a classifier[3]. 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[4]. 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 fastText embeddings for the random split speaks for this assumption, too. The neural network which showed the best accuracy on the random split (CNN_1) also deteriorates sharply. With either embeddings it only obtains an accuracy of 86.2% for the scenario-based split (despite outstanding accuracies on the validation sets). The bidirectional RNN approaches show the best performances; RNN_2 (BiGRU(32)) reaches accuracy levels beyond 93%. It also performs best on fastText embeddings, which is to be expected, since the test set comprises previously unseen vocabulary. Contrary to expectations, the other bidirectional RNNs show their best

[3]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.

[4]As already mentioned, the validation set is randomly drawn from the training set, which consist of the remaing three scenarios (see Section III).

TABLE VI
FIRST-LEVEL CLASSIFICATION ACCURACY ACHIEVED BY THE NEURAL NETWORKS ON VALIDATION (IN BRACKETS) AND TEST SET. THE BEST RESULTS ARE PRINTED IN BOLD TYPE.

Name	Configuration	Random		Scenario	
		self-trained	fastText	self-trained	fastText
ANN_1	Flat, D(100)	(.916) .914	(.846) .867	(.905) .781	(.874) .715
ANN_2	GMax, D(100)	(.899) .896	(.879) .896	(.893) .668	(.918) .674
CNN_1	Conv(128, 5), Max(2), Conv(128, 5), GMax, D(10)	(**.952**) **.964**	(.954) **.966**	(**.973**) .862	(**.977**) .862
RNN_1	GRU(128), D(100)	(.562) .625	(.562) .625	(.519) .702	(.519) .702
RNN_2	BiGRU(32), DO(0.2), D(64), DO(0.2)	(.947) .944	(.952) .959	(.954) .911	(.958) **.932**
RNN_3	LSTM(128), D(100)	(.562) .625	(.562) .625	(.519) .702	(.519) .702
RNN_4	BiLSTM(128), D(64)	(.951) .955	(**.956**) .959	(.960) **.927**	(.962) .919
RNN_5	BiLSTM(128), D(100), DO(0.3), D(50)	(.936) .937	(.945) .941	(.937) .922	(.954) .917
Baseline (Log. Reg.)	–		(*.927*) .947		(*.891*) .719

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 (RNN_4) showed the third-best validation accuracy; the neural network with the best validation accuracy (CNN_1) is to be found at rank four on the test set.

D. 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. However, the tested hyper-parameters differ slightly (see Table IV) due to the changed boundary conditions of this task (see Section II). For this task a batch size of 32 proved to be best performing. Also, the results are best for the tokenized (unlemmatized) dataset. However, we still do not exclude stop words. The unbalanced dataset poses a challenge; the class *Specification* clearly dominates (see Table II). In return, the *Zero-Rule* classifier becomes a strong baseline.

In Table VII we report the results of the best performing neural networks. We again distinguish results for the random and scenario-based dataset split as well as using self-trained versus fastText embeddings.

Overall, the results are promising. All approaches outperform the baseline clearly. However, no CNN is among the best eight and ANN_1 performs considerably worse (more than 10%) than the remaining; the RNN approaches dominate this task. More particularly, the bidirectional RNNs obtain surprisingly good results. The classification accuracy of RNN_6 for the random split using fastText is 98.8%. The accuracies of all other RNNs are .3% less only. Encouragingly, the results for the scenario split are almost on the same level. Three RNNs exceed 97% using fastText; RNN_3 (BiLSTM(128)) performs best with 97.6%. However, there are only small differences between the configurations. Thus, bidirectional RNNs seem to be suitable for this task in general.

E. 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_4 for the first task and RNN_3 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 alter the separating value to 0.1. This improves the binary classification by 0.8% for the scenario-based split. In a second adjustment 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 [0.01,0.1) and at least two *Declaration*-labels[5]. With both adjustments the accuracy of the binary classification improves by 1.8%.

The second adaptation uses linguistic information to generate continuous semantic parts. For our heuristic we use the semantic role labeler SENNA [5]. 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

[5]The presence of *Declaration*-labels suggests that the description is a teaching effort (first-level classification label *Teaching*).

TABLE VII
SECOND-LEVEL CLASSIFICATION ACCURACY ACHIEVED BY THE NEURAL NETWORKS ON VALIDATION (IN BRACKETS) AND TEST SET. THE BEST RESULTS ARE PRINTED IN BOLD TYPE.

Name	Configuration	Random		Scenario	
		self-trained	fastText	self-trained	fastText
ANN_1	D(100)	(.853) .856	(.853) .848	(.851) .822	(.851) .827
RNN_1	LSTM(128)	(.974) .976	(.978) .977	(.973) .960	(.973) .964
RNN_2	LSTM(128), D(64)	(.973) .972	(.977) .976	(.970) .955	(.971) .963
RNN_3	BiLSTM(128)	(**.986**) .983	(.987) .985	(**.983**) .960	(.981) **.976**
RNN_4	BiGRU(128)	(.984) **.984**	(.985) .985	(.976) .955	(**.982**) .968
RNN_5	BiLSTM(128), D(100), DO(0.3), D(50)	(.982) .982	(.982) .985	(.978) .955	(.981) .968
RNN_6	BiLSTM(128), DO(0.2)	(.985) **.984**	(**.988**) **.988**	(.982) .958	(.981) .975
RNN_7	BiLSTM(256), DO(0.2)	(**.986**) **.984**	(.987) .985	(.982) **.964**	(**.982**) .975
Baseline (MFL)	–		*.759*		*.757*

this label to all words[6]. Since a re-evaluation of the sequence labels is time-consuming, we tested the heuristic on a small random set so far. First results are promising, but we have to run a full-blown evaluation before we can talk about numbers.

V. 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 [6]. 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 [7], [8]. 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 [9]. 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 [10]. 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 [11]. 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 [12]. Landhäußer et al. additionally reconstruct time lines [13]. However, their tool NLCI provides marginal user feedback only. Le et al. enable users to create short scripts for smartphones with SmartSynth [14]. 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 on programming with natural language 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 functionality is not considered so far. Guu et al. use reinforcement learning in combination with the maximal marginal likelihood method to map natural language to code [15]. Rabinovich et al. use an AST-like structured BiLSTM to infer ASTs from textual descriptions [16] and Chen et al. use recurrent neural networks to learn so-called action embeddings [17]. 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 [18].

Teaching new functionality to intelligent systems is of peculiar 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 [19]. 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 [20]. For the transformation the approach uses semantic parsing. Even though the approach does not expect technical terms, the vocabulary and wordings

[6]The rationale behind this decision: if in doubt, it is a specifying part, since they occur most frequently.

are restricted. Markievicz et al. use descriptions that were originally created to teach humans [21]. 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.

VI. Conclusion & Future Work

We presented a hierarchical classification task to grasp the semantic structure of natural language teaching sequences. We define a hierarchical classification task. On the first level we determine whether an utterance contains the (explicitly stated) intent to teach new functionality. On the second we break down these teaching sequences into semantic parts: a declarative part that contains the teaching intent and a name for the new skill, a specifiying part that states the intermediate steps, and miscellaneous, useless information.

We implemented classical machine learning approaches and neural networks to solve the task. For training, validation, and testing we used a dataset from a preliminary study. The neural networks outperform the classical approaches in almost all cases. Even though we tested different types of networks and experimented with different hyper-parameter combinations, bidirectional RNNs proved to be the most suitable for both classification levels. The bidirectional RNNs remain on high accuracy levels even when they are exposed to input that is conceptually different to the training instances. In these tests the best RNN for the first task (BiGRU) shows an accuracy of 93.2%; for the second task it is even 97.6% (BiLSTM).

To further improve the classification we implemented two heuristics. The first uses overruling by the second classifier to increase the first-level accuracy by 1.8%. The second creates continuous semantic parts with the help of a semantic role labeler, but is not yet fully evaluated. However, first results are promising.

We plan to evaluate our approach on other datasets in the near future. We could conceivably launch another online study (with a different setting) or use open-access corpora. To gain an even deeper understanding of the semantic structure of teaching sequences, we might refine the label set of our second-level task, e.g. define a label for skill naming. The next logical step would be to use the gained knowledge on teaching sequences to synthesize actual methods to extend intelligent systems. We plan to create method definitions, including their signatures and body, from natural language utterances.

References

[1] S. Cohen, L. Rokach, and O. Maimon, "Decision-tree Instance-space Decomposition with Grouped Gain-ratio," *Inf. Sci.*, vol. 177, no. 17, pp. 3592–3612, Sep. 2007.

[2] S. Weigelt, V. Steurer, and W. F. Tichy, "At Your Command! An Empirical Study on How Laypersons Teach Robots New Functions," *Submitted to 2020 IEEE 14th International Conference on Semantic Computing (ICSC) Resource Track*.

[3] 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*. Rochester, New York: Association for Computational Linguistics, Apr. 2007, pp. 196–203.

[4] 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*. Valencia, Spain: Association for Computational Linguistics, Apr. 2017, pp. 427–431.

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

[6] 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*, ser. IUI '00. New Orleans, Louisiana, USA: ACM, 2000, pp. 207–211.

[7] A. Begel, "Spoken Language Support for Software Development," in *2004 IEEE Symposium on Visual Languages and Human Centric Computing*, Sep. 2004, pp. 271–272.

[8] A. Begel and S. Graham, "Spoken programs," in *2005 IEEE Symposium on Visual Languages and Human-Centric Computing*, Sep. 2005, pp. 99–106.

[9] 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*, ser. CHI EA '06. New York, NY, USA: ACM, 2006, pp. 239–242.

[10] 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, Jul. 2017, pp. 929–938.

[11] 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.

[12] 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*, ser. CICLing'06. Berlin, Heidelberg: Springer-Verlag, 2006, pp. 319–330.

[13] M. Landhäußer, S. Weigelt, and W. F. Tichy, "NLCI: A Natural Language Command Interpreter," *Automated Software Engineering*, vol. 24, no. 4, pp. 839–861, Dec. 2017.

[14] 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*. Taipei, Taiwan: ACM Press, 2013, p. 193.

[15] 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)*. Vancouver, Canada: Association for Computational Linguistics, Jul. 2017, pp. 1051–1062.

[16] 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.

[17] 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)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 766–777.

[18] 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)*. Melbourne, Australia: Association for Computational Linguistics, Jul. 2018, pp. 731–742.

[19] N. K. Lincoln and S. M. Veres, "Natural Language Programming of Complex Robotic BDI Agents," *Journal of Intelligent & Robotic Systems*, vol. 71, no. 2, pp. 211–230, Sep. 2012.

[20] 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*. Edinburgh, UK: IEEE, Aug. 2014, pp. 868–873.

[21] 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.