applied sciences

MDPI

Article

Intent Detection Problem Solving via Automatic DNN Hyperparameter Optimization

Jurgita Kapočiūtė-Dzikienė [1,2,*], Kaspars Balodis [3,4] and Raivis Skadiņš [3,4]

1 JSC Tilde Information Technology, Naugarduko Str. 100, LT-03160 Vilnius, Lithuania
2 Department of Applied Informatics, Vytautas Magnus University, Vileikos Str. 8, LT-44404 Kaunas, Lithuania
3 Tilde SIA, Vienības Str. 75A, LV-1004 Riga, Latvia; kaspars.balodis@tilde.com (K.B.); Raivis.Skadins@Tilde.lv (R.S.)
4 Faculty of Computing, University of Latvia, Raiņa Blvd. 19, LV-1586 Riga, Latvia
* Correspondence: jurgita.kapociute-dzikiene@vdu.lt

Received: 25 September 2020; Accepted: 19 October 2020; Published: 22 October 2020

Abstract: Accurate intent detection-based chatbots are usually trained on larger datasets that are not available for some languages. Seeking the most accurate models, three English benchmark datasets that were human-translated into four morphologically complex languages (i.e., Estonian, Latvian, Lithuanian, Russian) were used. Two types of word embeddings (fastText and BERT), three types of deep neural network (DNN) classifiers (convolutional neural network (CNN); long short-term memory method (LSTM), and bidirectional LSTM (BiLSTM)), different DNN architectures (shallower and deeper), and various DNN hyperparameter values were investigated. DNN architecture and hyperparameter values were optimized automatically using the Bayesian method and random search. On three datasets of 2/5/8 intents for English, Estonian, Latvian, Lithuanian, and Russian languages, accuracies of 0.991/0.890/0.712, 0.972/0.890/0.644, 1.000/0.890/0.644, 0.981/0.872/0.712, and 0.972/0.881/0.661 were achieved, respectively. The BERT multilingual vectorization with the CNN classifier was proven to be a good choice for all datasets for all languages. Moreover, in the majority of models, the same set of optimal hyperparameter values was determined. The results obtained in this research were also compared with the previously reported values (where hyperparameter values of DNN models were selected by an expert). This comparison revealed that automatically optimized models are competitive or even more accurate when created with larger training datasets.

Keywords: intent detection; fastText and BERT embeddings; CNN; LSTM; BiLSTM classifiers; hyperparameter optimization; English; Estonian; Latvian; Lithuanian; Russian

1. Introduction

Our society is not imaginable without virtual assistants and chatbots such as Siri, Alexa, and Cortana. The AI technology in chatbots is responsible for intelligent human–computer interaction [1]; chatbots can answer vital questions 24/7 [2,3] and even assist in learning [4].

Usually, chatbots are composed of the following components: natural language understanding (NLU; responsible for comprehension of user's questions), dialog management (responsible for a fluent conversation), content (responsible for chatbot's properly selected answers), and custom data (that helps to personalize conversations). The focus of this research is on the NLU module (specifically, on the intent detection) because comprehension of the structure and meaning of user questions is the core of smooth operation in any dialog system.

The intent detection task that is a typical example of text classification can be solved with the rule-based or machine learning (ML) approaches. However, the creation of rules usually requires a lot

of human labor; rules bind to the domain and scope of specific dialog systems [5–7] and, therefore, are hardly transferable to new systems.

On the contrary to rule-based approaches, supervised ML (SML) techniques (especially the deep neural network (DNN)-based techniques) can be chosen instead. In the case of SML, a model (equivalent to a set of rules) is learned automatically from question–answer pairs as training instances [8,9]. Afterward, the learned model can be used to predict intents from the unseen user's questions.

The intent detection task can be complicated due to a number of reasons. Language-independent problems arise due to small training datasets (where intents are covered by only a few questions) and too-short questions (carrying too little information to describe some intent clearly), whereas language-dependent reasons (as free word-order in a sentence, inflection of words, grammatical errors, jargon, abbreviations, or missing diacritics) are more complicated and more difficult to tackle.

In recent years, traditional SML approaches have almost been completely replaced with deep learning (DL). Moreover, neural vectorization has helped to overcome many language-dependent problems as well. Tremendous progress in this area has opened opportunities to apply DNNs for various tasks: image recognition, price prediction, and processing of natural language. In the field of NLP, DNNs are used for language generation, machine translation, text classification, named entity recognition, and many other tasks. Moreover, the development of dialogue systems is not an exception: DNN-based ML methods are used for end-user dialog systems [10–13] and their separate components [14–17].

DNN methods represent an enormous group of methods, having in mind their different types, architectures, and hyperparameter values. Therefore, a manual search for optimal DNN-based models can be very time consuming; moreover, some parameter values simply cannot be investigated as they are considered to be less effective in the English language. Due to these reasons, we assume that automatic DNN parameter optimization is the right solution if seeking the most accurate models. Hence, in this research, parameters of three DNN classifiers (convolutional neural network (CNN); long short-term memory method (LSTM), and bidirectional LSTM (BiLSTM)), with two types of word embeddings (fastText and BERT) on three different datasets (containing 2, 5, and 8 intents) for five different languages (English (EN), Estonian (ET), Latvian (LT), Lithuanian (LV), and Russian (RU)) were tuned automatically. The obtained results are important from the scientific point of view because (1) a huge number of different DNN modifications were tested to determine the most accurate model; (2) the investigation was done on English and four more morphologically complex languages (from the Finno-Ugric, Baltic, and Slavic branches); (3) comparative analysis was possible due to similar experimental conditions and was performed across different datasets and languages.

2. Related Work

ELIZA is the first keyword-based chatbot invented in 1966 [18]. Instead of answering questions, ELIZA acted more like a psychotherapist by questioning users based on their responses. Since then, many different types of chatbots have been created: covering closed, general, or open knowledge domains; with voice or text communication channels; providing interpersonal or intrapersonal services; selecting and prompting or generating answers; created with rule-based, retrieval-based, or ML-based methods.

One big group of chatbots generate answers instead of prompting them to the user. These chatbots typically function in the machine translation manner, but instead of translating source language texts into the target language, they "translate" questions to related answers [19–21]. Some of these chatbots can consider the whole conversation (i.e., previous meaningful utterances) [22] and even mimic the communication style of a user when generating responses [23]. Despite all the positive things, generative chatbots are less accurate and require more training data to achieve sufficient accuracy.

On the contrary, intent-detection-based (i.e., classification-based) chatbots do not require large training datasets and are more accurate compared to generative ones. Intent detectors, like all classifiers, can be trained using labeled instances in a supervised manner [24] and later be used to predict intents

from unseen but domain-related questions (more details about various intent detection techniques can be found in [25]). Supervised classifiers are trained with traditional SML or DL approaches. Traditional ML methods are usually applied to discrete feature representations as textual (e.g., word and character n-grams) or syntactic (e.g., part-of-speech tags) features (e.g., research with support vector machines in [26]).

Nevertheless, major progress in addressing intent detection problems has been made due to DL. Effective DL-based research presented knowledge distillation and a posterior regularization method to detect a user's intent of leaving a service for another service provider (known as a churn detection problem) on an English microblog dataset [27]: the applied CNN method learns simultaneously from logic rules and supervised data (represented as random, skip-gram, CBOW, and gloVe embeddings). The churn detection problem was also successfully tackled in [28]: the CNN method, with bidirectional GRU and bilingual German and English fastText embeddings, was applied to a conversational English and German Twitter dataset. Another research direction covers topic-based intent detection problems. Comparative topic-based intent detection experiments in the English, Estonian, Latvian, Lithuanian and Russian languages, performed with two methods (i.e., the feed forward neural network and fastText embeddings with CNN), demonstrated the superiority of CNN [29]. Authors used rather small datasets (three English benchmark datasets that were also machine translated into Estonian, Latvian, Lithuanian, and Russian languages) but claimed to achieve state-of-the-art performance.

Previously summarized research works focused on closed-set intent detection problems. However, there have been some attempts to detect even those intents that have no training data, such as, e.g., in [30]. Authors have tackled this problem for the English and Chinese languages by applying a two-fold architecture based on BiLSTM, with multiple self-attention heads to discriminate existing intents. However, if this cannot determine any intent, emerging intents are detected from the existing ones (by specifying or generalizing them) using the knowledge transfer method based on a similarity evaluation. Despite the fact that the majority of intent detection research relies on the assumption that any intent can be predicted solely from a user's question, some researchers have offered additional measures to help clarify the meaning of some questions in further conversation. Such a problem (which is called a multiturn response problem) is tackled in [31]. Authors use the deep attention matching network, with stacked attention on text segments with different granularities, and then extract-matched sentence pairs from the conversational context and the author's responses. The authors successfully applied their offered method on the English corpus containing conversations about system troubleshooting and the Chinese social networking corpus.

In this research, a topic-based intent detection problem is tackled for the English, Estonian, Latvian, Lithuanian and Russian languages. This work is a continuation of the research presented in [29,32]. In [32], similar DNN hyperparameter tuning was performed; however, it was done on one Lithuanian dataset only. In contrast, in this research, three different datasets for five different languages are used. Compared to [29], the purpose of this research is to test more types of DNNs, more architectures, more options of DNN hyperparameter values, and more word-embedding types. Contrary to [29], the parameters in our research are tuned automatically by using two hyperparameter optimization strategies. In comparison to [32], a goal of this work is to determine (1) which choices of methods (embedding types, classifier types, DNN architectures, and hyperparameter values) boost the most accuracy for different datasets and languages, (2) if those choices are valid on a dataset-level and/or a language-level; (3) if there are choices that are equally good for all languages. Compared to [29], our goal is to determine (1) if the intent detection benefits from automatic hyperparameter optimization and (2) if the achieved accuracies exceed previously reported ones.

3. Datasets

Tilde's (www.tilde.com) research interests cover morphologically complex languages, e.g., Estonian, Latvian, Lithuanian, and Russian. Unfortunately, labeled datasets for the intent detection problems are not publicly available or may not even exist for some of these languages. The problem

was overcome by taking English benchmark datasets and manually translating them into target languages. Similar datasets (having the same number of instances and intents; the same distribution of instances among training/testing subsets) contribute to the equalization of experimental conditions that, in turn, make comparative analysis possible for different languages. A detailed description of English benchmark datasets can be found in [33]: (1) the **chatbot** dataset (presented in Table 1) contains real users' questions about public transport connections; (2) **askUbuntu** (Table 2) and (3) **webapps** (Table 3) datasets are based on questions from the StackExchange (https://www.stackexchange.com) platform. It is important to notice that training/testing splits for all these benchmark datasets were kept the same as in [33], and it is the main reason why the cross-validation has not been performed. Moreover, having several folds of the same dataset, it would be much more difficult to come up with the summarized recommendations.

Table 1. Statistics about the **chatbot** dataset (numbers in brackets present numbers of distinct words).

Intent	Numb. of Instances	Numb. of Words				
		EN	ET	LT	LV	RU
Training dataset						
FindConnection	57	510 (115)	393 (186)	460 (137)	449 (145)	482 (123)
DepartureTime	43	328 (62)	223 (74)	255 (59)	245 (64)	250 (64)
Testing dataset						
FindConnection	71	508 (99)	369 (154)	484 (113)	449 (120)	483 (104)
DepartureTime	35	241 (48)	168 (53)	201 (49)	193 (55)	192 (48)

Table 2. Statistics about the **askUbuntu** dataset.

Intent	Numb. of Instances	Numb. of Words				
		EN	ET	LT	LV	RU
Training dataset						
Make Update	10	77 (42)	59 (39)	73 (40)	69 (42)	74 (40)
None	3	17 (16)	13 (12)	13 (12)	13 (12)	15 (14)
Setup Printer	10	109 (67)	77 (57)	85 (65)	83 (60)	93 (62)
Shutdown Computer	13	96 (61)	67 (48)	78 (61)	74 (53)	87 (65)
Software Recommendation	17	113 (77)	90 (66)	104 (85)	98 (78)	108 (80)
Testing dataset						
Make Update	37	305 (116)	243 (139)	288 (117)	283 (120)	293 (116)
None	5	46 (39)	34 (28)	44 (37)	38 (33)	43 (39)
Setup Printer	13	99 (54)	77 (55)	78 (52)	81 (51)	85 (50)
Shutdown Computer	14	103 (64)	75 (63)	73 (61)	81 (67)	93 (77)
Software Recommendation	40	322 (197)	259 (174)	296 (206)	278 (201)	303 (204)
Make Update	37	305 (116)	243 (139)	288 (117)	283 (120)	293 (116)

Table 3. Statistics about the **webapps** dataset.

Intent	Numb. of Instances	Numb. of Words				
		EN	ET	LT	LV	RU
Training dataset						
Change Password	2	19 (15)	16 (15)	15 (13)	15 (12)	18 (15)
Delete Account	7	50 (20)	36 (19)	40 (19)	38 (17)	42 (21)
Download Video	1	7 (7)	5 (5)	5 (5)	4 (4)	5 (5)
Export Data	2	16 (13)	13 (12)	17 (15)	13 (11)	19 (16)
Filter Spam	6	43 (36)	38 (37)	43 (39)	42 (38)	42 (37)
Find Alternative	7	40 (33)	35 (30)	34 (29)	35 (30)	38 (32)
None	2	12 (12)	11 (11)	11 (11)	17 (17)	14 (14)
Sync Accounts	3	29 (22)	23 (21)	26 (24)	26 (21)	26 (22)
Testing dataset						
Change Password	6	50 (37)	42 (34)	42 (31)	42 (33)	46 (37)
Delete Account	10	75 (36)	56 (36)	65 (35)	63 (36)	70 (36)
Export Data	3	35 (29)	23 (23)	32 (28)	28 (26)	30 (26)
Filter Spam	14	141 (83)	98 (76)	123 (86)	129 (92)	134 (88)
Find Alternative	16	104 (67)	99 (69)	94 (71)	87 (63)	89 (67)
None	4	35 (33)	26 (26)	32 (32)	31 (31)	34 (32)
Sync Accounts	6	61 (45)	49 (40)	53 (46)	50 (39)	58 (45)

Some language differences can already be seen directly from the tables. For example, the English language usually has the largest average number of words per instance. English is then followed by Russian, Lithuanian, and Latvian, whereas Estonian has the smallest (e.g., 510 and 393 of words covering FindConnection in the training dataset for EN and ET, respectively). If analyzing the distinct words, Estonian comes first as having the largest number, and English the last. All these observations can be explained linguistically: the English language has the least complex morphology, and different morphological forms are expressed with the help of functional words. Estonian and the other three languages are morphologically complex. Estonian is an agglutinative language (having prefixes, suffices, and infixes to express inflections); Lithuanian, Latvian and Russian are fusional languages (allowing the word ending to have several categories depending on the inflection form); all this contributes to a larger number of different words and their forms.

4. Methodology

4.1. Formal Description of the Task

The intent detection problem is a typical example of a supervised text classification task. Formally, such a task is determined as follows:

Let $D = \{d_1, d_2, \ldots, d_n\}$ be a set of documents (questions/statements—an input from a user). Let $C = \{c_1, c_2, \ldots, c_m\}$ be a set of intents (classes). In this research, a closed-set and single-label classification problem is tackled because each $d_i \in D$ can be labeled with only one $c_j \in C$. Let function η be a classification function that maps documents from the determined domain into their correct classes: $D \rightarrow C$. Let $D^L \subset D$ be a set of documents for which intents are known. Thus, (d_i, c_j) pairs are labeled instances used to train a model. Let Γ be a classification method (i.e., classifier, its

architecture, parameter set) that, from labeled instances, can learn an accurate model (which is the approximation of η).

The aim of our solving intent detection task is to offer a classification method Γ that can find the best approximation of η, achieving as high an intent detection accuracy as possible on unseen instances, i.e., on instances D–D^T that were not used for training.

4.2. DNN-Based Classifiers

Training datasets (presented in Section 3) contain labeled instances and, therefore, can be used to train classifiers in a supervised manner. A binary classification problem will be tackled with the **chatbot** dataset (containing only two classes) and multiclass classification with the **askUbuntu** and **webapps** datasets (containing more than two classes).

DL approaches used in various NLP tasks outperform traditional ML and, therefore, allow us to expect the higher accuracy for our intent detection problems as well. From a whole group of SML methods, the following DL approaches (that are considered to be the most suitable for text classification problems) were selected:

- CNN (convolutional neural network; introduced by LeCun [34]) is a DNN used to seek fixed-size patterns, so-called convolutions. The text has a one-dimensional structure in which sequences of tokens matter because convolutions are expressed with a sliding window function over these tokens. By resizing filters and linking their output to different sizes of patterns (consisting of 2, 3, or more adjacent tokens, so-called n-grams), tokens can be detected and generalized. The main advantage of the CNN method is that it learns to detect important patterns regardless of their position in the text. In our experiments, the architecture of the CNN, similar to [35], has been explored.
- The LSTM (long short-term memory) method (presented by Hochreiter and Schmidhuber [36]) is an improved version of the recurrent neural network (RNN). An advantage of RNNs over, e.g., feed forward neural networks is that RNNs have memory and, therefore, can be effectively applied on the sequential data (i.e., text). Sometimes, the presence/absence of some patterns (as in a case of CNN) does not play the major role, but rather, the order of tokens in sequences. However, RNNs confront a vanishing gradient problem and, therefore, cannot solve tasks that require learning long-term dependencies. In contrast, the LSTM method contains a "memory cell" that is able to maintain memory for a longer period of time; integrated gates control what information entering the "memory cell" is important, to which hidden state it has to be outputted, and when it has to be forgotten. Hence, LSTM methods are superior to RNNs when applied to longer sequences.
- The BiLSTM (bidirectional LSTM) method (introduced by Graves and Schmidhuber [37]). Like the LSTM classifier, the BiLSTM is suitable for tasks when a learning problem is sequential. If LSTMs run an input forward, preserving information only from the past, BiLSTMs analyze sequences in both directions, i.e., forward and backward; thus, in any hidden state, they preserve information from the-past-to-the-future and from the-future-to-the-past, respectively.

Experiments with CNN, LSTM, and BiLSTM methods were performed using our implementations in the Python programming language with the Keras library (Keras: the Python DL library; available online: https://keras.io/; adjusted to create DL architectures) and the internal TensorFlow engine (an end-to-end open source ML platform; available online: https://www.tensorflow.org/; used for developing ML methods, managing large data flows, and performing mathematical operations).

4.3. Vectorization Types

The input/output of DNNs must be numerical. Calculated output neuron values (linked to separate classes/intents) can indicate how likely each predicted class/intent is. Input neurons linked to incoming text elements must be numerical in order to apply DNN classifiers (described in Section 4.2) on top of them. For this reason, word embeddings (also called word vectors) that project words into

N-dimensional space are used. In our experiments, the highly recommended types of distributional word embeddings that are able to catch semantic similarities between words were chosen:

- FastText embeddings [38] introduced by Facebook's AI Research Lab. In our experiments, separate English, Estonian, Latvian, Lithuanian and Russian fastText embedding models, cc.en.300.vec, cc.et.300.vec, cc.lv.300.vec, cc.lt.300.vec, and cc.ru.300.vec (the fastText embeddings were downloaded from https://fasttext.cc/docs/en/crawl-vectors.html), respectively, have been used. These models are trained on continuous bag-of-words (CBOW) architecture with position-weights, 300 dimensions, and character n-grams of a window size equal to 5 and 10 negatives [39]. Each fastText word embedding is a sum of sliding symbol n-gram embeddings composing that word (e.g., the 5-g embeddings <*chat*, *chatb*, *hatbo*, *atbot*, *tbot*> compose the word <*chatbot*>). Due to this reason, fastText word embeddings can be created even for misspelled words; moreover, obtained vectors are close to their correct equivalents. It is especially beneficial for languages having a missing diacritics problem in non-normative texts. Despite the fact that the Estonian, Latvian, Lithuanian, and Russian languages have the missing diacritics problem in non-normative texts, datasets in this research contain only normative texts.
- BERT (bidirectional encoder representations from transformers) [40] introduced by Google AI. This neural-based technique (with multidirectional language modeling and attention mechanisms) demonstrates state-of-the-art performance on a wide range of NLP tasks, including chatbot technology. BERT embeddings are robust to disambiguation problems as homonyms are represented by different word vectors based on their context. In our experiments, the BERT service (available online: https://github.com/hanxiao/bert-as-service), with the base multilingual cased 12-layer, 768-hidden, 12-heads model for 104 languages (covering English, Estonian, Latvian, Lithuanian, and Russian), has been used.

4.4. Hyperparameter Tuning

The NLU problem is considered an AI-hard problem (meaning that the created software should be as intelligent as a human), and a lot of effort has been put into the optimization of DNN methods (as, e.g., in [41]). DNN methods have many hyperparameters, and each hyperparameter may have several determined choices (e.g., several types of activation functions) and discrete numeric (e.g., number of neurons) or real numeric (e.g., dropout from an interval [0, 1]) values. However, choosing optimal hyperparameter values manually is a difficult task, even for human experts. To overcome this problem, an open-source Python's library, Hyperas (the information about Hyperas is in https://github.com/maxpumperla/hyperas), implemented to optimize hyperparameters in Keras models automatically, has been used. The following options were experimentally investigated:

- Several DNN architectures (shallower and deeper), having different numbers of hidden layers (i.e., series of convolutional layers in CNN, simple or stacked LSTM and BiLSTM versions).
- DNN hyperparameters: numbers of neurons (100, 200, 300, or 400), dropouts (values from an interval [0, 1]), recurrent dropouts (from [0, 1]), activation functions (relu, softmax, tanh, elu or selu), optimizers (Adam, SGD, RMSprop, Adagrad, Adadelta, Adamax or Nadam), batch sizes (32 or 64), and numbers of epochs (20, 30, 40 or 50).

Tuning of the DNN models (i.e., their hyperparameters) was performed automatically. The training of some models was done on the training split (which contains 80% of instances from the shuffled training set), and the validation was done on the rest (20% of instances from the training set). The hyperparameter optimization was done to increase the validation accuracy, and, for this reason, the following two optimization algorithms were used:

- Random.suggest performs a random search over a set of hyperparameter values in 100 iterations (the experimental investigation revealed that 100 iterations are enough to find the optimal hyperparameter value set that gives maximum accuracy on the validation dataset). When seeking

the most accurate combination, it explores the hyperparameter value space by randomly checking different combinations.

- Tpe.suggest (tree-structured Parzen estimator) [42] performs a Bayesian-based iterative search (for the schematic representation of TPE, see Figure 1). The search strategy of TPE contains two phases. During an initial warm-up phase, it randomly explores a hyperparameter value space. These hyperparameter values can be conditional (additional layer in the architecture), sampled from an interval (as, e.g., for a dropout), or chosen from a determined list of values (e.g., activation functions). The chosen hyperparameter value combinations are used to train a model (with the training dataset split), which is evaluated with the validation split to see each chosen hyperparameter value combination impact on the accuracy. The warm-up phase lasts for *n_init* iterations (*n_init* = 20 in our experiments) and builds a function based on the Bayesian rule presented in Equation (1).

$$P(acc|param) = \frac{P(param|acc) \times P(param)}{P(acc)} \quad (1)$$

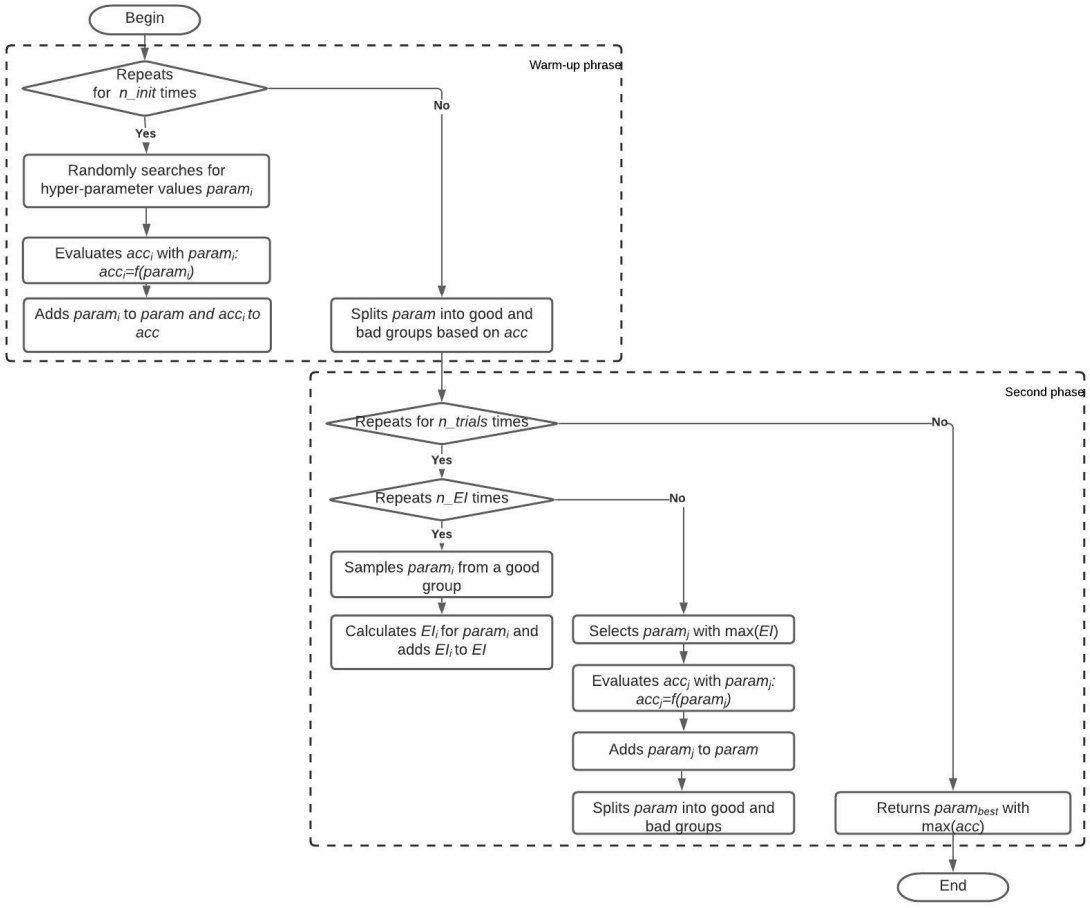

Figure 1. A schematic representation of the TPE (tree-structured Parzen estimator) optimization algorithm.

$P(acc|param)$ is the probability of some validation accuracy (acc) to be achieved with a determined set of hyperparameter values ($param$). Based on this accuracy, hyperparameter value combinations are distributed into good and bad splits. The parameter γ allows us to determine the size of a good split. In our experiments, $\gamma = 0.25$, which means that 25% of all hyperparameter value combinations belong to a good split, and the rest (75%) belong to a bad split. Based on how hyperparameter value

combinations are distributed, the accuracy threshold (denoted as *acc'*) is calculated. Thus, $P(acc|param)$ is expected to give an improvement only if $acc \geq acc'$. $P(param|acc)$ is presented in Equation (2).

$$P(param|acc) = \begin{cases} P^{bad}(param) & if\ acc < acc' \\ P^{good}(param) & if\ acc \geq acc' \end{cases} \quad (2)$$

A goal of the second TPE phase is to maximize the expected improvement (*EI*) ratio in Equation (3).

$$PEI = \frac{P^{good}(param)}{P^{bad}(param)} \quad (3)$$

The maximization of *EI* can be done by choosing the hyperparameter values *param*, with high probability under $P^{good}(param)$ and low probability under $P^{bad}(param)$. It is done by sampling *n_EI* hyperparameter value combinations ($n\_EI = 24$ in our experiments) and choosing the best one with the largest *EI* improvement. Then, the combination with the biggest improvement is memorized and used in the next iteration. In the next iteration, TPE calculates the validation accuracy and distributes the hyperparameter value combinations into good and bad splits, but this time, it uses all previous combinations together with the recent one. The process is repeated until the determined number of trials *n_trials* is reached ($n\_trials = 100$ in our experiments). In our experiments, the default TPE parameters ($n\_init = 20$, $\gamma = 0.25$, $n\_EI = 24$), together with *n_trials* set to 100, have been used. The reason for not experimenting with other values is that these specified parameter values allowed the trained model to reach 100% accuracy on the validation dataset, which resulted in finding the optimal set of hyperparameter values.

5. Experiments and Results

Experiments were carried out with datasets, DNN methods (i.e., CNN, LSTM, and BiLSTM), and vectorization techniques (i.e., fastText and BERT) described in Section 3, Section 4.2, and Section 4.3, respectively. Moreover, DNN Keras models were optimized with the tpe.suggest and random.suggest algorithms presented in Section 4.4. Models were tuned to achieve as high an accuracy on the validation dataset as possible for each language, dataset, and classifier, with word embedding type tuned separately, and later evaluated in the testing phase.

The performance of each model was evaluated with the accuracy metric, as presented in Equation (4).

$$accuracy = \frac{N_{correct}}{N_{all}}, \quad (4)$$

where $N_{correct}$ and N_{all} stands for correctly predicted and all tested instances, respectively.

The model is considered reasonable if its accuracy on the testing dataset is above random Equation (5) (assigning labels to instances according to their probabilities in the testing set) and majority Equation (6) (assigning all instances to a class having the largest probability in the training set) baselines (see Table 4).

$$random\_baseline = \sum P^2\left(c_j\right), \quad (5)$$

where $P(c_j)$ is a probability of a class (intent).

$$majority\_baseline = max\left(P\left(c_j\right)\right), \quad (6)$$

The testing results for English, Estonian, Latvian, Lithuanian, and Russian are summarized in Tables 5–9.

Table 4. Calculated random and majority baselines for different datasets.

Dataset	Random Baseline	Majority Baseline
chatbot	0.558	0.670
askUbuntu	0.283	0.367
webapps	0.186	0.271

Table 5. Evaluated accuracies on the English testing datasets with optimized models. The best accuracy values are in bold; values not statistically significant, differing from the best, are underlined.

	fastText embeddings								
	chatbot			**askUbuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	0.981	0.981	**0.991**	0.725	0.642	0.826	0.373	0.424	0.661
rand.suggest	0.972	0.953	0.981	0.798	0.817	0.872	**0.712**	0.237	**0.712**
	BERT embeddings								
	chatbot			**askUbuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	**0.991**	0.962	0.943	0.853	0.826	0.817	0.508	0.542	0.559
rand.suggest	0.981	0.953	0.915	**0.890**	0.853	0.844	0.678	0.458	0.661

Table 6. Evaluated accuracies on the Estonian testing datasets with optimized models. For the other notation, see Table 5.

	fastText embeddings								
	chatbot			**askUbuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	0.943	0.943	0.953	0.780	0.706	0.633	0.508	0.475	0.542
rand.suggest	0.953	0.943	0.943	0.771	0.670	0.771	0.627	0.356	0.508
	BERT embeddings								
	chatbot			**askUbuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	0.953	0.943	0.943	0.872	0.752	0.761	**0.644**	0.407	0.508
rand.suggest	**0.972**	0.934	0.943	**0.890**	0.706	0.780	0.254	0.525	0.492

Table 7. Evaluated accuracies on the Latvian testing datasets with optimized models. For the other notation, see Table 5.

	fastText embeddings								
	chatbot			**askUbuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	0.962	0.934	0.943	0.761	0.706	0.798	0.559	0.475	0.593
rand.suggest	0.330	0.896	0.953	0.771	0.697	0.826	0.542	0.322	0.559
	BERT embeddings								
	chatbot			askUbuntu			webapps		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	**1.000**	0.972	0.953	0.826	0.780	0.835	**0.644**	0.458	0.525
rand.suggest	0.981	**1.000**	0.962	**0.890**	0.798	0.881	0.610	0.492	0.508

Table 8. Evaluated accuracies on the Lithuanian testing datasets with optimized models. For the other notation, see Table 5.

fastText embeddings									
	chatbot			**askUbuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	0.972	0.962	0.972	0.780	0.752	0.853	0.356	0.508	0.508
rand.suggest	0.915	0.972	**0.981**	0.844	0.339	0.853	0.153	0.390	0.492
BERT embeddings									
	chatbot			**askUbuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	**0.981**	0.972	0.962	0.844	0.651	0.771	0.661	0.542	0.492
rand.suggest	0.972	0.972	0.962	**0.872**	0.706	0.761	**0.712**	0.508	0.525

Table 9. Evaluated accuracies on the Russian testing datasets with optimized models. For the other notation, see Table 5.

fastText embeddings									
	chatbot			**ask Ubuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	0.934	0.943	0.943	0.771	0.743	0.798	0.610	0.542	0.610
rand.suggest	0.830	0.934	0.962	0.780	0.752	0.780	0.271	0.593	0.627
BERT embeddings									
	chatbot			**ask Ubuntu**			**webapps**		
	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM	CNN	LSTM	BiLSTM
tpe.suggest	0.962	0.962	0.934	0.844	0.798	0.826	0.661	0.593	0.644
rand.suggest	**0.972**	0.943	0.953	**0.881**	0.761	0.844	0.576	0.610	0.576

When comparing different evaluation results, it is important to determine if those differences are statistically significant. For this purpose, the McNemar test [43], with 95% confidence ($\alpha = 0.05$), has been used. Differences were considered statistically significant if the calculated p-value was below $\alpha = 0.05$.

6. Discussion

This research assumes that DNN hyperparameter optimization can be done without manual intervention. However, there are a few things that set this process in the right direction: the usage of the most promising word embedding types (i.e., fastText and BERT) and the most suitable classifiers (CNN, LSTM, and BiLSTM), adjusted to deal with the text. Moreover, two hyperparameter optimization algorithms have been applied: random search (rand.suggest) and TPE (tpe.suggest) that combines exploration (reaching new regions of hyperparameter values) and exploitation (searching for optimal solutions in a given region of hyperparameter values) strategies.

Zooming into Tables 5–9 allows us to make the following statements. With some rare exceptions, all obtained results are reasonable because they exceed random and majority baselines (presented in Table 4).

The best results on testing splits for each dataset and each language (English (EN), Estonian (ET), Latvian (LV), Lithuanian (LT), and Russian (RU)) are summarized in Figure 2.

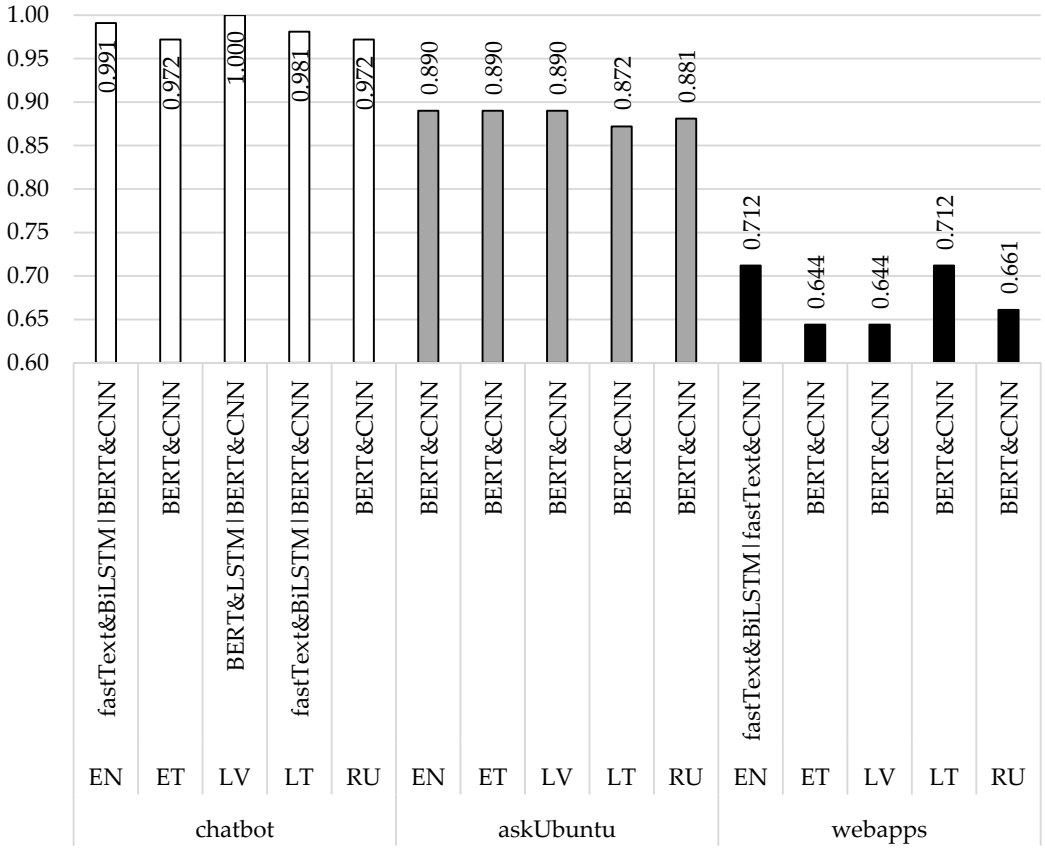

Figure 2. The best results achieved on different datasets with different languages.

As it can be seen from Figure 2, the best and more stable results across different languages are achieved with fewer intents but more training instances (i.e., the **chatbot** dataset), and the worse results are with more intents and less training instances (**webapps**). The validation split (which is 20% from the shuffled training dataset) of the **webapps** dataset is extremely small (i.e., only 6 instances); moreover, these randomly selected instances do not necessarily overlap for different languages. All it means is that for some languages, the validation split happened to be less representative (and less consistent with the testing dataset) than for the others. Despite the fact that the DNN hyperparameter optimization algorithm was able to find very accurate models on the validation splits, these models performed poorly on the testing datasets. From this experimental investigation, it can be concluded that automatic hyperparameter optimization is suitable only for the larger and more representative datasets (as, in our case, for **chatbot** or **askUbuntu**).

When comparing our best results with the results reported in [29], it can be concluded that our DNN hyperparameter optimization method, unfortunately, underperforms [29] on **webapps** for all languages. The reason is that the DNN hyperparameter optimization method is not suitable for smaller datasets. Our approach is competitive (giving the same or very similar accuracy) on the **askUbuntu** dataset for all languages, and it is better on the **chatbot** dataset. Hence, automatic hyperparameter optimization was able to surpass methods that had DNN hyperparameters selected by experts; therefore, the automatic DNN hyperparameter optimization is the right way to seek the most accurate DNN models for intent detection problems.

As can be seen from Tables 5–9, it is hard to make a conclusion on which of the hyperparameter optimization algorithms (rand.suggest or tpe.suggest) is more suitable for our tasks. Thus, both methods are equally good if a large enough number of iterations (100 in our experiments) is selected for the optimization.

In the past, text classification tasks with morphologically complex languages could not reach the same accuracy levels as with English, but the results for English in Figure 2 do not fall out of the picture. Since experimental conditions are equalized for all languages (the same number of intents, the same distribution of instances among different classes, the same classifiers, the same optimization algorithms), the only difference lies in the language processing, i.e., vectorization. However, with neural vectorization, none of these languages have an advantage due to a smaller vocabulary (because the vector space dimensionality is the same for all languages) or less variety in inflection forms (because word embeddings are not discrete but distributional). However, some differences between the choices of word embedding types still exist. As seen from Figure 2, BERT vectorization is a better choice compared to fastText for all morphologically complex languages for all datasets, and this is not surprising. Morphologically complex languages (especially fusional languages) suffer from disambiguation problems, but BERT has mechanisms that are able to vectorize even those words that are written the same but have different meanings, depending on their context, differently. Despite the fact that fastText embeddings are also trained to consider a context around the target word, that context is restricted to only a few words. Despite this, fastText is a suitable vectorization solution for languages (such as English) with strict word order in a sentence. In contrast, BERT is able to consider a much broader context (words, sentences, their order) compared to fastText and is, therefore, more suitable for languages that have a relatively free word order in a sentence (such as Latvian, Lithuanian, and Russian).

Despite LSTM, BiLSTM classifiers can sometimes be very accurate (especially on the **chatbot** dataset, having only two intents and enough representative training data); the domination of the CNN classifier is obvious (see Figure 2). Since we are solving the topic-based intent detection problem (where different intents are related to different topics), the contextual words or their n-grams seem to play a more important role than the sequential nature of the text.

Furthermore, our focus is on the DNN architectures (the most accurate DNN architectures are visualized using the plot_model utility function in Keras) and the hyperparameter values of the most accurate models. The architecture of the most accurate CNN method happens to be the same for all datasets and languages (see Figure 3). Here, a notation *WE* defines a dimensionality of word embeddings (*WE* = 300 with fastText and 768 with BERT), and *C* stands for the number of classes (i.e., 2, 5, and 8 in the **chatbot**, **askUbuntu**, and **webapps** datasets, respectively). The *None* dimension in shape tuples refers to a batch size, which, in our case, is variable (because it is among optimized parameters). Moreover, the *None* value is presented automatically by plot_model and means that the layer can accept input of any size.

Next to the CNN method architecture, architectures of LSTM and BiLSTM methods, which happen to be equally accurate, are presented: i.e., BiLSTM on the **chatbot** and **webapps** datasets with English (see Figure 4); LSTM on the **chatbot** dataset with Latvian (Figure 5); BiLSTM on the **webapps** dataset with Lithuanian (Figure 6).

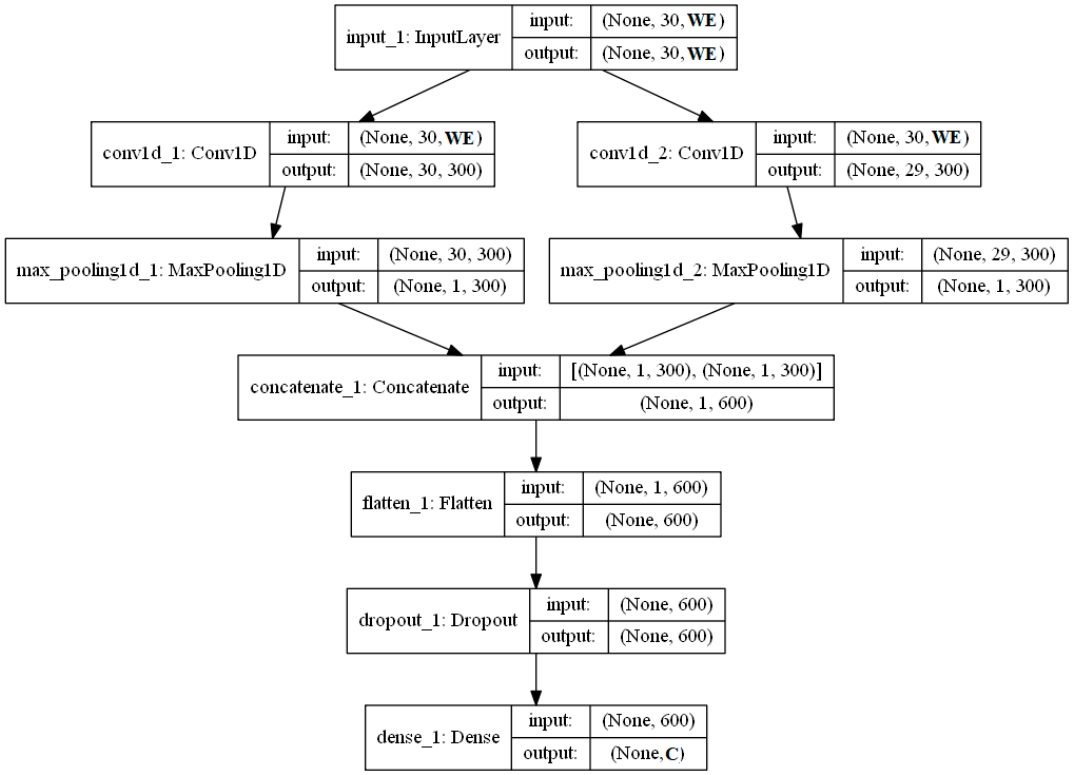

Figure 3. The optimal convolutional neural network (CNN) model architecture.

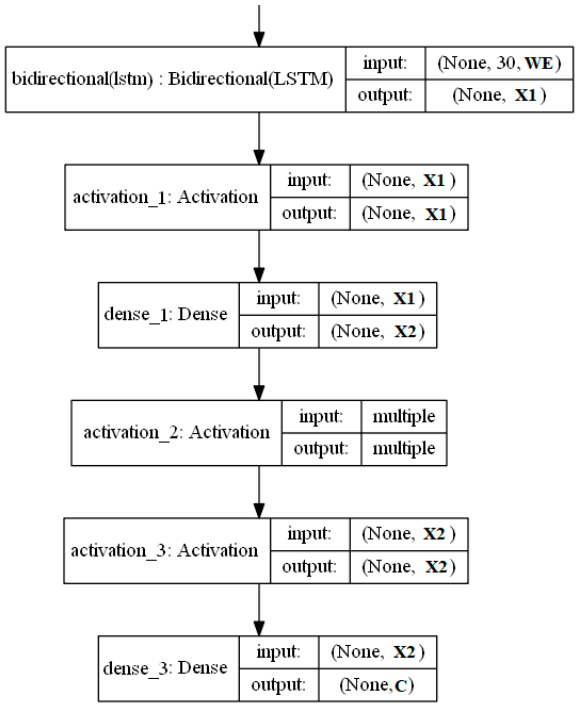

Figure 4. The optimal bidirectional long short-term memory method (BiLSTM) model architecture on **chatbot** and **webapps** datasets for the English language, with X1 = 800 and X2 = 300 on **chatbot** and X1 = 400 and X2 = 200 on **webapps**.

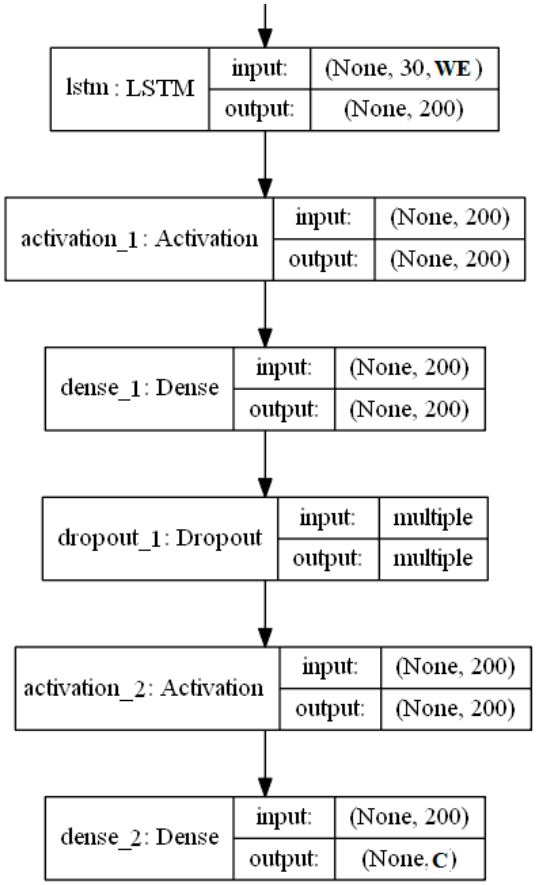

Figure 5. The optimal LSTM model architecture on **chatbot** for the Latvian language.

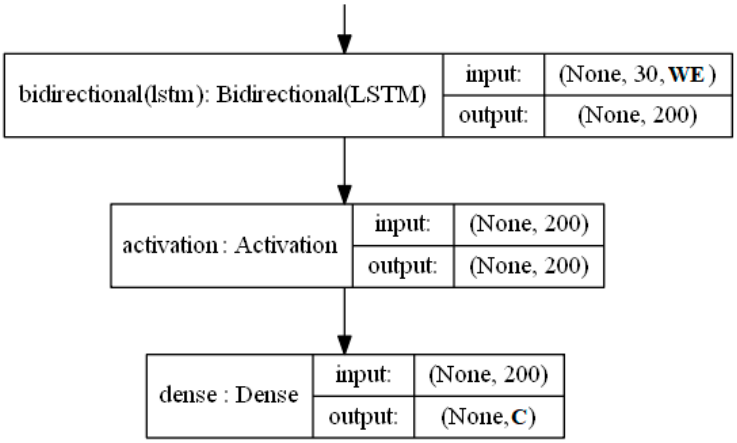

Figure 6. The optimal BiLSTM model architecture on **chatbot** for the Lithuanian language.

Since the plot_model function presents only partial information about hyperparameter values, the missing values are summarized in Appendix A. For many datasets and languages, not only the same CNN classifier and the same CNN architecture (in Figure 3) but also the same set of hyperparameter values allows us to reach the best performance. This set is presented in Appendix A with the English language, the **askUbuntu** dataset, and BERT vectorization. Since this set happened to be optimal in almost half of the best-determined models, it is recommended for various intent detection problems.

7. Conclusions

In this research, we were solving a supervised intent detection problem for the English language and four morphologically complex languages, i.e., Estonian, Latvian, Lithuanian, and Russian. This problem has been tackled by seeking the most accurate models via automatic DNN hyperparameter optimization. In our research, two types of word embeddings (fastText, BERT), three types of DNNs (CNN, LSTM, BiLSTM), their different architectures (shallower and deeper), and various hyperparameter values (e.g., activation functions, numbers of neurons, dropouts) have been explored. The optimization was performed on three English benchmark datasets (containing 2, 5, and 8 intents) that were also manually translated into other languages.

Despite the fact that very strict conclusions cannot be drawn due to a lack of statistical significance in the result differences, some trends are apparent: (1) DNN hyperparameter optimization is the right solution when seeking accurate models for the larger training datasets; (2) the BERT embeddings type is an especially good vectorization choice for morphologically complex languages, whereas English can benefit from fastText as well; (3) the CNN classifier allows us to reach high accuracy levels despite the dataset or language; the other classification techniques are equally good only with enough training data.

The best accuracies were achieved on three testing datasets with 2/5/8 intents and are equal to 0.991/0.890/0.712, 0.972/0.890/0.644, 1.000/0.890/0.644, 0.981/0.872/0.712, and 0.972/0.881/0.661 for the English, Estonian, Latvian, Lithuanian, and Russian languages, respectively. Moreover, compared to the previously reported results in [29], our achieved accuracies are competitive on the **askUbuntu** dataset and better on the **chatbot** dataset.

This research is important from the scientific perspective due to (1) the automatic hyperparameter optimization of DNN models for various intent detection problems and (2) the comparison of obtained results for different languages: i.e., for English and morphologically complex languages from the Finno-Ugric (for Estonian), Baltic (Latvian and Lithuanian), and Slavic (Russian) branches. Moreover, some solutions work across datasets and even languages. It allows us to anticipate that similar results can also be expected for other languages of the same branches.

The research is important due to practical reasons. The optimal parameters are here revealed and, therefore, can be used to train other intent detection-based chatbots for the English, Estonian, Latvian, Lithuanian and Russian languages. However, higher accuracy can be expected only with larger and more representative training datasets.

In future research, it would be useful to experiment with larger datasets, try other classification methods such as BERT fine-tuning, and even go beyond intent detection problems (by focusing on generative chatbots).

Author Contributions: Conceptualization, J.K.-D., K.B., and R.S.; methodology, J.K.-D.; software, J.K.-D. and K.B.; validation, J.K.-D.; formal analysis, J.K.-D. and R.S.; investigation, J.K.-D. and K.B.; resources, K.B.; writing—original draft preparation, J.K.-D.; writing—review and editing, J.K.-D., K.B., and R.S.; visualization, J.K.-D.; project administration, R.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research has been supported by the European Regional Development Fund within the joint project of SIA TILDE and the University of Latvia "Multilingual Artificial Intelligence-Based Human–Computer Interaction" No. 1.1.1.1/18/A/148.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A

The best determined DNN hyperparameter values for the English (EN), Estonian (ET), Latvian (LV), Lithuanian (LT), and Russian (RU) languages:

Language + Dataset	Word Embedding Type + Classifier	Hyperparameter Values
EN + **chatbot**	fastText + BiLSTM (Figure 4)	Activation function (*activation_1*): *elu* Activation function (*activation_2*): *relu* Activation function (*activation_3*): *selu* Activation function in Dense (*dense_3*): *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adamax*
	BERT + CNN (Figure 3)	Activation function in Conv1D: *softmax* Activation function in Dense: *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adamax* Dropout rate: *0.437*
EN + **askUbuntu**	BERT + CNN (Figure 3)	Activation function in Conv1D: *softmax* Activation function in Dense: *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adam* Dropout rate: *0.437*
EN + **webapps**	fastText + CNN (Figure 3)	Activation function in Conv1D: *tanh* Activation function in Dense: *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adagrad* Dropout rate: *0.089*
	fastText + BiLSTM (Figure 4)	Activation function (*activation_1*): *tanh* Activation function (*activation_2*): *relu* Activation function (*activation_3*): *tanh* Activation function in Dense (*dense_3*): *softmax* Batch size: *64* Epochs: *20* Optimizer: *Adagrad*
ET + **chatbot** ET + **askUbuntu** ET + **webapps**	BERT + CNN (Figure 3)	Activation function in Conv1D: *softmax* Activation function in Dense: *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adam* Dropout rate: *0.437*
LV + **chatbot**	BERT + CNN (Figure 3)	Activation function in Conv1D: *softmax* Activation function in Dense: *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adam* Dropout rate: *0.437*
	BERT + LSTM (Figure 5)	Activation function (*activation_1*): *tanh* Activation function (*activation_2*): *relu* Activation function in Dense (*dense_2*): *softmax* Batch size: *64* Epochs: *20* Optimizer: *Adagrad*

Language + Dataset	Word Embedding Type + Classifier	Hyperparameter Values
LV + **askUbuntu**	BERT + CNN (Figure 3)	Activation function in Conv1D: *softmax* Activation function in Dense: *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adam* Dropout rate: *0.437*
LV + **webapps**	BERT + CNN (Figure 3)	Activation function in Conv1D: *relu* Activation function in Dense: *softmax* Batch size: *32* Epochs: *30* Optimizer: *Adam* Dropout rate: *0.276*
LT + **chatbot**	fastText + BiLSTM (Figure 6)	Activation function (*activation*): *elu* Activation function in Dense (*dense*): *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adamax*
	BERT + CNN (Figure 3)	Activation function in Conv1D: *softmax* Activation function in Dense: *softmax* Batch size: *64* Epochs: *40* Optimizer: *Adam* Dropout rate: *0.437*
LT + **askUbuntu**	BERT + CNN (Figure 3)	Activation function in Conv1D: *selu* Activation function in Dense: *elu* Batch size: *64* Epochs: *40* Optimizer: *Adagrad* Dropout rate: *0.734*
LT + **webapps**	BERT + CNN (Figure 3)	Activation function in Conv1D: *softmax* Activation function in Dense: *softmax* Batch size: *64*Epochs: *40* Optimizer: *Adam* Dropout rate: *0.437*
RU + **chatbot** RU + **askUbuntu**	BERT + CNN (Figure 3)	Activation function in Conv1D: *softmax* Activation function in Dense: *softmax* Batch size: *64*Epochs: *40* Optimizer: *Adam* Dropout rate: *0.437*
RU + **webapps**	BERT + CNN (Figure 3)	Activation function in Conv1D: *selu* Activation function in Dense: *softmax* Batch size: *32*Epochs: *30* Optimizer: *Nadam* Dropout rate: *0.148*

References

1. Adamopoulou, E.; Moussiades, L. An Overview of Chatbot Technology. In *Artificial Intelligence Applications and Innovations, IFIP Advances in Information and Communication Technology, Proceedings of the 16th IFIP WG 12.5 International Conference, AIAI 2020, Neos Marmaras, Greece, 5–7 June 2020*; Maglogiannis, I., Iliadis, L., Pimenidis, E., Eds.; Springer: Cham, Switzerland; Volume 584.
2. Battineni, G.; Chintalapudi, N.; Amenta, F. AI Chatbot Design during an Epidemic like the Novel Coronavirus. *Healthcare* **2020**, *8*, 154. [CrossRef] [PubMed]
3. Maniou, T.A.; Veglis, A. Employing a Chatbot for News Dissemination during Crisis: Design, Implementation and Evaluation. *Futur Internet* **2020**, *12*, 109. [CrossRef]
4. Villegas-Ch, W.; Arias-Navarrete, A.; Palacios-Pacheco, X. Proposal of an Architecture for the Integration of a Chatbot with Artificial Intelligence in a Smart Campus for the Improvement of Learning. *Sustainability* **2020**, *12*, 1500. [CrossRef]
5. Fonte, F.A.; Carlos, J.; Rial, B.; Nistal, M.L. TQ-Bot: An AIML-based tutor and evaluator bot. *J. Univ. Comput. Sci.* **2009**, *15*, 1486–1495.
6. Van Der Goot, R.; Van Noord, G. MoNoise: Modeling Noise Using a Modular Normalization System. *Comput. Linguist. Neth. J.* **2017**, *7*, 129–144.
7. Shawar, B.A.; Atwell, E. Machine Learning from dialogue corpora to generate chatbots. *Expert Update J.* **2003**, *6*, 25–29.
8. Xu, P.; Sarikaya, R. Convolutional neural network based triangular CRF for joint intent detection and slot filling. In *2013 IEEE Workshop on Automatic Speech Recognition and Understanding*; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2013; pp. 78–83.
9. Yao, K.; Peng, B.; Zhang, Y.; Yu, D.; Zweig, G.; Shi, Y. Spoken language understanding using long short-term memory neural networks. In *2014 IEEE Spoken Language Technology Workshop (SLT)*; Institute of Electrical and Electronics Engineers (IEEE): New York, NY, USA, 2014; pp. 189–194.
10. Serban, I.V.; Sordoni, A.; Bengio, Y.; Courville, A.C.; Pineau, J. Building End-To-End Dialogue Systems Using Generative Hierarchical Neural Network Models. In Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence AAAI, Phoenix, AZ, USA, 12–17 February 2016; Volume 16, pp. 3776–3784.
11. Shang, L.; Lu, Z.; Li, H. Neural responding machine for short-text conversation. In Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing, Beijing, China, 26–31 July 2015; Volume 1, pp. 1577–1586.
12. Wen, T.H.; Vandyke, D.; Mrkšíc, N.; Gašíc, M.; Rojas-Barahona, L.M.; Su, P.H.; Ultes, S.; Young, S. A network-based end-to-end trainable task-oriented dialogue system. In Proceedings of the 15th Conference of the European Chapter of the Association for Computational Linguistics, EACL 2017-Proceedings of Conference, Valencia, Spain, 3–7 April 2017; Volume 1, pp. 438–449.
13. Yang, X.; Chen, Y.-N.; Hakkani-Tür, D.; Crook, P.; Li, X.; Gao, J.; Deng, L. A Network-based End-to-End Trainable Task-oriented Dialogue System. In Proceedings of the 2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 5–9 March 2017; pp. 5690–5694.
14. Kalchbrenner, N.; Blunsom, P. Recurrent Convolutional Neural Networks for Discourse Compositionality. In Proceedings of the Workshop on Continuous Vector Space Models and their Compositionality, Sofia, Bulgaria, 9 August 2013; pp. 119–126.
15. Liu, C.; Xu, P.; Sarikaya, R. Deep contextual language understanding in spoken dialogue systems. In Proceedings of the Sixteenth Annual Conference of the International Speech Communication Association (INTERSPEECH), Dresden, Germany, 6–10 September 2015.
16. Lowe, R.; Pow, N.; Serban, I.V.; Pineau, J. The Ubuntu Dialogue Corpus: A Large Dataset for Research in Unstructured Multi-Turn Dialogue Systems. In Proceedings of the 16th Annual Meeting of the Special Interest Group on Discourse and Dialogue, Prague, Czech Republic, 2–4 September 2015; pp. 285–294.
17. Wen, T.-H.; Gasic, M.; Mrkšić, N.; Su, P.-H.; Vandyke, D.; Young, S. Semantically Conditioned LSTM-based Natural Language Generation for Spoken Dialogue Systems. In Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing, Lisbon, Portugal, 17–21 September 2015; pp. 1711–1721.
18. Weizenbaum, J. ELIZA–A computer program for the study of natural language communication between man and machine. *Commun. ACM* **1996**, *9*, 36–45. [CrossRef]
19. Vinyals, O.; Le, Q. A Neural Conversational Model. *arXiv* **2015**, arXiv:1506.05869.

20. Kim, J.; Lee, H.-G.; Kim, H.; Lee, Y.; Kim, Y.-G. Two-Step Training and Mixed Encoding-Decoding for Implementing a Generative Chatbot with a Small Dialogue Corpus. In Proceedings of the Workshop on Intelligent Interactive Systems and Language Generation (2IS&NLG), Tilburg, The Netherlands, 5 November 2018; pp. 31–35.
21. Kapočiūtė-Dzikienė, J. A Domain-Specific Generative Chatbot Trained from Little Data. *Appl. Sci.* **2020**, *10*, 2221. [CrossRef]
22. Kim, J.; Oh, S.; Kwon, O.-W.; Kim, H. Multi-Turn Chatbot Based on Query-Context Attentions and Dual Wasserstein Generative Adversarial Networks. *Appl. Sci.* **2019**, *9*, 3908. [CrossRef]
23. Zhang, W.N.; Zhu, Q.; Wang, Y.; Zhao, Y.; Liu, T. Neural Personalized Response Generation as Domain Adaptation. *World Wide Web* **2019**, *22*, 1427–1446. [CrossRef]
24. Sebastiani, F. Machine Learning in Automated Text Categorization. *ACM Comput. Surv.* **2002**, *34*, 1–47. [CrossRef]
25. Liu, J.; Li, Y.; Lin, M. Review of Intent Detection Methods in the Human-Machine Dialogue System. *J. Phys. Conf. Ser.* **2019**, 1267. [CrossRef]
26. Akulick, S.; Mahmoud, E.S. Intent Detection through Text Mining and Analysis. In Proceedings of the Future Technologies Conference (FTC), Vancouver, WA, Canada, 29 November 2017.
27. Gridach, M.; Haddad, H.; Mulki, H. Churn identification in microblogs using convolutional neural networks with structured logical knowledge. In Proceedings of the 3rd Workshop on Noisy User-generated Text, Copenhagen, Denmark, 7 September 2017; pp. 21–30.
28. Abbet, C.; M'hamdi, M.; Giannakopoulos, A.; West, R.; Hossmann, A.; Baeriswyl, M.; Musat, C. Churn Intent Detection in Multilingual Chatbot Conversations and Social Media. In Proceedings of the 22nd Conference on Computational Natural Language Learning, Brussels, Belgium, 31 October–1 November 2018; pp. 161–170.
29. Balodis, K.; Deksne, D. FastText-Based Intent Detection for Inflected Languages. *Information* **2019**, *10*, 161. [CrossRef]
30. Xia, C.; Zhang, C.; Yan, X.; Chang, Y.; Yu, P. Zero-shot User Intent Detection via Capsule Neural Networks. In Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, 31 October–4 November 2018; pp. 3090–3099.
31. Zhou, X.; Li, L.; Dong, D.; Liu, Y.; Chen, Y.; Zhao, W.X.; Yu, D.; Wu, H. Multi-Turn Response Selection for Chatbots with Deep Attention Matching Network. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, Melbourne, Australia, 15–20 July 2018; pp. 1118–1127.
32. Kapočiūtė-Dzikienė, J. Intent Detection-Based Lithuanian Chatbot Created via Automatic DNN Hyper-Parameter Optimization. In *Frontiers in Artificial Intelligence and Applications, Volume 328: Human Language Technologies–The Baltic Perspective*; IOS Press: Amsterdam, The Netherlands, 2020; pp. 95–102.
33. Braun, D.; Hernandez, M.A.; Matthes, F.; Langen, M. Evaluating Natural Language Understanding Services for Conversational Question Answering Systems. In Proceedings of the 18th Annual SIGdial Meeting on Discourse and Dialogue, Saarbrücken, Germany, 15–17 August 2017; pp. 174–185.
34. LeCun, Y.; Bottou, L.; Bengio, Y.; Haffner, P. *Gradient-Based Learning Applied to Document Recognition*; IEEE: Pasadena, CA, USA, 1998; pp. 2278–2324.
35. Kim, Y. Convolutional Neural Networks for Sentence Classification. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1746–1751.
36. Hochreiter, S.; Schmidhuber, J. Long Short-Term Memory. *Neural Comput.* **1997**, *9*, 1735–1780. [CrossRef] [PubMed]
37. Graves, A.; Schmidhuber, J. Framewise phoneme classification with bidirectional LSTM and other neural network architectures. *Neural Netw.* **2005**, *18*, 602–610. [CrossRef] [PubMed]
38. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146. [CrossRef]
39. Grave, E.; Bojanowski, P.; Gupta, P.; Joulin, A.; Mikolov, T. Learning Word Vectors for 157 Languages. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, 7–12 May 2018.
40. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In Proceedings of the NAACL-HLT 2019, Minneapolis, MN, USA, 2–7 June 2019; pp. 4171–4186.

41. Baioletti, M.; Di Bari, G.; Milani, A.; Poggioni, V. Differential Evolution for Neural Networks Optimization. *Mathematics* **2020**, *8*, 69. [CrossRef]
42. Bergstra, J.; Bardenet, R.; Bengio, Y.; Kégl, B. Algorithms for Hyper-Parameter Optimization. In Proceedings of the 24th International Conference on Neural Information Processing Systems, Granada, Spain, December 2011; pp. 2546–2554.
43. McNemar, Q.M. Note on the Sampling Error of the Difference Between Correlated Proportions or Percentages. *Psychometrika* **1947**, *12*, 153–157. [CrossRef] [PubMed]