information

MDPI

Article

FastText-Based Intent Detection for Inflected Languages †

Kaspars Balodis [1,2,*] and Daiga Deksne [1]

[1] Tilde, Vienības Gatve 75A, LV-1004 Rīga, Latvia; daiga.deksne@Tilde.lv
[2] Faculty of Computing, University of Latvia, Raiņa blvd. 19, LV-1586 Rīga, Latvia
* Correspondence: kaspars.balodis@Tilde.lv
† This paper is an extended version of our paper published in 18th International Conference AIMSA 2018, Varna, Bulgaria, 12–14 September 2018.

Received: 15 January 2019; Accepted: 25 April 2019; Published: 1 May 2019

Abstract: Intent detection is one of the main tasks of a dialogue system. In this paper, we present our intent detection system that is based on fastText word embeddings and a neural network classifier. We find an improvement in fastText sentence vectorization, which, in some cases, shows a significant increase in intent detection accuracy. We evaluate the system on languages commonly spoken in Baltic countries—Estonian, Latvian, Lithuanian, English, and Russian. The results show that our intent detection system provides state-of-the-art results on three previously published datasets, outperforming many popular services. In addition to this, for Latvian, we explore how the accuracy of intent detection is affected if we normalize the text in advance.

Keywords: intent detection; word embeddings; dialogue system

1. Introduction and Related Work

Recent developments in deep learning have made neural networks the mainstream approach for a wide variety of tasks, ranging from image recognition to price forecasting to natural language processing. In natural language processing, neural networks are used for speech recognition and generation, machine translation, text classification, named entity recognition, text generation, and many other tasks. In virtual assistants and dialogue systems, neural networks are used for either end-to-end system training [1–5], or, in different parts of the system, for intent detection, generation of the response text, and tracking of the dialogue state [6–9].

Intent detection is formulated as a classification task; since the dialogue system is created to answer a limited range of questions, there is a finite predefined set of intents. The task of intent detection is to map the user input to the most probable intent. Although pattern-based recognition was commonly used before the spread of neural network-based techniques and is a working solution [10–13], this approach is somewhat limited, and requires labor-intensive creation of a large number of patterns by hand. Therefore, it is more efficient to use classifiers that learn from labeled examples [14–16]. The newest dialogue systems use neural network classifiers for intent detection [17–19].

In a dialogue system, the specific nature of intent detection that separates it from other text classification tasks is that there is often only a very small amount of data available to train the model. Moreover, the utterances to be classified are relatively short and can contain grammatical errors, the sentences can be poorly structured, and users may use informal slang or abbreviations. Deep learning methods generally require a large amount of training data, which is not the typical case for chatbots. However, one can still leverage the power of deep learning by using pre-trained word embeddings that are pre-trained in an unsupervised manner on large corpora. Until recently, massive pre-trained language representation models, such as ULMFiT (Universal Language Model

Fine-Tuning) [20], ELMo (Embeddings from Language Models) [21], or BERT (Bidirectional Encoder Representations from Transformers) [22] were only available for the English language, and thus of limited usability for other languages.

Word embeddings [23] represent an effective, unsupervised, pre-trainable way of injecting general knowledge about the language in the classification model, given enough monolingual data to train on. In this research, we used the *fastText* [24] to generate word embeddings because its use of subword units makes it more appropriate for less-resourced and inflected languages. We evaluate the quality of the generated word embeddings in the intent detection setting. We extend our previous research [25] by evaluating the resulting intent detection system on the five languages commonly used in Baltic countries. To the best of our knowledge, this is the first research paper on intent detection of the languages used in Baltic countries. We also explore more sophisticated methods for text normalization and automatic error-correcting to further improve intent detection accuracy. We believe that our research can be adapted to other inflected languages.

The rest of the paper is organized as follows. In Section 2, a general overview of the intent detection problem is given. We present our intent detection models in Section 3, and evaluate them in Section 4. Section 5 deals with text normalization for the Latvian language. In Section 6, we provide some conclusions about the results.

2. Intent Detection

The input for an intent detection system is a user's utterance (a sentence or a question). The task of the system is to determine which of several predefined intents is most likely for the given utterance. The intent detection task can be approached by dividing it into separate subtasks. The utterance is sequentially preprocessed, vectorized, and classified. An example of processing an utterance is shown in Table 1.

Table 1. An example of intent detection.

Step	Result	Example
user input	the original user utterance	Do you wrok in the morning??/
preprocessing	preprocessed question	do you work in the morning
vectorization	vectorized form	[0.315243, −0.35235, ..., 1.0521]
classification	probability distribution over intents	working_hours: 0.79, greeting: 0.12, ...

2.1. Preprocessing

Utterance preprocessing is comprised of several subtasks, as depicted in Table 2. Depending on the system, a subset of these steps can be done, or additional steps can be introduced. The preprocessing step has been heavily researched, and is shown to have a great impact on the performance of the classification [26–28]. We offer a more in-depth analysis of the error correction and text normalization step in Section 5.

Table 2. An example of preprocessing.

Task	Description	Example Utterance
input	the original user utterance	please say, Doez it work in usa?
tokenization	distinct tokens, such as punctuation marks, words, email addresses, links, numbers, abbreviations, etc. are properly separated	please say, Doez it work in usa?
automatic error correction/ text normalization	some of the grammatical errors are programmatically corrected	please say, Does it work in usa?

Table 2. *Cont.*

Task	Description	Example Utterance
truecasing or lowercasing	the text is converted into lowercase or the true case	Please say, does it work in USA?
removal of punctuation	punctuation marks and other symbols are removed	Please say does it work in USA
stemming or lemmatization	words are transformed into a base form	Please say do it work in USA
removal of stopwords	insignificant words are removed, typically from a predefined stoplist	do work USA

2.2. Vectorization

Vectorization is the process that transforms the utterance from text to a vector of real numbers. Typically, this is done using *word embeddings* that map the words to vectors of real numbers so that words with a similar semantical meaning (according to cosine similarity) are positioned close to each other in this vector space. Many algorithms perform better when using such continuous representation instead of, for example, *one-hot* vectors. The word embeddings are pre-trained on a large corpora of texts in the given language. Word embeddings can be trained with several different tools. Some of the most popular are *word2vec* [23], *gloVe* [29], and *fastText* [24].

Typically, word embedding models ignore the morphology of the word and learn a distinct vector for each word. In contrast, *fastText* creates a vector for each character *n*-gram. The word vector is then calculated as the average of its character *n*-gram vectors [24]. Therefore, even an out-of-vocabulary word gets assigned a vector based on its subword units. This is even more important for inflected languages, since some inflected forms of words are rare and may not even appear in the training data. Because of its simplicity and efficient realization, training the *fastText* embeddings is faster than most other alternatives.

One can obtain sentence embeddings simply by taking the average of the vectors of the words appearing in the sentence. There are also more sophisticated ways, such as using deep language representation models [20–22,30].

Different methods to improve the quality of word embeddings have been researched. It has been shown that enriching the word embeddings with semantical information improves intent detection accuracy [31]. In [32], the authors introduced probabilistic *fastText* word representations in order to deal with multiple word senses. Classification quality can also be improved by using domain-specific word embeddings [33,34].

2.3. Classification

In the classification phase, the most probable intent is determined for the vectorized utterance. This is done with a machine learning algorithm, typically a neural network. The classifier outputs either the most probable intent or a probability distribution over intents.

3. Models

In this section, we describe the intent detection models we evaluate.

3.1. Preprocessing

We tokenized and lowercased the sentences, and deleted all non-alphanumeric characters.

3.2. Vectorization

We used the fastText tool for vectorization [24,35]. For all languages, we compared the intent detection accuracy using word embeddings publicly released by Facebook [24,36,37] and word embeddings trained on our internal monolingual corpora.

For the word embeddings released by Facebook, we used the ones trained on Wikipedia (https://fasttext.cc/docs/en/pretrained-vectors.html) because the ones trained on Common Crawl (https://fasttext.cc/docs/en/crawl-vectors.html) showed inferior results in our tests. However, the inferior results might have been a result of different preprocessing (the Common Crawl corpus was not lowercased).

Our word embeddings were trained on large monolingual text corpora mainly from online news articles. The sizes of the corpora used for training the word embeddings are shown in Table 3. The corpus used for word embedding training was preprocessed, as described in the previous subsection. We trained a 300-dimensional *skipgram* model using the default fastText parameters (*lr = 0.05, lrUpdateRate = 100, ws = 5, epoch = 5, neg = 5, loss = ns, wordNgrams = 1, bucket = 2,000,000, minn = 3, maxn = 6, t = 0.0001*). Both types of embeddings were 300-dimensional.

Table 3. Sizes of the corpora used for training the word embeddings.

Language	Words (millions)	Utterances (millions)
English	2445	125
Estonian	420	32
Latvian	1368	82
Lithuanian	819	58
Russian	1493	96

Improvement in the fastText Vectorizer

During our research, we found a method to improve the way in which fastText vectorizes sentences. Normally, the fastText sentence vector is calculated as the length-normalized average of the word vectors. The formula is

$$W = \frac{1}{m} \sum_{i=1}^{m} \frac{1}{\|w_i\|} \cdot w_i,$$

where W is the resulting sentence vector, $w_1, \ldots, w_m$ are word embedding vectors of the words in the sentence, and $\|w_i\|$ is the length (ℓ^2 norm) of the vector w_i. The $\frac{1}{\|w_i\|}$ factor can impart additional noise in the sentence vector. Usually, more important words have longer vectors than less important ones, as demonstrated in Table 4. Hence, this normalization of the vectors by their length assigns a relatively higher weight for the less important words.

We evaluated a modified version of the fastText vectorizer without the normalization by length. Therefore, the sentence vector was calculated simply as $W = \frac{1}{m} \sum_{i=1}^{m} w_i$. We observed that using the unnormalized version of the sentence vector often led to significantly better training and test set accuracy, and the training converged faster in almost all cases. We are not aware of the reasons behind this design choice by the fastText team to include normalization in sentence vector generation.

Table 4. Lengths of the word embeddings vectors.

Word	English Translation	Length $\|w_i\|$ of the Word Vector w_i
un	and	1.78
k = ada	what	2.76
ir	is	2.27
mana	my	3.12
ip	IP	3.96
adrese	address	4.91

3.3. Classifier Types

In this work, two different classifiers are investigated—namely, a shallow feedforward network and a convolutional network.

3.3.1. Feedforward Network

The fastText tool includes a classifier that uses the sentence vectors for the classification [35]. However, we observed that this classifier ignored out-of-vocabulary words when generating the sentence vector. This can lead to lower performance because it is likely that a word misspelled by a user will be ignored, even though creating a vector from the parts of the word would yield a similar vector to what it would be if the user had typed the word correctly.

Therefore, we examine a neural network classifier that has the same architecture as in the fastText classifier (shallow feedforward network with a softmax layer, see Figure 1) with the difference that it also includes out-of-vocabulary words in the sentence vector. The utterance vectorized by fastText (either with or without normalization by length) is used as the input, after which comes a layer of neurons, on top of which the softmax function is applied to get a probability distribution over intents.

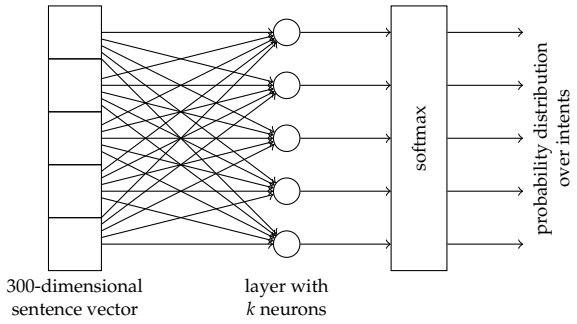

Figure 1. The architecture of the feedforward neural network, where k is the number of possible intents.

3.3.2. Convolutional Network

We examined classification with a convolutional network that had an architecture similar to [38]. The architecture of the network we used is presented in Figure 2. We used 100 filters with width 1, and 100 filters with width 2. We used regularization with a dropout layer and a dropout rate of 0.5. This setting showed the most consistent performance.

We trained the feedforward network and the convolutional network with the Adam optimizer for 200 and 40 epochs, respectively, as a relatively safe point when the training had converged. This could possibly be improved to early stopping on a validation set.

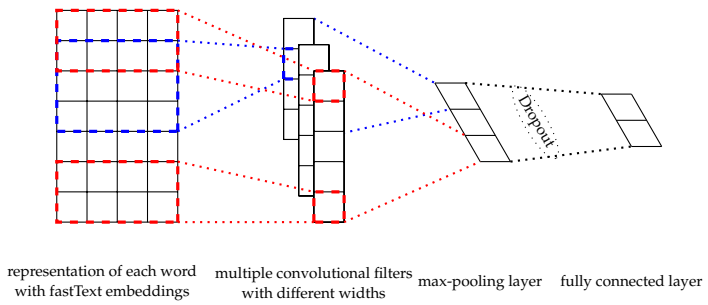

Figure 2. The architecture of the convolutional network.

4. Experimental Setup and Results

In this section, we evaluate different intent detection models on three previously published datasets [39]. We evaluate the intent detection accuracy of the feedforward network classifier

(with normalized and unnormalized versions of the sentence vectors) and the convolution network classifier on the two types of word embeddings.

We also compare our results with the results of several other popular dialogue system providers. In [39], the quality of intent detection and entity recognition was compared for several dialogue system providers—namely, *Google Dialogflow* (formerly called *API.ai*) (https://dialogflow.com/), *Microsoft LUIS* (https://www.luis.ai/), *IBM Watson* (https://www.ibm.com/watson/), and *RASA* (https://rasa.com/). Furthermore, the Snips team added *Snips.ai* (https://snips.ai/) and *RASA* with the *spaCy* backend to the comparison [40]. *Wit.ai* (https://wit.ai) was not tested in [39] because of the unreliability of their import system. However, we managed to test Wit.ai, and added it to the comparison.

In [39], the results for both intent detection and entity recognition were combined in a single score. However, in this paper, we only evaluate the intent detection accuracy and use it as the main metric to compare the different dialogue system providers. We calculate the intent detection accuracy using the counts of true and false positives and negatives listed in [39] and [40].

4.1. Datasets

In [39], three datasets are used. The *chatbot* dataset contains users' questions from a *Telegram* chatbot that answers questions related to public transport connections. The datasets *askubuntu* and *webapps* were created from the most popular questions of the *ask ubuntu* (https://askubuntu.com/) and *Web Applications* (https://webapps.stackexchange.com/) question and answer sites. The number of utterances in the datasets is shown in Table 5. We used the same training and test set split as in [39]. For a detailed description of the datasets, see [39].

Table 5. Number of utterances in the datasets.

Dataset	Training	Test
askubuntu	53	109
chatbot	100	106
webapps	30	59

For the English language, we used the original datasets. For Estonian, Latvian, Lithuanian, and Russian languages, the datasets were machine-translated by the Tilde machine translation system [41] and post-edited by humans to correct any translation errors.

4.2. Results

In this subsection, we test different intent detection models on the languages used in the Baltic countries—that is, Estonian, Latvian, and Lithuanian, as well as English and Russian.

As of publication time, language support for less-resourced languages was weak in the popular chatbot services. The only exception is Wit.ai, which supports all the languages examined in this subsection. Dialogflow supports Russian and English. Other services do not support any of these languages other than English, but Microsoft Language Understanding Intelligent Service (LUIS) does recommend translating the utterances via their machine translation application programming interface (API) before using the intent detection module (https://docs.microsoft.com/en-us/azure/cognitive-services/luis/luis-language-support). We tested Wit.ai on all languages, and also tested LUIS using the Microsoft machine translation service.

As an illustration, we reported the average accuracy of five runs in Table 6; however, in Appendix A we analyze the statistical significance using only the first run of each model. Our trained word embeddings performed better than the Wikipedia ones. Using the unnormalized version of the sentence vectors can improve the intent detection accuracy, as it shows a 1.2% absolute increase in accuracy when using our word embeddings. The convolutional network works better than the feedforward network. The LUIS solution with machine translation is usable, but its accuracy is lower

when compared with the systems that perform the intent detection in the original language. We note that for LUIS on the English datasets, we got slightly worse results than those reported in [39], even when repeating the test with their published scripts (https://github.com/sebischair/NLU-Evaluation-Scripts).

Table 6. Intent detection accuracy (in percentages) of the evaluated methods. NN [norm.] and NN are the feedforward neural networks using normalized and unnormalized versions of the sentence vectors, respectively. CNN is the convolutional network. Wiki and Tilde are the word embeddings trained on Wikipedia by Facebook and the word embeddings trained by us on the large internal corpus, respectively. LUIS (+MT) is the Microsoft LUIS with Microsoft machine translation service

Dataset	Language	NN [norm.]		NN		CNN		Wit.ai	LUIS (+MT)
		Wiki	Tilde	Wiki	Tilde	Wiki	Tilde		
askubuntu	EN	88.99	89.91	88.99	91.74	91.74	**92.66**	**92.66**	88.07
	ET	88.07	88.99	84.40	89.91	86.61	87.16	**90.83**	88.07
	LT	86.24	87.16	87.16	90.09	89.91	**91.74**	87.16	84.40
	LV	89.91	88.07	88.99	88.99	88.07	**90.83**	89.91	86.24
	RU	88.07	84.40	86.24	86.24	86.24	88.99	**89.91**	82.57
chatbot	EN	95.85	95.28	97.17	98.11	97.92	**98.49**	97.17	95.28
	ET	94.34	94.34	93.40	94.34	93.77	**95.28**	**95.28**	89.62
	LT	93.58	95.09	94.34	93.40	93.96	**96.79**	94.34	91.51
	LV	98.11	97.74	96.98	97.55	**100.00**	**100.00**	99.06	96.23
	RU	96.60	95.28	94.91	93.77	**97.17**	96.04	93.40	90.57
webapps	EN	76.27	74.58	81.36	77.97	78.31	**83.05**	81.36	77.97
	ET	72.88	72.88	69.49	76.27	69.49	**79.66**	76.27	74.58
	LT	74.58	74.58	74.58	77.97	73.22	81.36	**84.75**	79.66
	LV	69.49	81.36	74.58	81.36	67.80	**84.75**	74.58	83.05
	RU	83.05	**84.75**	83.05	**84.75**	74.92	81.69	77.97	79.66
Average		86.40	86.96	86.38	88.16	85.94	**89.90**	88.31	85.83

4.3. Comparison to Other Services on the English Language

Here, we compare the results of our best model with other popular chatbot services on the original English dataset. We use the results from [39] and [40] and add the results of our system and Wit.ai. As illustrated in Figure 3, our approach achieves accuracy results that are in line with other services.

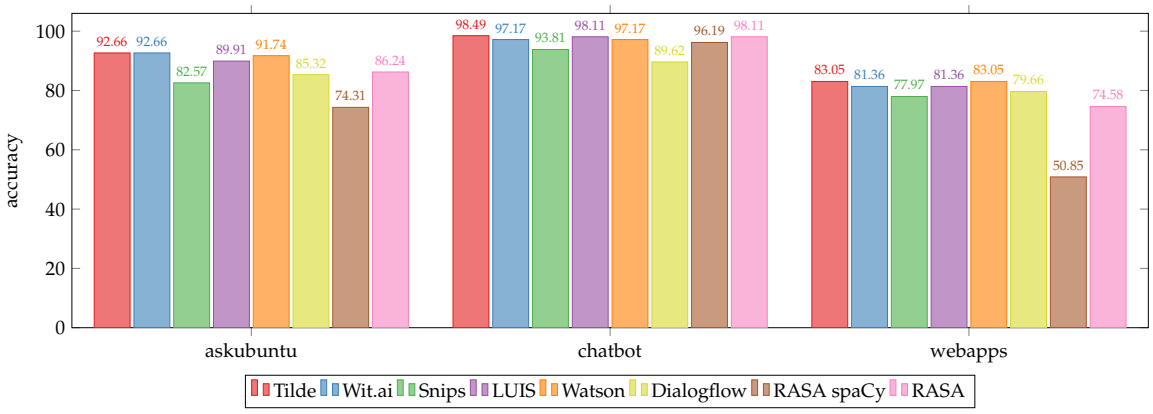

Figure 3. Intent detection accuracy (in percentages) of several dialogue system providers for the English language.

5. Latvian Language-Specific Error Correcting

In this section, we examine some error-correcting methods tailored to the Latvian language. We evaluate a system developed for chatroom text normalization and an approach of simplifying the language by removing the diacritical marks.

5.1. Dataset

We used a dataset in Latvian containing user questions addressed to customer service support personnel. There were 121 intents to be used for this dataset. The training set was constructed artificially. For every predefined intent, the experts created approximately 10 utterances, with 1231 utterances in total.

The test sets contained 236 utterances that were obtained by the following procedure. One thousand real user questions were randomly selected from the production system, and the most appropriate intent was assigned manually for each of them. This was possible for only 236 utterances, as the remaining part contained questions not related to the predefined intents (i.e., the majority of questions were off-topic). The test set *Real* contains the original user utterances. The test set *Fixed* contains the corrected versions of the same utterances, without grammatical errors.

5.2. Automatic Text Normalization

Language used in written communication on social media platforms or in Internet discussion forums can be quite distant from literary language. As many existing NLP tools are created on the basis of correct language, several authors have found it beneficial to normalize such user-created texts before further processing [42,43]. To find out if our intent detection module achieves better results if we first normalize the user utterances, in our previous research [25] we applied a module that corrects noisy chatroom text and compared the performance of intent detection on the real text and on the normalized text. In the above-mentioned module, three different tools were integrated—the regular expression module and the spelling and grammar checking modules. With the joint help of these modules, a single normalization hypothesis was provided for every utterance. The algorithm for applying the modules was simple—at first, the text was corrected by the regular expression module, then it was passed to the spelling checking module, which retained the first correction suggestion of every misspelled word (usually there are several suggestions), and finally, the text was corrected by the grammar checker.

In this research, we have replaced this simple module with a more advanced one that is based on machine learning methods. The creation details of the new model can be found in [44]. The model was trained using a random forest classifier that learned to rate normalization candidates for every word. For every word in an utterance, a list of candidates was generated by different means. As previously, we used the regular expressions and corrections suggestions generated by the spell-checking engine. The improved regular expression module now contained 799 regular expressions, allowing it to fix a wide range of typical chat language errors. We supplemented the spell-checking module with a slang lexicon, as it was previously built on the lexicon of general language. In the list of correction candidates, we included all the spelling suggestions generated by this module. We built a word embedding file on a general text corpus and included the 20 most similar words for every word in an utterance, based on cosine similarity. We also included the words with the same root. In addition, we used unigram and bi-gram lists to rate the popularity of a particular candidate, it's compatibility with the left and the right adjacent candidates, the length of the original word versus a particular candidate, and some other features. The suitability of the random forest classifier for the normalization task was based on its ability to consider different features, because errors found in chat language require different normalization actions. The previous normalization module shows 82.2% sentence-level accuracy. The sentence-level accuracy of the new normalization module is 86.02%. Although the real number is even better as several normalization variants are valid, there are five correctly normalized

sentences that does not match a human correction. The main mistakes of the normalization module are: (1) having a word that does not match the context used; (2) an incorrect inflectional form being used; (3) a proper noun being corrected to some other common noun.

5.3. Language Simplification

It is a common habit for many users to not use diacritics in written chat language. A large part of the Latvian language lexicon is formed from words with one or several diacritics. 47% of the words in the Latvian UD Treebank v2.0 contain diacritics [45]. Along with short vowels, four long vowels are used in Latvian: *ā, ē, ī, ū*. In written chat language, users might write them without a diacritic mark (*a, e, i, u*), or repeat the letter twice (*aa, ee, ii, uu*). The writing of consonants *ģ, ķ, ļ, ņ, č, ž*, and *š* is also inconsistent. They are written either without diacritics (*g, k, l, n, c, z, s*), or by adding an extra letter (*gj, kj, lj, nj, ch, zh, sh*).

For reducing the effect of such errors on intent detection quality, we tried to simplify the language. In the simplified training and test sets, all the letters containing diacritics and their common substitutions were replaced by a corresponding simple letter without the diacritic mark, according to the rules in Table 7. The corpora for training the word embeddings were also preprocessed in the same way.

This simplification introduces some ambiguity, as there are words that differ only by diacritics. However, it might also have the side-benefit of facilitating the training of the word embeddings, as there is a smaller number of possible subword units.

Table 7. Letter substitution rules used for language simplification.

Letters		Replace with
aa	ā	a
ch	č	c
ee	ē	e
gj	ģ	g
ii	ī	i
kj	ķ	k
lj	ļ	l
nj	ņ	n
	ō	o
	ŗ	r
sh	š	s
uu	ū	u
zh	ž	z

5.4. Results

We evaluated our best-performing model (the convolutional network with our word embeddings) on the differently error-corrected datasets. See the results in Table 8. Datasets *Autofix* and *Simplified* were created from the test set Real by applying the error-correcting methods described previously. The overall low accuracy can be explained by the heterogeneity of the datasets used for training and testing. We can see that the normalization of the text improves the intent detection accuracy. However, the improvement with language simplification seems to be very small.

Table 8. Results for error-corrected data.

Dataset	Accuracy (%)
Real	43.13
Simplified	43.59
Autofixed	44.84
Fixed	45.41

6. Conclusions

In this paper, we have shown an intent detection system based on the fastText vectorizer and two types of neural network classifiers. The convolutional network showed better performance than the shallow feedforward network. This approach works for many languages, provided that a large enough corpus for training the word embeddings is available. The quality of the word embeddings is important for intent detection accuracy. Therefore, training the word embeddings on a large corpus is preferred to using word embeddings trained on Wikipedia.

The fastText sentence vectorization can be modified by removing the normalization by vector length. This improves the intent detection accuracy and training convergence speed. This effect could be researched further on other natural language processing tasks where sentence embeddings are used.

The quality of intent detection in popular chatbot providers varies. Support for lesser-resourced languages is rare, and a solution with machine translation lacks quality. Therefore, if possible, it is better to create an intent detection system specifically for the required language.

In real-life conversations, users make grammatical errors that reduce intent detection accuracy. This effect can be mitigated by using automatic error-correction methods.

Author Contributions: D.D. contributed the section on the Latvian language-specific automatic error-correction. K.B. contributed everything else. Both authors have read and approved the final manuscript.

Funding: The research has been supported by the European Regional Development Fund within the project "Neural Network Modelling for Inflected Natural Languages" No. 1.1.1.1/16/A/215.

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

Appendix A. Analysing the Statistical Significance

Due to the small size of the test sets it is hard to draw statistically significant conclusions using a standard approach of reporting the accuracy and an error margin. Therefore we take a Bayesian approach. As we have not seen such analysis often in other papers, it seems that it warrants an additional discussion. Let us illustrate this with an example.

Consider two different cases where two systems, A and B, are compared on a test set containing 1000 examples (see Table A1). In each of the cases, the system A achieves accuracy 0.71 and the system B achieves accuracy 0.70 on the test set. In Case (a) there are 698 examples where both systems are correct and 288 examples where both systems are wrong. In Case (b) there are 500 examples where both systems are correct and 90 examples where both systems are wrong. Although in both cases the accuracies of the compared systems are the same, in Case (a) we can be quite confident that A performs better than B, but not so much in Case (b).

Table A1. The number of examples with the given results. X corresponds to an example where system X gave the correct answer, and $\neg X$—to an example where it gave an incorrect answer.

Case (a)			**Case (b)**		
	A	$\neg A$		A	$\neg A$
B	698	2	B	500	200
$\neg B$	12	288	$\neg B$	210	90

In Case (a), we have observed 6 times as many examples in the test data where A outperformed B, when compared with the number of examples where B outperformed A (12:2). This is very unlikely to have happened only by chance. We could reasonably expect that on a new example, that is similar to the test examples, the both systems will perform alike (either both will be correct, or both will be wrong). However, if there will be a discrepancy, it is much more likely that the system A will be correct and B—wrong.

In Case (b), the respective numbers (210:200) are relatively much closer to each other, while having the same absolute difference. It is not unreasonable to expect that on another test set that contains different 1000 examples we could obtain results that are the other way around (200:210).

Therefore, any method that analyzes the confidence that A outperforms B based only on their accuracies on the test set will either be underconfident in Case (a), or overconfident in Case (b). Hence, to more precisely compare two systems on a small dataset, it is important to remember that they are tested on the same test set and take into account the correlation of their errors.

Let T be a distribution of examples and let us assume that the test set comprises of examples $x \sim T$ drawn from this distribution. Let $A(x)$ be the intent given by a system A on an example x and $I(x)$ be the real intent of x.

Let $\mathrm{C}(A, x)$ denote the event that A gives the correct answer on x:

$$\mathrm{C}(A, x) = \begin{cases} 1, & \text{if } A(x) = I(x) \\ 0, & \text{otherwise} \end{cases}$$

Then we define the accuracy $\mathrm{Acc}(A) = \Pr_{x \sim T}[\mathrm{C}(A, x)]$ as the probability that A gives a correct answer on a random example. We estimate the accuracy of a system by evaluating it on a test set. We want to estimate the likelihood that one system is better than another, based on the observed results on the test set.

We say that system A is better than system B if $\mathrm{Acc}(A) > \mathrm{Acc}(B)$, i.e., if $\mathrm{Acc}(A) - \mathrm{Acc}(B) > 0$. Observe that:

$$\begin{aligned} \mathrm{Acc}(A) - \mathrm{Acc}(B) &= \Pr_{x \sim T}[\mathrm{C}(A, x) \wedge \mathrm{C}(B, x)] \cdot 0 \\ &\quad + \Pr_{x \sim T}[\mathrm{C}(A, x) \wedge \neg\mathrm{C}(B, x)] \cdot 1 \\ &\quad + \Pr_{x \sim T}[\neg\mathrm{C}(A, x) \wedge \mathrm{C}(B, x)] \cdot (-1) \\ &\quad + \Pr_{x \sim T}[\neg\mathrm{C}(A, x) \wedge \neg\mathrm{C}(B, x)] \cdot 0 \\ &= \Pr_{x \sim T}[\mathrm{C}(A, x) \wedge \neg\mathrm{C}(B, x)] \cdot 1 \\ &\quad + \Pr_{x \sim T}[\neg\mathrm{C}(A, x) \wedge \mathrm{C}(B, x)] \cdot (-1). \end{aligned}$$

Notice that the difference in accuracy only depends on the cases where one system gives a correct answer and the other one – an incorrect, but not on the cases where both systems are correct or both are wrong.

Let $p_A = \Pr_{x \sim T}[\mathrm{C}(A, x) \wedge \neg\mathrm{C}(B, x)]$ and $p_B = \Pr_{x \sim T}[\neg\mathrm{C}(A, x) \wedge \mathrm{C}(B, x)]$ be the probability that A outperforms B, and the probability that B outperforms A, respectively, on a random example. Therefore, $\mathrm{Acc}(A) > \mathrm{Acc}(B)$ is equivalent to $p_A > p_B$.

$p_A \in [0, 1]$ is an unknown quantity that we want to estimate. Before observing any results, we assume that p_A is uniformly distributed in the interval $[0, 1]$ (i.e., we assume a non-informative prior). That corresponds to $p_A \sim \mathrm{Beta}(1, 1)$, where Beta is the beta distribution. Then, after observing S successes (examples x, where $\mathrm{C}(A, x) \wedge \neg\mathrm{C}(B, x)$) and F failures (all other examples x) on the test set, p_A is distributed according to the posterior distribution $p_A \sim \mathrm{Beta}(1 + S, 1 + F)$. We can apply a similar reasoning for p_B.

Let us return to the above-mentioned cases.

In Case (a), after the observations on the test set: $p_A \sim \mathrm{Beta}(1 + 12, 1 + 988)$, $p_B \sim \mathrm{Beta}(1 + 2, 1 + 998)$.

In Case (b): $p_A \sim \mathrm{Beta}(1 + 210, 1 + 790)$, $p_B \sim \mathrm{Beta}(1 + 200, 1 + 800)$.

As shown in Figure A1, the probability density functions overlap more significantly in Case (b) than in Case (a). The probability that $p_A > p_B$ is much larger in Case (a).

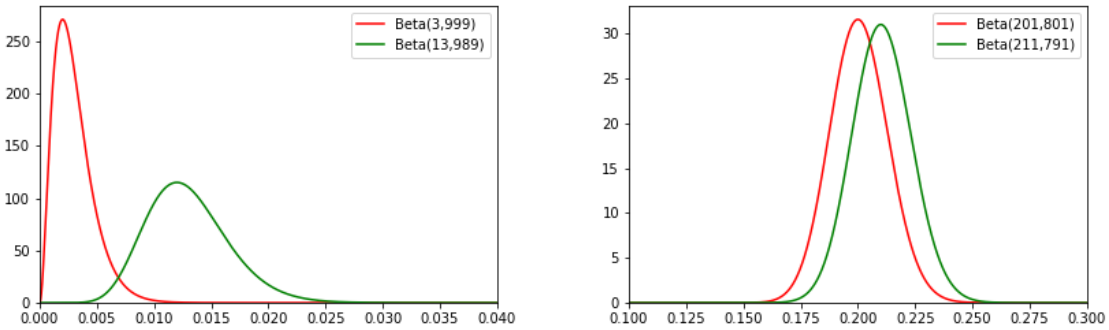

Figure A1. The overlap of probability density functions of the Beta functions corresponding to p_A and p_B in Case (a) (**left**) and Case (b) (**right**).

Conveniently, if $p_A \sim Beta(\alpha_A, \beta_A)$, $p_B \sim Beta(\alpha_B, \beta_B)$, and all parameters are integers, then the probability of $p_A < p_B$ can be evaluated in a closed form (http://www.evanmiller.org/bayesian-ab-testing.html)

$$\Pr[p_B > p_A] = \sum_{i=0}^{\alpha_B - 1} \frac{B(\alpha_A + i, \beta_A + \beta_B)}{(\beta_B + i) B(1 + i, \beta_B) B(\alpha_A, \beta_A)}$$

where B is the beta function:

$$B(x, y) = \int_0^1 t^{x-1} (1-t)^{y-1} dt$$

However, this approach is slightly inaccurate, as p_A is not entirely independent of p_B (for example, $p_A + p_B \leq 1$). More precisely, on each test example there are three possible outcomes we are interested in: A is correct and B is wrong; B is correct and A ir wrong; and both are correct or both are wrong. Therefore, p_A and p_B (and the probability that both systems perform the same on a test example) are distributed according to the trivariate Dirichlet distribution $\mathrm{Dir}(\alpha_1, \alpha_2, \alpha_3)$.

Assuming a non-informative prior $\mathrm{Dir}(1, 1, 1)$, after observing a total of N examples of which N_A and N_B is the number of examples where A outperformed B, and B outperformed A, respectively, the posterior distribution is $\mathrm{Dir}(1 + N_A, 1 + N_B, 1 + N - N_A - N_B)$.

If we consider only in the cases where one system outperforms another, they are distributed according to $\mathrm{Beta}(1 + N_A, 1 + N_B)$. The likelihood that $p_A > p_B$ corresponds to the probability that a variable p drawn from $\mathrm{Beta}(1 + N_A, 1 + N_B)$ has $p > 0.5$.

We can calculate that $\Pr[p \sim \mathrm{Beta}(13, 3) > 0.5] \approx 0.9963$ and $\Pr[p \sim \mathrm{Beta}(211, 201) > 0.5] \approx 0.6891$, therefore, confirming our previous intuitive reasoning that in Case (a) we can be much more confident that the system A is better than B (see Figure A2 for illustration).

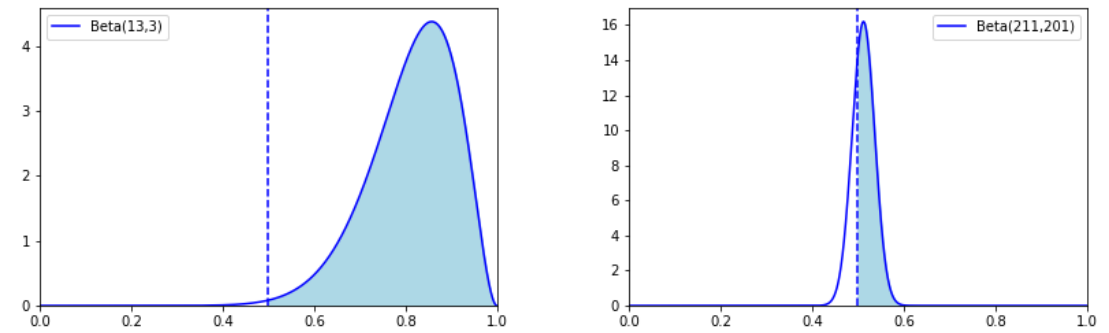

Figure A2. The probability distributions corresponding to the Case (a) (**left**) and Case (b) (**right**).

In our paper, the datasets used are quite small. Therefore, we consider the combined results on all 3 datasets and all 5 languages for each of the systems. This results in a total of $1370 = 5 \times (109 + 106 + 59)$ test examples to compare the systems on.

We report the numbers N_A and N_B for all the compared models in Table A2. They correspond to the probabilities that $p_A > p_B$ given in Table A3.

Table A2. N_A and N_B for pairs of models, where A is the model corresponding to the row and B is the model corresponding to the column. Models are listed in the same order as in Table 6.

	NN[n] W	**NN[n] T**	**NN W**	**NN T**	**CNN W**	**CNN T**	**Wit**	**LUIS**
NN[n] W	0-0	28-28	26-21	28-44	37-34	16-52	29-48	76-55
NN[n] T	28-28	0-0	39-34	8-24	44-41	13-49	33-52	73-52
NN W	21-26	34-39	0-0	27-48	28-30	17-58	28-52	78-62
NN T	44-28	24-8	48-27	0-0	52-33	17-37	39-42	77-40
CNN W	34-37	41-44	30-28	33-52	0-0	12-51	33-55	76-58
CNN T	52-16	49-13	58-17	37-17	51-12	0-0	43-26	84-27
Wit	48-29	52-33	52-28	42-39	55-33	26-43	0-0	81-41
LUIS	55-76	52-73	62-78	40-77	58-76	27-84	41-81	0-0

Table A3. Comparison of the models. In row A and column B—the probability (in percent) that model A is better than model B. Probabilities greater than 0.95 (or smaller than 0.05) are shown in bold.

	NN[n] W	**NN[n] T**	**NN W**	**NN T**	**CNN W**	**CNN T**	**Wit**	**LUIS**
NN[n] W	50.00	50.00	76.46	**3.02**	63.80	**0.00**	**1.54**	**96.64**
NN[n] T	50.00	50.00	71.93	**0.23**	62.67	**0.00**	**1.99**	**96.95**
NN W	23.54	28.07	50.00	**0.77**	39.74	**0.00**	**0.36**	91.12
NN T	**96.98**	**99.77**	**99.23**	50.00	**98.01**	**0.32**	37.03	**99.97**
CNN W	36.20	37.33	60.26	**1.99**	50.00	**0.00**	**0.96**	93.95
CNN T	**100.00**	**100.00**	**100.00**	**99.68**	**100.00**	50.00	**97.93**	**100.00**
Wit	**98.46**	**98.01**	**99.64**	62.97	**99.04**	**2.07**	50.00	**99.99**
LUIS	**3.36**	**3.05**	8.88	**0.03**	6.05	**0.00**	**0.01**	50.00

There are some limitations of applying this approach to our data. The limitations include the following:

- When combining the results on all datasets, all test examples are treated equally. However, the test set for webapps is considerably smaller than the test sets for askubuntu and chatbot. Therefore this dataset is underrepresented in the test set, while the other two are overrepresented.
- This approach assumes that the test examples are independent of each other. However, this is not true, because, for different languages, the examples are translations of each other.
- This approach works for comparing two systems. Increasing the number of different systems, increases the probability that some system will get better results simply by chance.

However, we still believe that this approach gives a reasonable estimate of the confidence that one system outperforms another.

References

1. 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, Phoenix, AZ, USA, 12–17 February 2016; Volume 16, pp. 3776–3784.
2. Vinyals, O.; Le, Q. A neural conversational model. *arXiv* **2015**, arXiv:1506.05869.
3. Shang, L.; Lu, Z.; Li, H. Neural responding machine for short-text conversation. *arXiv* **2015**, arXiv:1503.02364.
4. Yang, X.; Chen, Y.N.; Hakkani-Tür, D.; Crook, P.; Li, X.; Gao, J.; Deng, L. End-to-end joint learning of natural language understanding and dialogue manager. In Proceedings of the International Conference on Acoustics, Speech and Signal Processing (ICASSP), New Orleans, LA, USA, 5–9 May 2017; pp. 5690–5694.

5. Wen, T.; Vandyke, D.; Mrkšíc, N.; Gašíc, M.; Rojas-Barahona, L.; Su, P.; 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, Valencia, Spain, 3–7 April 2017; Volume 1, pp. 438–449.
6. 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, Association for Computational Linguistics, Lisboa, Portugal, 17–21 September 2015; pp. 1711–1721. [CrossRef]
7. 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; p. 285.
8. Kalchbrenner, N.; Blunsom, P. Recurrent convolutional neural networks for discourse compositionality. *arXiv* **2013**, arXiv:1306.3584.
9. 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, Dresden, Germany, 6–10 September 2015.
10. Wallace, R. *The Elements of AIML Style*; Alice AI Foundation: New York, NY, USA, 2003.
11. Shawar, B.A.; Atwell, E. Machine Learning from dialogue corpora to generate chatbots. *Expert Update J.* **2003**, *6*, 25–29.
12. Neves, A.M.M.; Barros, F.A.; Hodges, C. iAIML: A Mechanism to Treat Intentionality in AIML Chatterbots. In Proceedings of the 18th IEEE International Conference on Tools with Artificial Intelligence, Washington, DC, USA, 13–15 November 2006; IEEE Computer Society: Washington, DC, USA, 2006; pp. 225–231. [CrossRef]
13. Fonte, F.; Carlos, J.; Rial, B.; Nistal, M.L. Tq-bot: An AIML-based tutor and evaluator bot. *J. Univ. Comput. Sci.* **2009**, *15*, 1486–1495.
14. McCallum, A.; Nigam, K. A Comparison of Event Models for Naive Bayes Text Classification. In Proceedings of the AAAI-98 Workshop on Learning for Text Categorization, Madison, WI, USA, 26–27 July 1998; Volume 752, pp. 41–48.
15. Joachims, T. Transductive Inference for Text Classification Using Support Vector Machines. In Proceedings of the Sixteenth International Conference on Machine Learning, Bled, Slovenia, 27–30 June 1999; Volume 99, pp. 200–209.
16. Lodhi, H.; Saunders, C.; Shawe-Taylor, J.; Cristianini, N.; Watkins, C. Text Classification Using String Kernels. *J. Mach. Learn. Res.* **2002**, *2*, 419–444.
17. Xu, P.; Sarikaya, R. Convolutional neural network based triangular CRF for joint intent detection and slot filling. In Proceedings of the 2013 IEEE Workshop on Automatic Speech Recognition and Understanding (ASRU), Olomouc, Czech Republic, 8–12 December 2013; pp. 78–83.
18. Liu, B.; Lane, I. Attention-based recurrent neural network models for joint intent detection and slot filling. *arXiv* **2016**, arXiv:1609.01454.
19. Yao, K.; Peng, B.; Zhang, Y.; Yu, D.; Zweig, G.; Shi, Y. Spoken language understanding using long short-term memory neural networks. In Proceedings of the Spoken Language Technology Workshop (SLT), South Lake Tahoe, NV, USA, 7–10 December 2014; pp. 189–194.
20. Howard, J.; Ruder, S. Universal language model fine-tuning for text classification. *arXiv* **2018**, arXiv:1801.06146.
21. Peters, M.E.; Neumann, M.; Iyyer, M.; Gardner, M.; Clark, C.; Lee, K.; Zettlemoyer, L. Deep contextualized word representations. *arXiv* **2018**, arXiv:1802.05365.
22. Devlin, J.; Chang, M.W.; Lee, K.; Toutanova, K. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv* **2018**, arXiv:1810.04805.
23. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient estimation of word representations in vector space. *arXiv* **2013**, arXiv:1301.3781.
24. Bojanowski, P.; Grave, E.; Joulin, A.; Mikolov, T. Enriching Word Vectors with Subword Information. *Trans. Assoc. Comput. Linguist.* **2017**, *5*, 135–146. [CrossRef]
25. Balodis, K.; Deksne, D. Intent Detection System Based on Word Embeddings. In *Artificial Intelligence: Methodology, Systems, and Applications, Proceedings of the International Conference on Artificial Intelligence: Methodology, Systems, and Applications, Varna, Bulgaria, 12–14 September 2018*; Springer: Berlin, Germany, 2018; pp. 25–35.

26. Denny, M.J.; Spirling, A. Text preprocessing for unsupervised learning: Why it matters, when it misleads, and what to do about it. *Polit. Anal.* **2018**, *26*, 168–189. [CrossRef]
27. Uysal, A.K.; Gunal, S. The impact of preprocessing on text classification. *Inf. Process. Manag.* **2014**, *50*, 104–112. [CrossRef]
28. Srividhya, V.; Anitha, R. Evaluating preprocessing techniques in text categorization. *Int. J. Comput. Sci. Appl.* **2010**, *47*, 49–51.
29. Pennington, J.; Socher, R.; Manning, C. Glove: Global vectors for word representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), Doha, Qatar, 25–29 October 2014; pp. 1532–1543.
30. Conneau, A.; Kiela, D.; Schwenk, H.; Barrault, L.; Bordes, A. Supervised Learning of Universal Sentence Representations from Natural Language Inference Data. In Proceedings of the 2017 Conference on Empirical Methods in Natural Language Processing, Copenhagen, Denmark, 7–11 September 2017; Association for Computational Linguistics: Copenhagen, Denmark, 2017; pp. 670–680.
31. Kim, J.K.; Tur, G.; Celikyilmaz, A.; Cao, B.; Wang, Y.Y. Intent Detection Using Semantically Enriched Word Embeddings. In Proceedings of the 2016 IEEE Spoken Language Technology Workshop (SLT), San Diego, CA, USA, 13–16 December 2016; pp. 414–419.
32. Athiwaratkun, B.; Wilson, A.G.; Anandkumar, A. Probabilistic Fasttext for Multi-Sense Word Embeddings. *arXiv* **2018**, arXiv:1806.02901.
33. Risch, J.; Krestel, R. Learning Patent Speak: Investigating Domain-Specific Word Embeddings. In Proceedings of the Thirteenth International Conference on Digital Information Management (ICDIM 2018), Porto, Portugal, 19–21 September 2018.
34. Nooralahzadeh, F.; Øvrelid, L.; Lønning, J.T. Evaluation of Domain-specific Word Embeddings using Knowledge Resources. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC-2018), Miyazaki, Japan, 7–12 May 2018.
35. Joulin, A.; Grave, E.; Bojanowski, P.; Mikolov, T. Bag of tricks for efficient text classification. *arXiv* **2016**, arXiv:1607.01759.
36. 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.
37. Mikolov, T.; Grave, E.; Bojanowski, P.; Puhrsch, C.; Joulin, A. Advances in Pre-Training Distributed Word Representations. In Proceedings of the International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, 7–12 May 2018.
38. 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.
39. Braun, D.; Hernandez-Mendez, 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.
40. Coucke, A.; Saade, A.; Ball, A.; Bluche, T.; Caulier, A.; Leroy, D.; Doumouro, C.; Gisselbrecht, T.; Caltagirone, F.; Lavril, T.; et al. Snips Voice Platform: An embedded Spoken Language Understanding system for private-by-design voice interfaces. *arXiv* **2018**, arXiv:1805.10190.
41. Pinnis, M.; Rikters, M.; Krišlauks, R. Tilde's Machine Translation Systems for WMT 2018. In Proceedings of the Third Conference on Machine Translation: Shared Task Papers, Belgium, Brussels, 31 October–1 November 2018; pp. 473–481.
42. Damnati, G.; Auguste, J.; Nasr, A.; Charlet, D.; Heinecke, J.; Béchet, F. Handling Normalization Issues for Part-of-Speech Tagging of Online Conversational Text. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, 7–12 May 2018; European Language Resources Association (ELRA): Paris, France, 2018.
43. van der Goot, R.; van Noord, G. MoNoise: Modeling Noise Using a Modular Normalization System. *Comput. Linguist. Neth. J.* **2017**, *7*, 129–144.

44. Deksne, D. Chat Language Normalisation using Machine Learning Methods. In Proceedings of the 11th International Conference on Agents and Artificial Intelligence (ICAART 2019), Prague, Czech Republic, 19–21 February 2019.
45. Náplava, J.; Straka, M.; Straňák, P.; Hajič, J. Diacritics Restoration Using Neural Networks. In Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018), Miyazaki, Japan, 7–12 May 2018; European Language Resources Association (ELRA): Paris, France, 2018.