Development of neural network based rules for confusion set disambiguation in LanguageTool

Markus Brenneis and Sebastian Krings

Institut für Informatik, Heinrich-Heine-Universität Düsseldorf
Universitässtraße 1, D-40225 Düsseldorf
Markus.Brenneis@uni-duesseldorf.de, krings@cs.uni-duesseldorf.de

Abstract. Confusion set disambiguation is a typical task for grammar checkers like the free LanguageTool. In this paper we present a neural network based approach which has low memory requirements, high precision with decent recall, and can easily be integrated into LanguageTool. What is more, adding support for new confusion pairs does not need any knowledge of the target language. We examine different sampling techniques and neural network architectures and compare our approach with an existing memory-based algorithm.

1 Introduction

Grammar checkers are used to detect errors which cannot be detected by a simple spell-checker, e. g. confusion of words and agreement errors. We have developed rules for confusion set disambiguation based upon neural networks and integrated them into the existing grammar checker LanguageTool.

1.1 LanguageTool

LanguageTool is free, open-source and rule-based grammar and style checker originally developed by Naber 2003 and written in Java. The majority of rules are manually written in either XML or Java, hence rule development requires knowledge of the target language. The program is available as stand-alone version and can be used in several other applications like LibreOffice and TeXstudio.

When a text is checked, LanguageTool uses its own language-specific sentence splitter, tokenizer and part-of-speech tagger to assign part-of-speech texts to every token in the input. Each sentence is then checked against the style and grammar rules.

1.2 Confusion Set Disambiguation

A typical type of mistake which is not detectable by a spell checker are confused words. Confusion set disambiguation is the task of choosing the right word from a finite set of words (e. g. $\{\text{to}, \text{too}, \text{two}\}$). In this paper, we will focus a confusion sets with exactly two tokens t and t'.

LanguageTool already supports detecting confused words. Currently, there are basically two types of rules: Pattern rules written in XML or Java, which are usually created by hand which is time-consuming and prone to errors.

As an alternative, there are 3-gram based rules, which require a copy of a large 3-gram corpus (e. g. 10 trillion tokens for English, stored in a 11 GB database) which bases upon the Google n-gram corpus[1]. The error detection algorithm is memory-based and works as follows: Let t be a token in a confusion pair (t, t') and t_{+n} the nth token after t in the text being checked. When t is encountered in the text, the number of occurrences n of the 3-grams (t_{-2}, t_{-1}, t), (t_{-1}, t, t_{+1}), and (t, t_{+1}, t_{+2}) are counted and compared with the number of occurrences n' of the same 3-grams containing t' instead of t. If n' is x times greater than n (where a suitable x with good precision and recall is determined beforehand), t is considered incorrect.

The 3-gram based rules have the advantage that rules have not to be written manually. On the other hand, there are several disadvantages: First, the rules fail to detect errors if the exact 3-gram is not part of the corpus. For instance, the mistake in *We go *too Gimli's birthday party.* is not detected, because the 3-grams (*We go to*), (*go to Gimli*), and (*to Gimli 's*) are not part of the corpus, although the individual tokens are.

Furthermore, the user of LanguageTool needs to download a big corpus in order to use the rules and must have a sufficiently fast hard drive and enough memory, in order not to slow down the process of text checking too much.

1.3 Related work

Miłkowski 2012 has studied automatic and semi-automatic creation of symbolic rules using transformation-based learning. The created rules have very good recall, but often suffer from a low precision, i. e. cause many false alarms, unless there is human intervention.

Banko and Brill 2001 compared different classifiers for the confusion set task with regard to their performance if the training corpus is increased from 1 million words to 1 billion words. They have shown that a memory based algorithm is outperformed by a more complex perceptron algorithm when the training corpus has more than 1 million words.

1.4 Goals of our work

The goal of our work was to develop confusion set disambiguation rules for LanguageTool which also work in contexts which are not part of the training corpus, work without having to save and load several gigabytes of data and do not cause too many false alarms.

In the following section we will introduce our neural network architectures and the training process. Afterwards we compare our classifiers and the existing memory-based 3-gram rules with regard to precision, memory usage and speed.

[1] `http://storage.googleapis.com/books/ngrams/books/datasetsv2.html`. Accessed 12 Nov. 2017.

2 Model architecture and training process

2.1 Data set

Neural network training and rule testing need a large corpus which can be considered to have no or at least very few mistakes. As shown by ibid., using larger data sets can improve the performance of a classifier significantly. Furthermore, some words like second person verb forms can only seldom be found in some corpora, for example newspaper articles. Thus, a corpus with sentences randomly chosen from newspaper articles from Project Deutscher Wortschatz[2] and sentences from Tatoeba[3] has been created. The final corpus contains more than 30,000,000 words and has been divided in a training (90 %) and testing set (10 %).

The corpus has been tokenized using the tokenizer of LanguageTool.

2.2 Sampling

It is often the case that one word of a confusion set occurs several times more often in the training corpus than the other word. Considering the German confusion set {wider, wieder}, there are around 40.000 sentences containing "wieder" in the training corpus, but only 471 sentences with "wider". Our experiments have shown that this class imbalance leads to heavy overfitting, since the classifier is biased towards the majority class.

To overcome the issue of class imbalance, we compared three different approaches which can commonly be found in research (cf. Chawla 2009): Random undersampling, random oversampling, and a combination of over- and undersampling. In the latter case, the oversampling has been limited to a factor of 2, and the majority class has been undersampled such that the class label ratio is 1. This approached seemed to be feasible because we did not want to throw away too many training samples as in undersampling, but we also wanted to prevent the classifier to overfit on the few samples of the minority class.

2.3 Neural network architecture

The artificial neural network gets the two tokens before and after a confusion word candidate as input. It outputs a number for each token in the confusion set, which can be interpreted as the logits, i.e. the logarithm of the odd $\log\left(\frac{p}{1-p}\right)$, where p is the probability for the corresponding token to be correct in the given context.

A vector representation using the word2vec model by Mikolov et al. 2013 with 64 dimensions is used for the word tokens. In this vector space model, words with similar meaning are mapped to vectors which are close to each other, which enables the neural network to detect errors in contexts it has not seen before. All words which appeared at least five times in the training corpus are part of the

[2] `http://wortschatz.uni-leipzig.de/en/download`. Accessed 1 Nov. 2017.
[3] `https://tatoeba.org/eng/downloads`. Accessed 1 Nov. 2017.

word2vec model's dictionary; this way, the model is kept small by ignoring less frequently used words and possible typos in the training corpus, which probably do not occur very often, are excluded. Words which are not part of the dictionary are replaced by the special token "UNKNOWN".

Our main architecture is a single layer network without any hidden layers and activation function, i. e. a linear model (called "NN"). For comparison, we also trained a network with one hidden layer with 8 neurons and ReLU activation function ("NNH") and variants which get only two tokens from the context as input ("NN2" and "NNH2", respectively). We did not train any deep models or models with large hidden layers because our goal was to create compact rules.

All architectures use the Adam Optimizer by Kingma and Ba 2014 to minimize the softmax cross entropy loss

$$L = -\log\left(\frac{e^{y_i}}{e^{y_i} + e^{y_j}}\right)$$

where y_i is the output for the correct label and y_j the output for the wrong label.

2.4 Output interpretation

The output (y, y') of the neural network is used as follows: Given a threshold $\theta \in \mathbb{R}^+$, the token t of the confusion set is considered incorrect and t' is considered correct, if and only if $y < -\theta$ and $y' > \theta$ (i. e. the network thinks t' is much more likely than t and t' seems to fit).

If we assume that

$$p_t = \frac{1}{1 + e^{-y}} \qquad\qquad p_{t'} = \frac{1}{1 + e^{-y'}}$$

are the probabilities that the first or second token are correct, respectively, then the aforementioned approach is equivalent to saying that $p_t < 0.5 + \sigma$ and $p'_t > 0.5 - \sigma$, with

$$\sigma = \frac{1}{1 + e^{-\theta}} - 0.5 \in [0, 0.5)$$

i. e. t' is considered at least 2σ more probable to be correct than t and $p_t < 0.5$ and $p_{t'} > 0.5$.

The practical advantage of the first criterion is that it requires fewer calculations, and is therefore used in our implementation.

3 Rule quality and comparison

In this section we will have a look at the quality of the rules with regard to precision and recall, comparing our different architectures and the existing 3-gram-based rules.

3.1 Precision and Recall

In order to be useful for a grammar checking application, the neural network based rules must not cause any or at least very few false alarms. In the context of the error detection task, we define true positives (tp), true negatives (tn), false positives (fp) and false negatives (fn) as depicted in table 1.

	marked as error	not marked as error
correct usage	fp	tn
incorrect usage	tp	fn

Table 1. Definition of true positives, true negatives, false positives, false negatives

Note that a true positive is an incorrect usage of a token which is marked as error, and not solely the case where the neural network would choose the right token (which is $tp + tn$).

For each rule created for a confusion pair (t, t'), we checked it against up to 5,000 sentences containing t and 5,000 sentences containing t' from the test set and calculated precision P and recall R for different thresholds θ.

$$P = \frac{tp}{tp + fp} \qquad\qquad R = \frac{tp}{tp + fn}$$

A rule is considered good if $P > 0.99$ (i. e. the probability for false alarms is less than 1 %) and $R > 0.5$ (i. e. more than 50 % of incorrect usages are detected as error).

3.2 Comparison of network architectures

The neural network architectures show different recall at the same level of precision on the test corpus. In general, looking at different confusion pairs, the architectures having 2 tokens as input have for a fixed precision lower recall than the corresponding architecture with 4 input tokens.

Moreover, the architectures with hidden layer perform better than those without hidden layer. Whether "NN" or "NNH2" performed better depended on the confusion set. The distance between the smaller "NN" architecture and the larger "NNH" within the interesting precision interval $[0.99, 0.995]$ has, in general, been rather small.

3.3 Comparison of sampling techniques

We also had a look on how different sampling methods during the training process influenced the performance on the test set. Figure 2 shows precision and recall for the $\{\text{wider}, \text{wieder}\}$ confusion pair using the "NN" architecture. For

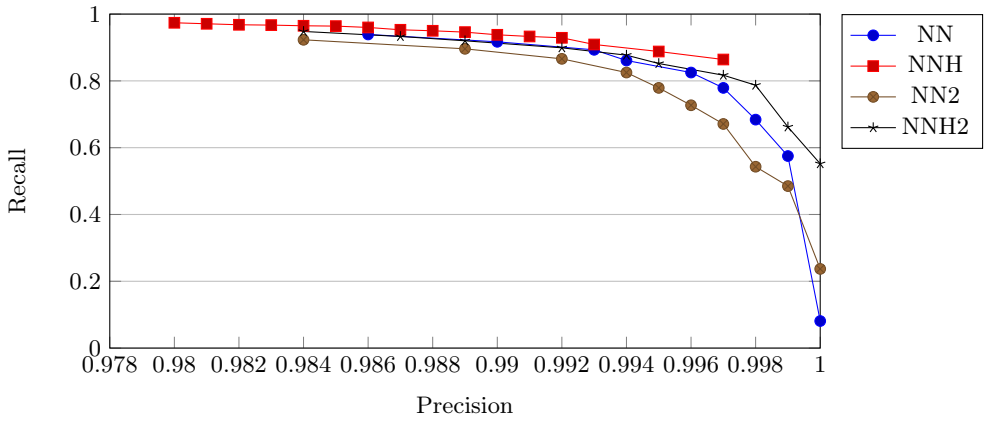

Fig. 1. Precision and recall for different network architectures for the confusion pair {to, too}

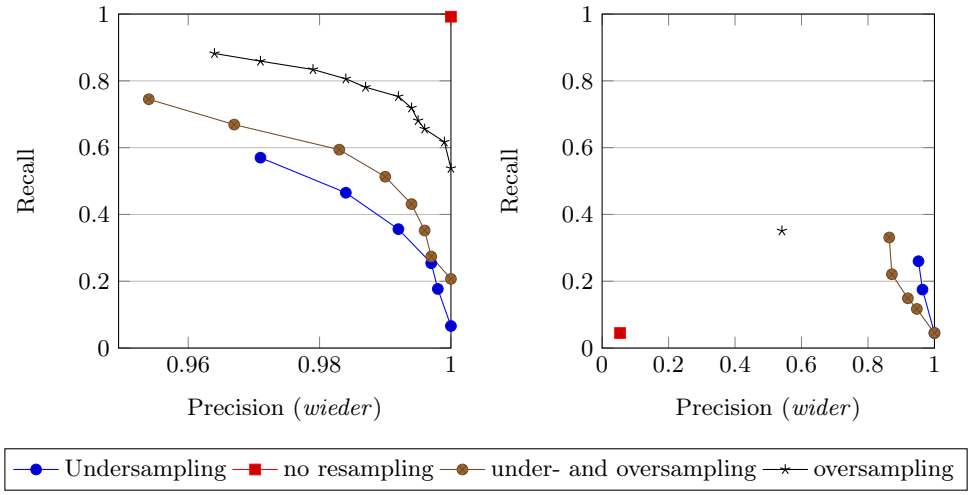

Fig. 2. Precision and recall for the confusion pair {wieder, wider}

the diagram for *wieder*, only sentences where *wieder* is correct has been used, i. e. sentences with correct usage of *wieder* and sentences with incorrect usage of *wider*.

While there are decent results for detecting the right use of the more common word *wieder* when oversampling is used, the recall for the around 100 times less common *wider* is much worse, with a maximum precision of around 0.5, probably due to overfitting. If no resampling is used, the network is very good at dealing with contexts where *wieder* is correct, but has very low precision in contexts where *wider* must be used. Using a mixture of over- and undersampling produces relatively close results, where undersampling is worse for the recall of the more common *wieder* case and better for the less common *wider*.

For other imbalanced confusion pairs like {to, too} (factor 10), the differences have not been that big, such that undersampling has been used in the other experiments.

3.4 Comparison with 3-gram rules

confusion pair	$R_{\text{3-gram}}$	R_{NN}	confusion pair	$R_{\text{3-gram}}$	R_{NN}
and/end	0.81	**0.84**	da/dar	**0.82**	0.74
five/give	**0.97**	0.94	das/dass	0.43	**0.81**
it/its	**0.95**	0.92	den/denn	0.70	**0.90**
our/out	**0.98**	0.93	fielen/vielen	0.85	**0.94**
then/the	0.45	**0.57**	ihm/im	**0.96**	0.94
to/too	0.82	**0.95**	schon/schön	**0.73**	0.44
some/same	**0.99**	0.98	seid/seit	**0.98**	0.93

Table 2. Comparison of recall at $P = 0.99$ for some English and German confusion pairs.

As our goal was to be at least as good as the existing 3-gram rules, we also compared the performance of our system with the existing rules. Note, however, that the comparison is not completely accurate, since the 3-gram rules use a different tokenization algorithm, which is compatible with Google's n-gram database. For instance, the 3-gram rules can detect the error in **give-year-old*, because this expression consists of 5 tokens according to the Google style tokenizer, whereas our rules fail to detect the error, since the expression is one token for the LanguageTool tokenizer. So to not end up with a lot of "false" false negatives for our rules, we changed the testing algorithm to exclude those cases.

The results depicted in table 2 show that our rules have, on average, a performance comparable to those using the memory-based 3-gram rules.

3.5 Memory usage and runtime performance

The files for the word2vec embedding for English have a size of around 65 MB (uncompressed). The weight files for the "NN" architecture have a size of 13 KB for each confusion pair. Thus, around 800,000 neural network based rules need the same amount of storage memory as the 3-gram corpus, which is stored as 11 GB Lucene index.

The start-up wall-clock time of the LanguageTool standalone GUI without 3-gram and neural network rules, from the start till the English example sentence is checked, is about 4.8 seconds on our test system with SSD. If only the 3-gram rules are enabled, the start-up time is 1.2 seconds longer, with only neural network rules enabled, the time is 1.5 seconds longer.

The memory usage of the GUI 10 seconds after start-up and a garbage collection call is around 80 MB without 3-gram and neural network rules, 130 MB with 3-gram rules enabled and 100 MB with neural network rules loaded.

Checking a German text with around 3,000 words using the command line version of LanguageTool takes around 2.9 seconds with both rule types disabled, 4.5 seconds with 76 3-gram rules enabled and 3.0 seconds with 29 neural network rules ("NN" architecture) enabled.

To sum up, the calculation done by the neural network code have a lower impact on the performance than the 3-gram lookup, and storing as well as loading the 3-gram index requires more memory than the word2vec model and the neural network data.

4 Conclusion

In this paper we have presented a new kind of rule for the free style and grammar checker LanguageTool which uses neural networks, and tested them successfully on a confusion set disambiguation task. The rule quality is similar to the memory based rules which are already part of LanguageTool, but our rules require less memory and are faster. Hence our rule can be used instead or in addition to the existing 3-gram rules. It has to be noted, though, that creating new neural network based rules requires several minutes of computation time for the training process, which is not needed for a new 3-gram rule.

Possible next steps include using information from the part-of-speech tagger to handle words which are not part of the training vocabulary more appropriately. Furthermore, the neural network architecture can easily be extended to support bigger confusion sets, such that rules for $\{\text{to}, \text{too}\}$, $\{\text{to}, \text{two}\}$ and $\{\text{two}, \text{too}\}$ can be merged in one $\{\text{to}, \text{too}, \text{two}\}$ rule. Adding support for confusion sets containing larger expressions instead of single tokens (e. g. $\{\text{das}, \text{dass},\}$ or $\{\text{in dem}, \text{indem}\}$) is also planned. In addition, training on even larger corpora might further improve the performance.

What is more, a totally different architecture using recurrent neural networks, which have e. g. successfully been used for machine translation by Bahdanau et al. 2014.

Acknowledgements

Computational support and infrastructure was provided by the "Center for Information and Media Technology" (ZIM) at the University of Düsseldorf (Germany). We also thank the LanguageTool community for the feedback during the integration of the new rules into LanguageTool.

References

Bahdanau, Dzmitry, Kyunghyun Cho, and Yoshua Bengio (2014). "Neural machine translation by jointly learning to align and translate". In: *arXiv preprint arXiv:1409.0473*.

Banko, Michele and Eric Brill (2001). "Scaling to very very large corpora for natural language disambiguation". In: *Proceedings of the 39th annual meeting on association for computational linguistics*. Association for Computational Linguistics, pp. 26–33.

Chawla, Nitesh V (2009). "Data mining for imbalanced datasets: An overview". In: *Data mining and knowledge discovery handbook*. Springer, pp. 875–886.

Kingma, Diederik and Jimmy Ba (2014). "Adam: A method for stochastic optimization". In: *arXiv preprint arXiv:1412.6980*.

Mikolov, Tomas, Kai Chen, Greg Corrado, and Jeffrey Dean (2013). "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781*.

Miłkowski, Marcin (2012). "Automating rule generation for grammar checkers". In: *arXiv preprint arXiv:1211.6887*.

Naber, Daniel (2003). "A rule-based style and grammar checker". In: