Negation Scope Resolution: Quantifying Neural Uncertainty In An Imbalanced Setting

Chris Ghai

October 7, 2019

Negation Scope Resolution: Quantifying Neural Uncertainty In An Imbalanced Setting

Chris Ghai
Language Technology Group, Department of Informatics / University of Oslo
chrisgh@math.uio.no

Abstract

Negation scope detection is an interesting task for neural machine learning models, because of the sequential dependencies in the input data. Having a neural classifier being able to untangle negated parts of a sentence from the non-negated part is useful for downstream tasks. Additionally, generally in classification tasks one has to work with quite imbalanced data sets. Within natural language only a subset of sentences contain negations – thus negation annotated data might be prone to imbalance in such a way that there are many annotated sentences without any negations (positive sentences) versus sentences with negations (negative sentences). This paper looks at how this kind of imbalance affects neural model performance by comparing models trained on the full data set, with models trained on a subset in which the positive sentences have been filtered out. The results evaluated on the *SEM 2012 shared task on negation scope detection show that there does seem to be a difference in how the classifiers are affected by imbalance, depending on architecture; and how including part-of-speech (PoS) features helps to reduce this difference.

1 The Task: Introduction & Data

Within Negation Analysis the aim is to be able to automatically *detect cues* that lead to negation(s) within a sentence; *scope resolution* to relate the cue to its arguments; and *event identification* which is the key event of the scope. This can be useful in itself for example in triggering different services in web applications; but it can also be used as a pre-processing step for Sentiment Analysis (Lapponi et al., 2012). Negation Analysis can thus be separated into three sub-problems which can be solved in a sequential manner. In (1) the **cue** is shown in bold. It acts on the $\underline{\text{scope}}$ which has been underlined, and the key *event* of the scope is italicised (and by definition will also always be underlined).

(1) $\underline{\text{Some people}}$ **without** $\underline{\textit{possessing}}$ genius have a remarkable power of stimulating it.

Language can however be complicated, and there are many cases where cues can act on cues, and scopes be within scopes. The sentence in (2) shows one such example. Which is quite difficult to even grasp as a human reader, but showcases the complexity of language. The subscripts correspond to cues which act on the scopes in braces with the corresponding number as superscript. The events are also annotated with the subscript in front of the word. Here we see how a word can be both a cue, be within multiple scopes, and be an event as well.

(2) Mr. Sherlock Holmes, $\{\underline{\text{who }{}_{2}\textit{was}}\}^{2}$ usually $\{\underline{\text{very }{}_{2}\textit{late}\text{ in the mornings,}}\}^{2}$ **save**$_{2}$ $\{\underline{\text{upon }\{\text{those}\}^{0}\}^{1}\ \textbf{not}_{1}\ \{{}_{0,1}\textbf{\textit{in}}_{0}\{\textit{frequent}\text{ occasions when he was up all night}}\}^{0}\}^{1}\}^{2}$, was seated at the breakfast table.

The word *infrequent* in (2) displays all these properties, and we also see how the cue can be an affix of the word; e.g. the *in* in *infrequent*. Furthermore cue words can also be multi-token words, e.g. *no more* or *neither ... nor*. This complexity is difficult to encode in a reasonable way for a machine learning system to take advantage of, and thus we will make some welcome simplifications to adapt the data for neural methods.

As stated, Negation Analysis consists of three sub-tasks: (a) cue detection, (b) scope resolution and (c) event identification. The first simplification will be to only consider (b) given (a). The second simplification is to transform the data into a (sequential) binary classification task – which is to distribute the sentences with multiple cues (that have their own scopes) into separate sentences. We therefore get a copy of each sentence corresponding to how many cues it originally contains. To illustrate this, we can take (2) and transform it into (2a)-(2c).

(2a) Mr. Sherlock Holmes, who was usually very late in the mornings, save upon those not **in**frequent occasions when he was up all night, was seated at the breakfast table.

(2b) Mr. Sherlock Holmes, who was usually very late in the mornings, save upon those **not** infrequent occasions when he was up all night, was seated at the breakfast table.

(2c) Mr. Sherlock Holmes, who was usually very late in the mornings, **save** upon those not infrequent occasions when he was up all night, was seated at the breakfast table.

We have also removed the *events* in the latter three examples, because they are not part of our simplified variant of the task anyway. Lastly, since we are given (a), we will take the cue information and use it to solve the task. That means we will use the gold cue in the data set as features, together with the sentences (and later PoS tags).

To sum up, the task for a neural classifier will be to output a vector of binary values of the same length as the input sentence. For each position in the vector the classifier must predict whether the word is within a scope or not. Input to the system will be the sentences themselves, and additionally some binary value saying whether the word is (part of) a cue or not. The details of this are explained in section 3.

All this said, a model can not be built without data. The example sentences above have given a hint about what kind of data we are working with: the *SEM 2012 negation annotations over short stories by Sir Arthur Conan Doyle (Morante and Daelemans, 2012). Conveniently the data has been compiled into a training, development and evaluation (held-out) set which permits for reliable comparisons between systems. Table 1 gives an overview of the distribution.

The second column denotes the number of unique sentences in each data set. The third one is the number of sentences when we have distributed the sentences as in examples (2a)-(2c). While the last column is the number of sentences left when we filter out the positive sentences (i.e. sentences with no negations) after distributing sentences. Therefore the number of negative sentences comprise about 20-25% of the data, meaning we have quite an imbalanced data set. We will set out to explore how this affects the *uncertainty* of model performance in different settings. By uncertainty here we mean how the use of Distributed versus Negative data influences the final predictions of a neural classifier.

Data set	Unique	Distributed	Negative
Train	3643	3779	983
Development	787	816	173
Evaluation	1089	1118	264

Table 1: Distribution of the Conan Doyle data set. Unique is the number of unique sentences. Distributed is the number of sentences when they have been multiplied out as described in examples (2a)-(2c). The Negative column describes the number of sentences when we filter out positive sentences from the distributed set.

2 Related Work

Before describing the experimental setup we will reference other work on Negation Scope Resolution, as our latter results in this paper will be compared with these systems. The main inspiration for our work comes from Fancellu et al. (2016), where they use a bidirectional long-short term memory (BiLSTM) network to handle Scope Resolution. There they use word embeddings, and additionally create what they refer to as cue embeddings. Both of the embeddings types are high-dimensional representations (vectors) of words or cues, but in the latter case there will only be a few such vectors; and in theory only two because a cue is binary – however in code there will be an additional vector to represent a pad value. More on this in section 3.

Figure 1 shows this architecture. $\mathbf{E_w}$ and $\mathbf{E_c}$ denote the word and cue embeddings, respectively. These are concatenated together and fed sequentially into the BiLSTM from both directions. The softmax layer projects the high-dimensional output from the BiLSTM at each sequential step into a lower-dimensional probability distribution over negation or non-negation for each word in the sequence, and the final prediction for each word corresponds to the class with the largest softmax value at each sequence step.

Fancellu et al. (2016) show in their work that

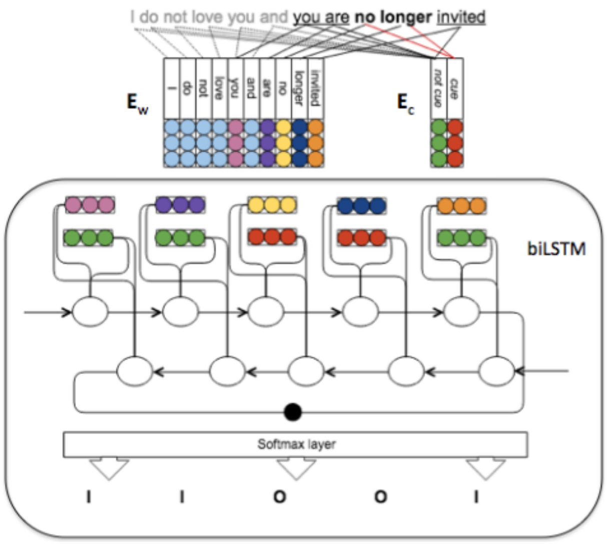

Figure 1: An example of scope detection using BiLSTM for the tokens ‘you are **no longer** invited’. Figure taken from Fancellu et al. (2016).

neural methods achieve results that are comparable to approaches which combine pre-neural methods, heuristics and upstream neural systems. In Lapponi et al. (2017) they use such an approach, by combining a pre-neural approach to sequence labelling with a state-of-the-art neural dependency parser and somewhat carefully engineered lexical and structural linguistic features. While Fancellu et al. (2016) report scope tokens (ST, tokens individually classified correctly to be within some scope) and scope match (SM, number of correctly classified full-scopes as defined by human gold standards) results, Lapponi et al. (2017) also report event tokens (ET, event identification) and full negation (FN). The latter work also gives a large overview of results using their system in combination with different parsers. All results are reported in F_1 scores evaluated on the *SEM metrics, and thus we will also report all results in F_1 scores as well.

3 Experimental Setup

As mentioned above we will do mostly the same as Fancellu et al. (2016), however we will describe some details which are left out in their paper, and other small differences. Firstly since we are given cue information and the cue itself is part of the sentence, in converting the output from the system to the *SEM format for further scoring, we do some post-processing to have the system always predict cues correctly. This is done by comparing with the gold standard by only looking at the input word and given cue information. If the cue and the input word are the same, the prediction is a non-negation (because cues are not negated themselves). However if the cue is only part of the input word (for instance an affix) then part of the input word counts as within the scope. If so we look at whether the input word starts with or ends with the given cue – and the rest of the word is predicted to be within the scope. There are also a few cases (although neither in the development nor evaluation sets) where cues are in the middle of a word, for example the word “help**less**ly”, and we have marked the scope and **cue**. Notice that the scope is considered to be in front of the cue; so in such cases we split the word on the cue and have the system predict the first part to be within scope.

Secondly, we experiment with using stacked BiLSTM networks. This means that each of the horizontal BiLSTM layers in figure 1 are fed into another BiLSTM layer, before going to a softmax layer. This is akin to using multiple hidden layers in a feed-forward neural network, and is considered a hyperparameter to tune. We also experiment with (and without) dropout regularisation (Hinton et al., 2012) which has been shown to possibly give improved results in NLP as well (Zhang and Wallace, 2015).

While they do not explicitly explain in Fancellu et al. (2016) how they create the cue embeddings from the given binary cue information, we will randomly initialise a cue embedding matrix and update it during training. This embedding matrix will in this case be of dimensionality $\mathbb{R}^{3\times d_c}$ where d_c is the dimensionality of each cue. In our case d_c ended up being 50 after doing a hyperparameter search over a few dimensionality values (same as (Fancellu et al., 2016), in other words). The number 3 in the dimensionality (as opposed to the expected number being 2 because of binary possibility) of the cue embedding comes from the fact that we also define a padding index for the cues, meaning that when using the cue embedding to look up the token in the input data there is a possibility of the “cue” being a padded value along with a padded “word” (useful for batching).

As for the word embeddings, we will experiment with using 300-dimensional pre-trained Global Vectors[1] (GloVe) trained on the Common

[1]Courtesy of Stanford NLP, downloaded from https://nlp.stanford.edu/projects/glove/

Crawl data set with 840B tokens. Additionally we will try 300-dimensional pre-trained fastText[2] that were also trained on Common Crawl but on a subset of 600B tokens. This differs from Fancellu et al. (2016) where they used 50-dimensional Google word2vec[3] embeddings in their experiments with pre-trained embeddings. This was also tried during hyperparameter search in our experiments, but never yielded considerably better results than with 300-dimensional embeddings and were thus left out in later experiments.

Similarly to Fancellu et al. (2016) we look at the effect of adding PoS / Universal Dependencies[4] PoS (UPoS will be used interchangeably with UD PoS) information as features together with the input text. In our setting they are initialised in the same way as the cue embeddings, and fine-tuned during training in the same manner. They have been chosen to have the same dimensionality as the cue vectors. Therefore the PoS embeddings are represented as $\mathbb{R}^{(p+1)\times d_{PoS}}$, where $d_{PoS} = 50$ and $(p + 1)$ is the vocabulary size of PoS tags (number of unique tags in the training set) plus a padding index. For our experiments p ended up being 15–16 (UD PoS) or 42–43 (PoS) depending on tag scheme and whether the data was filtered or not. PoS information is conveniently part of the *SEM 2012 input files, and we use a simple look-up table to map these to UD PoS tags for the experiments using these tags. UD PoS tags are useful to look at – since we work on an English data set, this is a way of showing how adding PoS information can generalise as aid for other languages.

Since our aim is to study the effect of imbalance in the data set, we of course experiment with training models using the full (distributed) data sets versus using the filtered data sets as described in table 1 (i.e. the Negative column). In contrast, this is something Fancellu et al. (2016) did not report doing experimentation with – instead they only use *filtered data*. To get meaningful results for our experiments we train 10 models for every combination of full and filtered data sets with both word embedding types, PoS and UD PoS tags. More details on this in section 4.

In designing experiments involving neural architectures, there are very many hyperparameters to tune and perform optimisation over. To reduce this search complexity, we have chosen to use the Adam optimiser (Kingma and Ba, 2014) and keep learning rate fixed at 0.001. We also use early stopping during training to prevent overfitting, and stop training when there is no improvement consecutively after 5 epochs. Batch size was fixed to 64. Other hyperparameters are the number of stacked BiLSTM layers and the hidden state dimensionality of the BiLSTM cells. After doing a coarse grid search over the remaining hyperparameters we ended up finding 2 or 3 stacked layers, and either 300 or 450 hidden state dimensionality to be optimal. With dropout probability we tested using a probability of 0 (i.e. no dropout) versus 0.5. Lastly, implementation was done using PyTorch[5], an open source deep learning framework for Python.

4 Empirical Results

We summarise the results from our experiments in table 2. Reported are the results (we remind that is the F_1 score) using the *SEM scorer, along with the metrics ST and SM on both the development (Dev) and evaluation (Eval) set, along with the standard deviations of the 10 runs. The scores in each column correspond to the *average* score from training 10 similar models (only the randomly initialised weights being different), and the number in parentheses is the standard deviation on the same metric. Systems trained on the non-filtered (NF) and filtered (F) data are grouped together somewhat in the table. All systems in this table used GloVe pre-trained embeddings. We report best scores from some of the runs in table 3, where we also include systems using fastText. Comments on this table is deferred to section 4.1.

Looking at the results in table 2 one can see that there is considerably higher variation in the scope match results than for the scope tokens. Since the task of getting a scope match is much stricter than for getting a single token correct, this is not surprising. However, looking at the scope tokens results it might seem like there is larger variation in the ST results for the models trained on the non-filtered data set than for the model using the filtered data set when no PoS features are used. On the other hand, when the model is provided with PoS / UD PoS information, the models trained on

[2]By Facebook Inc. downloaded from https://fasttext.cc/docs/en/english-vectors.html

[3]More information about this can be found at at https://code.google.com/archive/p/word2vec/

[4]Read more about this at https://universaldependencies.org/

[5]See https://pytorch.org/

System	ST-Dev	SM-Dev	ST-Eval	SM-Eval
NF-BiLSTM	83.34 (2.91)	70.68 (5.27)	87.50 (1.58)	74.39 (2.73)
NF-BiLSTM + Dropout	81.45 (3.20)	69.15 (5.83)	87.12 (1.97)	74.02 (3.18)
F-BiLSTM	81.70 (2.06)	66.61 (3.02)	86.60 (1.56)	70.73 (2.91)
F-BiLSTM + Dropout	80.99 (2.91)	66.67 (3.88)	86.27 (2.75)	70.81 (3.89)
NF-BiLSTM + PoS	86.78 (1.26)	75.51 (2.37)	88.91 (0.58)	76.89 (2.68)
NF-BiLSTM + PoS + Dropout	87.00 (0.79)	76.74 (1.69)	89.04 (0.67)	79.52 (0.85)
F-BiLSTM + PoS	85.95 (2.31)	74.43 (4.68)	88.66 (1.18)	76.77 (2.14)
F-BiLSTM + PoS + Dropout	85.21 (2.14)	73.50 (3.68)	87.93 (1.41)	77.02 (2.37)
NF-BiLSTM + UPoS	86.81 (1.37)	75.98 (3.33)	88.39 (1.29)	76.86 (3.67)
NF-BiLSTM + UPoS + Dropout	86.78 (1.72)	75.92 (2.54)	88.38 (1.37)	77.63 (2.61)
F-BiLSTM + UPoS	85.22 (2.02)	74.54 (3.34)	87.88 (1.86)	77.81 (2.16)
F-BiLSTM + UPoS + Dropout	86.30 (1.71)	75.22 (2.23)	88.56 (1.35)	77.11 (2.55)

Table 2: Main results from the experiments looking to analyse the effect of imbalance in the data set. Models trained on the full set denoted by NF (non-filtered), and the subset denoted by F (filtered). We report scope tokens (ST) and scope match (SM) on the development (-Dev) and evaluation (-Eval) sets. The score in each column is the average from 10 runs. Standard deviations based on the same 10 runs in parentheses.

the non-filtered data set seem to have somewhat lower standard deviation. Furthermore it seems like dropout in most cases leads to higher variability as well, which is also not surprising considering it introduces more stochasticity into the system. For ST then it looks like PoS / UD PoS tags help "guide" and stabilise the systems when it has a lot of "irrelevant" data – or rather, that "irrelevant" data becomes useful for the system since it gets the opportunity to learn patterns stemming from the combination of words and PoS tags. Finally by looking at the scores themselves it is clear that, on average, using the full data set seems to give a slight boost to overall performance. This is easily seen by comparing the scores of the two topmost rows with the two bottom rows within each group (i.e. the groups being not using PoS/UD PoS; using PoS; and using UD PoS), in which the topmost rows (NF-rows) have better scores.

System	Run	ST	SM
Packard et al. (2014)	-	88.20	78.70
FLW-BiLSTM+UPoS+E	-	88.72	77.77
Stanford-Paris	0	89.65	82.08
Stanford-Paris	6	89.11	**82.63**
F-BiSLTM+PoS	3	**90.22**	77.98
F-BiLSTM+UPoS+fT+D	4	89.38	80.86
F-BiLSTM+UPoS+fT+D	8	<u>89.99</u>	<u>82.55</u>

Table 3: Packard et al. (2014) is the best reported system using hand-crafted heuristics. FLW is the top-performing from Fancellu et al. (2016) where they used external embeddings (E), while Stanford-Paris results are reported in Lapponi et al. (2017). "fT" is fastText embeddings, and "D" stands for dropout. Bolded values are the currently reported best scores, and the underlined values are the next-best scores. All scores are calculated based on the held-out evaluation set.

4.1 More Results

We explained that we ran each experiment 10 times. Hyperparameter search was also performed at a preliminary stage. As such, there are many results not reported in this paper. We also tried with fastText embeddings to see if there were any noticeable differences, and while there were no significant changes, some of the runs gave decent results. In table 3 we list some more runs and compare with some of the (state-of-the-art) results reported in Lapponi et al. (2017) as well as the best results Fancellu et al. (2016) achieved using the same architecture but a different setup. Best result from Packard et al. (2014) is also shown, where they used hand-crafted heuristics together with reasoning techniques to parse the structure of the negative sentence.

The last two rows show a system using fastText embeddings instead, which is included because of its overall performance with ST and SM. Particularly interesting is that the use of Universal Dependencies PoS leads to such results, which might support the application of UPoS tags in similar problems for other languages. These systems are very much comparable to systems with heuristics built in (Packard et al., 2014), and display that neural systems are competitive when it comes to Scope Detection. The results also show the usefulness of PoS tags in guiding neural systems.

5 Other Considerations

During experimentation we also tried using pre-trained word embeddings that were trained on a PoS-tagged corpus. Specifially one downloaded from Nordic Language Processing Laboratory[6] (NLPL). Then the input to the system had to be modified to accommodate the vocabulary of the embeddings - which was trivial because PoS information is already in the *SEM data sets, so we only had to concatenate them with the input words. Surprisingly this did not yield good results – quite the opposite most systems struggled to get a score over 80 using such a pre-trained embeddings model. We believe that perhaps since the whole corpus the embeddings were trained on had to have been PoS-tagged, this has naturally changed the whole distribution of the language model in a way which might introduce bias; this could be dependent on the tagger that was used.

Another thing attempted was to combine a Convolutional Neural Network (CNN) with the BiLSTM in a meaningful manner. Wang et al. (2016) showed that using a CNN-LSTM architecture improved results within Sentiment Analysis. The idea is that CNN architectures might be good at encoding local features within a sentence, while a Recurrent Neural Network might excel at long-range dependencies. What we attempted was thus to run a CNN on the input, with different number of filters and kernel sizes, to create a sentence embedding. This was projected into the BiLSTM model's hidden state size through a dense layer (and using $\tanh$ activation) and fed into the BiLSTM as the initial hidden state in both directions. Although not reported here, such an architecture achieved about the same results as simply using a BiLSTM. The idea is interesting, and perhaps there are other data sets such an architecture works better on. In this case we chose Occam's Razor and decided not to report this earlier. We also note that this CNN-LSTM architecture displayed the same trend in variability when using full data versus filtered data.

[6]The embeddings with ID 1 at http://vectors.nlpl.eu/repository/

6 Conclusion and Outlook

With our work we have explored how the use of an imbalanced data set affects the uncertainty of a neural model's performance. We have seen that by using only word tokens as features the difference in uncertainty is quite small when looking at tokens correctly classified to be within a scope. On the other hand, if we are able to use PoS / UPoS features, *not* filtering the data set will lead to less uncertainty. Another important find is that not filtering the data seems to give an average increase in performance. Although we have not proven these results to be facts, the experiments seem to point in such a direction. We would therefore recommend future researchers to use the full data set, especially when provided with PoS information (though this of course comes with a computational cost), for scope resolution. We also encourage more research to be done using neural methods for Negation Analysis, seeing as they produce models with great performance without having to do much feature engineering.

For future work here there are several possibilities that might improve model performance. Perhaps using character-level convolutions over a word, together with the word, can help us. Alternatively, since the scope detection task is sequential in nature, it could be useful to try architectures with an inference layer that can exploit this – for example a BiLSTM-CRF architecture (Huang et al., 2015). Lastly, an interesting task for future work could be to look into multitask learning for Negation Analysis. It is not unthinkable that Negation Analysis can be improved through multitask and inductive learning, as it has shown to work both in general and for other NLP tasks as well (Caruana (1997), Luong et al. (2015), Dong et al. (2015)).

References

Rich Caruana. 1997. Multitask learning. *Machine learning*, 28(1):41–75.

Daxiang Dong, Hua Wu, Wei He, Dianhai Yu, and Haifeng Wang. 2015. Multi-task learning for multiple language translation. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint*

Conference on Natural Language Processing (Volume 1: Long Papers), volume 1, pages 1723–1732.

Federico Fancellu, Adam Lopez, and Bonnie Webber. 2016. Neural networks for negation scope detection. In *Proceedings of the 54th Meeting of the Association for Computational Linguistics*, page 495 – 504, Berlin, Germany.

Geoffrey E Hinton, Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever, and Ruslan R Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.

Zhiheng Huang, Wei Xu, and Kai Yu. 2015. Bidirectional lstm-crf models for sequence tagging. *arXiv preprint arXiv:1508.01991*.

Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

Emanuele Lapponi, Stephan Oepen, and Lilja Øvrelid. 2017. EPE 2017: The Sherlock negation resolution downstream application. In *Proceedings of the 2017 Shared Task on Extrinsic Parser Evaluation*, page 21 – 26, Pisa, Italy.

Emanuele Lapponi, Jonathon Read, and Lilja vrelid. 2012. Representing and resolving negation for sentiment analysis. In *Proceedings of the 2012 ICDM Workshop on Sentiment Elicitation from Natural Text for Information Retrieval and Extraction*, Brussels, Belgium.

Minh-Thang Luong, Quoc V Le, Ilya Sutskever, Oriol Vinyals, and Lukasz Kaiser. 2015. Multi-task sequence to sequence learning. *arXiv preprint arXiv:1511.06114*.

Roser Morante and Walter Daelemans. 2012. ConanDoyle-neg. Annotation of negation in Conan Doyle stories. In *Proceedings of the 8th International Conference on Language Resources and Evaluation*, page 1563 – 1568, Istanbul, Turkey.

Woodley Packard, Emily M. Bender, Jonathon Read, Stephan Oepen, and Rebecca Dridan. 2014. Simple negation scope resolution through deep parsing: A semantic solution to a semantic problem. In *Proceedings of the 52nd Meeting of the Association for Computational Linguistics*, page 69 – 78, Baltimore, MD, USA.

Xingyou Wang, Weijie Jiang, and Zhiyong Luo. 2016. Combination of convolutional and recurrent neural network for sentiment analysis of short texts. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 2428–2437.

Ye Zhang and Byron Wallace. 2015. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. *arXiv preprint arXiv:1510.03820*.