CoastalCPH at SemEval-2016 Task 11: The importance of designing your Neural Networks right

Joachim Bingel and **Natalie Schluter**
Centre for Language Technology
University of Copenhagen (Denmark)
{bingel,natschluter}@hum.ku.dk

Héctor Martínez Alonso
Alpage
INRIA & University Paris 7 (France)
hector.martinez-alonso@inria.fr

Abstract

We present two methods for the automatic detection of complex words in context as perceived by non-native English readers, for the SemEval 2016 Task 11 on *Complex Word Identification* (Paetzold and Specia, 2016). The submitted systems exploit the same set of features, but are highly disparate in (i) their learning algorithm and (ii) their angle on the learning objective, where especially the latter presents an effort to account for the sparsity of positive instances in the data as well as the large disparity between the distributions of positive instances in the training and test data.

We further present valuable insights that we gained during intensive and extensive post-task experiments. Those revealed that despite poor results in the task, our neural network approach is competitive with the systems achieving the best results. The central contribution of this paper is therefore a demonstration of the aptitude of deep neural networks for the task of identifying complex words.

1 Introduction

The identification of complex words plays an important role in the development of simplified reading resources. In particular, accurate automatic complex word identification strongly benefits lexical simplification (LS) as a first step in an LS pipeline (Paetzold and Specia, 2013; Shardlow, 2014). As such, accurate complex word identification may also be critical to higher-level tasks in text simplification, e.g. sentence compression, where the overall readability of a text is at least partly influenced by the difficulty of its vocabulary.

However, the perceived difficulty of words does not generalise across different groups of readers. That is, a system trained on manual annotations of word difficulty among second-language learners does not necessarily perform well in predicting words that are particularly difficult to persons of reduced literacy. The 2016 SemEval Task 11 puts its focus on the identification of complex words for second-language learners of English.

We chose to explore supervised methods for the task, employing a simple feed-forward neural network as well as a logistic regression classifier for two separate system submissions. Both systems take as input a single list of word and context features that we deemed to potentially indicate the complexity of words themselves and with respect to their context in the sentence.

In this system description paper, we focus specifically on our neural network model. As noted in the task description paper (Paetzold and Specia, 2016), systems based on neural networks generally performed rather weakly in the task, which the organisers speculate to be a consequence of the small amount of training data. However, we show in post-hoc experiments how careful network design may lead to performance figures close to those of the task-winning system. We make our revised system publicly available.[1]

[1] https://github.com/jbingel/cwi2016

Proceedings of SemEval-2016, pages 1028–1033,
San Diego, California, June 16-17, 2016.

2 Related work

The identification of complex words in context has in the past been embedded, often implicitly, into broader (lexical) simplification endeavours (Yatskar et al., 2010; Medero and Ostendorf, 2011; Horn et al., 2014). These models usually employ lexicons or corpus frequencies to determine candidates for lexical substitution. In a corpus study of the standard/Simple English Wikipedia, Medero and Ostendorf (2009) identify a number of word-level features that are indicative of texts with more difficult vocabulary. One of their findings is that words that are typical of simple texts tend to have longer definitions and more user-entered translations in Wiktionary. They also tend to be more ambiguous with respect to word class membership.

Research explicitly dedicated to complex word identification in context, however, has only appeared recently. Shardlow (2013a) presents experiments based on his complex word dataset mined from edit histories in Simple Wikipedia (Shardlow, 2013b); his classification system uses an SVM over a small number of features, achieving an F-score above 0.8 on the named dataset, where one word per sentence is a positive instance.

3 Task data

The data for the task was collected in a survey with 400 non-native speakers of English, such that it may in particular serve the identification of words that pose problems to language learners.

The organisers chose to have each of the 200 sentences in the training set annotated by 20 individuals, while in the test set each of the 9,000 sentences was only seen by one annotator. This presents an effort to model "how well one's individual vocabulary limitations can be predicted from the overall limitations of a group which they are part of" (Paetzold and Specia, 2016). The training data is released in two versions, once with the *individual* votings for each of the 20 annotators, and once with a *combined* vote that is defined as positive if at least annotator deems the word to be complex, negative otherwise. For both train training and the testing set, only a subset of the words in each sentence are annotated, resulting in 2,237 instances for training and 88,221 instances for testing, corresponding to roughly 40% and 39% of the overall number of tokens, respectively.

The challenge of the data. The task at hand poses two major challenges with regard to the data, *sparsity* of positive examples and *disparity* between train and test data. Regarding sparsity, combining the individual votings as described above results in roughly one third of the train data being marked positive. A one-to-two proportion is not too skewed, but if one considers each annotator in the training set separately, a mere 4.5% of the individual votings were positive.

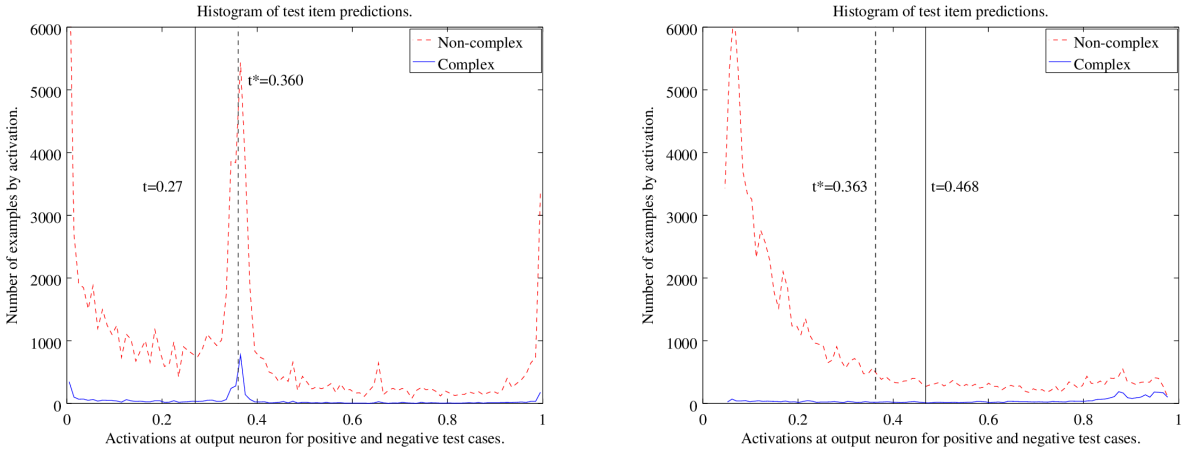
Regarding disparity, the share of complex examples is very different between the train and test instances (31.9% vs. 4.7%). Furthermore, the perception of word complexity shows great variation across annotators, with more than half of the complex examples in the training data (360 out of 716) marked as such by only one of the 20 annotators, and only 5.4% being perceived as difficult by the majority of the annotators.

4 Features

Both systems that we describe in Section 5 use the same set of features. We preprocessed the data using the Stanford NLP tools (Manning et al., 2014), obtaining for every sentence lemma forms, part-of-speech tags, name-entity types, and a dependency parse. We make use of Wikipedia and Simple Wikipedia for features based on corpus frequencies.

For a word w in a sentence s, we thus collect features from the following classes:

1. SIMPLE This feature group contains simple word properties such as length of w in characters, its named-entity type (if any) and its part-of-speech. Our feature model is delexicalised and we do not use word forms or lemmas as features.
2. POSITION This feature group contains the relative position of the w in s, and the number of commas as well as the number of verbs before or after w.
3. MORPH This feature group describes whether w has a Latin root, the length difference in characters between w and its lemma and stem, and the number of steps the Porter stemmer spends



(a) Submitted predictions

(b) Revised predictions

Figure 1: Histograms of submitted (left) and revised (right) Neural Network predictions. Solid vertical lines indicate the learned threshold at which we classify targets into simple and complex. The dashed vertical lines indicate the thresholds that would optimise the G-scores (cf. also Figure 3).

on w (Porter, 2001), which we use as a proxy for the number of inflectional morphemes that w has. We use the NLTK Porter for the latter and the Etymological WordNet (De Melo, 2014) for detecting Latin roots.

4. SYNTAX This class contains the distance (in tokens) between w and the root of the dependency tree, the dependency relation connecting w and its head, the distance to its head, and the number of dependents of w and its head.
5. PROB This group contains the probability of w in the English Wikipedia, in the Simple Wikipedia, and their ratio.
6. CHARCOMPLEXITY The feature group contains the character-level unigram and bigram probability of the word based on frequencies in Wikipedia and Simple Wikipedia, as well as the respective ratios. This is motivated by the observation that "words with simple grapheme-to-phoneme ratios [are] easier to learn than more phonetically complex words" (Dela Rosa and Eskenazi, 2011). Finally, this group includes the share of vowels in the word.
7. BROWN This feature group includes the Brown cluster of w, as well as height and depth of the cluster in the hierarchical clustering tree (Brown et al., 1992). We use the default 1000 clusters generated by Percy Liang's implementation.[2]
8. EMBS This feature group contains the 300-dimensional GloVe embeddings of w, calculated using Word2Vec over a Wikipedia dump.[3]
9. WORDNET This feature reflects the semantic complexity of w as measured by its number of WordNet synsets (Fellbaum, 1998).

5 Our systems

System 1: NeuralNet. We train a deep neural network with 2 hidden layers of 150 and 50 units, respectively, using PyCnn.[4] At every hidden layer, we perform $L2$-regularisation.

The network has a single output unit that yields a value between 0 and 1, which we map to binary complexity judgements after applying a threshold. If our neural network learned a perfect fit to the data, i.e. if it assigned a value close to 0 to every negative example and a value close to 1 to every positive

[2] `https://github.com/percyliang/brown-cluster`. We obtained the clusters from `http://derczynski.com/sheffield/brown-tuning/`.

[3] Downloaded from `http://nlp.stanford.edu/projects/glove/`.

[4] PyCnn is a Python wrapper for the C++ neural network library available at `https://github.com/clab/cnn`. The wrapper ships with the main library.

example, the natural decision boundary of 0.5 would be a good classification threshold to binarise the predicted values. However, as the model fit is far from perfect, the learning problem extends to finding an appropriate threshold (see below).

System 2: Information Sieve. Departing from the observation that different language learners consider different words as complex, the motivation for this system is to prevent our classifier from learning idiosyncratic annotator behaviour. To achieve this, we take an *information sieve* approach, which is meant to filter out idiosyncratic information. We train a logistic regression classifier based on the *concatenation* of all individual votings in the training set, such that every instance is seen 20 times during training, namely one per annotator. The intention behind this training method is to expose the learning algorithm to all individual annotator decisions, acquiring a preference for annotation decisions that are more consistent across annotators. We retrieve the probability outputs from the classifier and predict as positive the top decile.[5]

Hyperparameter optimisation All hyperparameter tuning for the neural network, including its architecture and the classification threshold, is based on 10-fold cross-validation over the training set. For each fold, we test the respectively trained model on each of the 20 individual votings in the test split, rather than on the combined votings. We do this in order to account for the mentioned disparity in the share of positive examples that we expect to find in the training and final testing data. Evaluating the performance of a single hyperparameter configuration thus involves training 10 models and testing on 200 sets of individual annotations.

For optimising the classification threshold, we compare the system's predictions to the respective train/test split annotations at every cross-validation fold (again testing on the individual votings of 20 annotators), and record the decision boundary that best separates negative and positive instances. For each of the 10 cross-validation folds, we thus get an array of 20 annotator-wise optimal thresholds. We finally compute the best threshold for a single fold as the median over these values, and the best overall threshold as the average over the per-fold medians.

[5]The decision to predict the top 10% as positive was based on our expectation of the positive share in the test data.

System	G-score	Recall	Acc.	Rank
NeuralNet	.506	.398	.693	35
InfoSieve	.285	.171	.869	41
SV000gg	.774	.769	.779	1
Revised NN	.756	.725	.791	(8)

Table 1: Test set results for the two submitted systems, the winning system and our revised system.

6 Results and System Revision

Table 1 shows the performance of our systems as measured by the G-score[6] as well as recall and accuracy. As suggested by a post-hoc experiment on the combined training examples, the poor result obtained by the InfoSieve system is due to its sampling strategy much more than the training and inference algorithm or any hyperparameters. We therefore discard this strategy and focus on the neural network in our following analysis.

In order to get a better understanding of the neural network's performance, Figure 1a visualises the distribution of the submitted predictions that we obtain in the original set-up. With a threshold at 0.27, we correctly discard only 43.8% of the non-complex examples, while identifying 69.3% of all complex instances. We observe that, relatively independent of the threshold, the model fails to clearly separate simple and complex instances.

With these numbers, the neural network is very far from the results achieved by the best systems in the task, which we generally observe to employ some variant of ensemble methods, most notably random forests. In the remainder of this section, we reconsider some design decisions for our neural network and introduce a revised model, which we demonstrate to obtain results on the test set that place it in the region of the mentioned ensemble systems.

Neural network library and architecture In a first post-task experiment, we exchange the PyCnn

[6]The task organisers define G as the harmonic mean between recall and accuracy. In using this metric to gauge system performance, the organisers reward systems that primarily maximise the number of true positives, while giving less weight to minimising false positives.

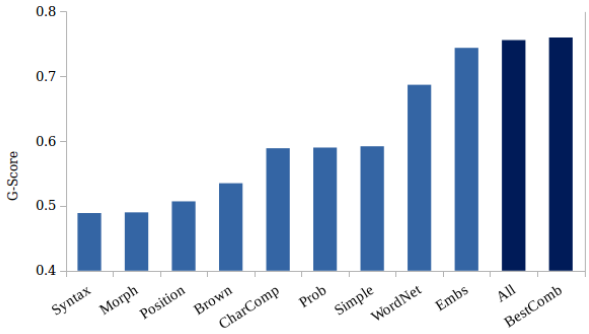


Figure 2: Contribution of feature classes in a deep neural network with three hidden layers of 50 computational units each.

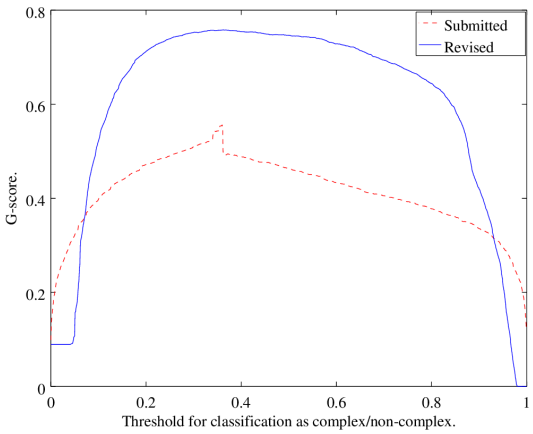


Figure 3: G-score curves in dependence of the threshold t for the submitted (red dashed) and the revised (blue solid) system.

implementation of deep NNs with Keras.[7] Surprisingly, we observe a tremendous increase in performance when using Keras: with the same set of features and the same basic network architecture, we reach a G-score that exceeds the 0.70 mark.[8] Subsequently, we evaluate different network architectures, including different rates of dropout after the hidden layers as an alternative way of regularising the model (Srivastava et al., 2014). Note that all hyperparameter optimisation is still performed in cross-validation experiments.

We finally find that we achieve a good fit to the data with a network of three hidden layers, each of which comprises 50 computational units, and a moderate dropout rate of 10% after every hidden layer.

Feature contribution To gain insights into which features are salient for our system, we train models on the individual feature classes listed in Section 4, as well as combinations of the classes. As Figure 2 shows, word embeddings are by far the most important single feature class, and they also contribute to the best combination of feature types, which is WORDNET+PROB+MORPH+EMBS. In the remainder of the paper, we refer to the system trained on this feature combination as the 'revised system.'

Threshold For the submitted as well as for the revised system, there is a certain discrepancy between the G-scores that we obtain from setting the thresholds to the value computed in the cross-validation and the figures we could achieve with optimal threshold setting. Concretely, this difference pertains to .05 in G-score (.506 vs. .556) for the submitted system and .002 (.756 vs. .758) for the revised system (cf. Figure 3).

While in Figure 1b it appears that we have missed the optimal threshold by a wide margin for the revised system, the actual difference in performance (as measured by G) is rather small. In fact, as Figure 3 illustrates, finding the optimal threshold is considerably less critical for the revised system due to a much flatter curve between threshold values 0.2 and 0.8.

Optimal result In conclusion, a G-score of .758 poses an upper bound for our system architecture (a neural network with three hidden layers of 50 units and a dropout rate of 10% after each hidden layer). How closely this figure can be approximated depends on the optimisation of the threshold.

[7] `https://github.com/fchollet/keras`. Keras is a Python neural networks library running on top of Theano and TensorFlow.

[8] The great performance differences between the two libraries are actually more than surprising. The differences are consistent across several experiments, which is why an infelicitous random initialisation or local optima can be ruled out as possible explanations. Other intuitions as to what might the reason for the different results touch on possible errors in the implementation of one of the libraries, but this is clearly very speculative.

7 Conclusion

This paper presented our submissions to the SemEval 2016 Shared Task 11, *Complex Word Identification*, which we approached with two very disparate systems: a deep neural network trained on the combined votings from 20 annotators, and a logistic regression classifier trained on each annotator separately. While neither of these systems reaches competitive performance figures, we re-design our neural network approach to train a revised system, which shows that neural networks do have the potential to successfully identify complex words.

Specifically, we find that 300-dimensional word embeddings carry a strong signal for word complexity, such that our revised system finally reaches a G-score that would place it in the top quartile of the task submissions. The strong contribution of word embeddings is not easily explained, but we suspect that certain distributional features may encode the presence of a complex word in more specific contexts such as technical jargon.

Our results contrast with the observation that task submissions which employed neural networks and/or word embeddings generally achieved rather poor results, and it shows that competitive performance figures can be reached with neural networks despite a relatively small amount of training data.

References

Peter F. Brown, Peter V. deSouza, Robert L. Mercer, Vincent J. Della Pietra, and Jenifer C. Lai. 1992. Class-based n-gram models of natural language. *Computational linguistics*, 18(4):467–479.

Gerard De Melo. 2014. Etymological wordnet: Tracing the history of words. In *LREC*, pages 1148–1154. Citeseer.

Kevin Dela Rosa and Maxine Eskenazi. 2011. Effect of Word Complexity on L2 Vocabulary Learning. In *Proc Workshop on Innovative Use of NLP for Building Educational Applications*, pages 76–80, Portland, OR, USA.

Christiane Fellbaum. 1998. *WordNet*. Wiley Online Library.

Colby Horn, Cathryn Manduca, and David Kauchak. 2014. Learning a lexical simplifier using wikipedia. In *ACL 2014*, pages 458–463.

Christopher D. Manning, Mihai Surdeanu, John Bauer, Jenny Finkel, Steven J. Bethard, and David McClosky. 2014. The Stanford CoreNLP natural language processing toolkit. In *Association for Computational Linguistics (ACL) System Demonstrations*, pages 55–60.

Julie Medero and Mari Ostendorf. 2009. Analysis of vocabulary difficulty using wiktionary. In *SLaTE*, pages 61–64.

Julie Medero and Mari Ostendorf. 2011. Identifying targets for syntactic simplification. In *SLaTE*, pages 69–72.

Gustavo H. Paetzold and Lucia Specia. 2013. Text simplification as tree transduction. In *Proceedings of the 9th Brazilian Symposium in Information and Human Language Technology*, pages 116–125.

Gustavo H. Paetzold and Lucia Specia. 2016. SemEval 2016 Task 11: Complex Word Identification. In *Proceedings of the 10th International Workshop on Semantic Evaluation (SemEval 2016)*.

Martin F. Porter. 2001. Snowball: A language for stemming algorithms.

Matthew Shardlow. 2013a. A Comparison of Techniques to Automatically Identify Complex Words. In *ACL 2013 Student Research Workshop*, pages 103–109. Citeseer.

Matthew Shardlow. 2013b. The CW Corpus: A New Resource for Evaluating the Identification of Complex Words. *ACL 2013*, page 69.

Matthew Shardlow. 2014. Out in the open: Finding and categorising errors in the lexical simplification pipeline. In *LREC*, pages 1583–1590.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.

Mark Yatskar, Bo Pang, Cristian Danescu-Niculescu-Mizil, and Lillian Lee. 2010. For the sake of simplicity: Unsupervised extraction of lexical simplifications from Wikipedia. In *Human Language Technologies: The 2010 Annual Conference of the North American Chapter of the Association for Computational Linguistics*, pages 365–368. Association for Computational Linguistics.