NLP Coursework Report

https://colab.research.google.com/drive/1M60JVaAmsqOSEXqa5ljS3G1wbtd0iFkC?usp=sharing

First Author
Nikolas Theodosiou
CID: 01894946
nt220@ic.ac.uk

Second Author
Ion Stagkos Efstathiadis
CID: 01074696
is1820@ic.ac.uk

Third Author
Tom Gotsman
CID: 01184964
tg220@ic.ac.uk

Abstract

For approach 1 where we used pre-trained representations of tokens, we attempted three approaches. One with simply the edited headline as input, a second with the edited headline with the edited out word concatenated at the end, and the last with only the edited out word and its replacement. We found the optimal results using the replaced/replacement word pair of an RMSE value of 0.552.

For approach 2, in one of our attempts we used our own trained representations, using Word2Vec. Here again, as in approach 1 we used three different types of inputs and found that once again the replaced/replacement work pair showed the best results on the validation set with an RMSE of 0.585. In our other approach 2 attempt we did not use any word embeddings at all. We explored how the parts of speech of each sentence affected the humour score. This approach, where we used linear regression, as we expected gave results very similar to the baseline.

1 Introduction

We chose to conduct the task of predicting a funniness score for an edited sentence. Through out this task we attempted three separate approaches. One using pre-trained word embeddings (part 1), one using our own personally trained word embeddings (part 2) and one with no word embeddings at all (part 2).

For part 1 we describe our tokenisation process where we had several revelations about the types of words that were commonly unknown, and we describe our approach to deal with unknown words. We follow by stating our hyperparameter search and evaluating our model on both a validation and test set. For part 2 we delve into our attempts to create and train our own word embeddings using Word2Vec, and follow by exploring approaches using no word embedding at all.

2 Approach 1

In this approach, the use of models and pre-trained representations of tokens was allowed. Therefore, pre-trained word embeddings known as Global Vectors for Word Representation (GloVe) were employed to represent the tokens comprising the headlines in the datasets. In addition to the provided testing and training datasets the ”funlines” dataset [1], constructed for the same purpose, was used for training. The general idea was to represent headlines as sequences of token embeddings and to feed them to a Bidirectional Long Short-Term Memory (BiLSTM) network to predict their funniness scores. The BiLSTM would then try to minimise the Root Mean Squared Error (RMSE) between its predicted scores and the corresponding average scores of edited headlines.

2.1 The Sequences Input to the BiLSTM

The most forthright approach is to feed the BiLSTM with the edited headlines: Sequences of embedded words reflecting the headlines after they have been edited to be made funny. Two additional inputs were tried in this project, both of which aimed at embodying the information of the word that was edited out in the sequences fed to the BiLSTM.

The first, perhaps less elegant input, consisted of the edited headlines with the edited out word added at their end. This approach has the disadvantage that the edited out word not only is not distinguished somehow from the rest of the sequence, but is also falsely presented to the BiLSTM, which takes the order of words into account, as the last word of the sequence. However, as the approach was very straightforward we still tried it.

The second approach was to represent headlines to be fed to the BiLSTM by sequences of just two words: the edited-out word and its replacement.

This reductive approach relies on the conjecture that most of the information required to predict the funniness of an edited headline is contained solely in the edited out word and its replacement. The question of whether it would be more appropriate to use some network type other than RNNs arose here but it was decided that it would still be beneficial to use a network that can perceive the sequential characteristics of its input so the BiLSTM was kept.

2.2 Tokenisation of Headlines

As explained, headlines were fed to the network as sequences of embedded tokens. In tokenising the headlines, it was assumed that tokens within each headline were separated either by a space or by a hyphen. Separation of tokens connected by a hyphen was introduced after it was noticed that a considerable proportion of tokens whose embeddings were not being found in GloVe were two well-known words connected by a hyphen. In tokenising the headlines, certain so-called stopwords were dropped as their contribution towards funniness was deemed insignificant.

To comprehend the effectiveness of the implemented tokenisation of headlines, the remaining tokens whose embedding was not found in GloVe were collected. In the case where the full edited headlines were to be fed to the BiLSTM, the unknowns in the training corpus were found to be 679 out of a total of 16787 words. This percentage of 4% was deemed satisfactorily small. A small sample of unknown words is printed here: {tonging, dicktator, twerking, birdfeeder, democrate, portriat, 200m, f*ck}. To further reduce the the number of unknown words, several techniques could have been used such as splitting compound words into their constituents, grouping curse words containing the asterisk together and dropping words containing digits.

To deal with the unknown words, it was decided to introduce an embedding to represent all of them. The possibility of learning their embeddings using the small corpus in hand and that of using sub-word unit conversion were rejected. Despite having obvious disadvantages, the way unknown words were handled here has the advantage that a large proportion of unknown words are strange, inappropriate and potentially funny words, so grouping them in one representation makes some sense.

2.3 BiLSTM Model and Training Loop

The BiLSTM model provided to the students was analysed and kept. It is composed of an embedding layer where the pre-trained embeddings were inserted, a single bidirectional LSTM layer and a fully connected output layer.

The provided training function was used with slight modifications. Notably, the loss criterion was MSE and the optimiser was Adam. After each epoch of training, the model was evaluated against validation data, extracted from the originally called training data, as instructed.

2.4 Hyperparameter search

The hyperparameters that were chosen to undergo tuning were the size of vector embeddings, the batch size, the number of epochs of training and the learning rate. A combination of babysitting and random search was employed for hyperparameter search. The size of embedding vectors was varied manually and model performance was recorded for each of the following sizes: 50, 100, 200 and 300. Random search was then performed for batch size in the range of [32, 1024], number of training epochs in the range of [4, 20], and learning rate in the range of [0.005, 0.0001]. Random search was chosen over grid search because it has been mathematically proven [2] to give a higher probability of approaching the optimum when multiple hyperparameters are involved. The sets of hyperparameters that were found to perform best for each model input approach described in section 2.1 are as follows:

edited headlines input:
$lr_{ideal} = 0.0008$
$batch\_size_{ideal} = 49$
$num\_epochs_{ideal} = 5$
$embedding\_size_{ideal} = 200$

edited headlines plus replaced word input:
$lr_{ideal} = 0.0006$
$batch\_size_{ideal} = 100$
$num\_epochs_{ideal} = 9$
$embedding\_size_{ideal} = 200$

replaced/replacement word pairs input:
$lr_{ideal} = 0.0018$
$batch\_size_{ideal} = 761$
$num\_epochs_{ideal} = 18$
$embedding\_size_{ideal} = 200$

Input Type RMSE on validation set
edited headlines 0.554
edited headlines plus replaced word 0.561
replaced/replacement word pair 0.552

Table 1: Performance on validation set of approach using pre-trained embeddings.

2.5 Performance on Validation Set Extracted from Training Data

Table 1 shows the performance of BiLSTM models trained on different types of inputs, using the tuned hyperparameters presented in section 2.4. The three types of inputs yield similar results, which are slightly better than a baseline RMSE value of 0.6. Interestingly, feeding the model with just the replaced word and its replacements marginally outperforms the other two types of inputs, although the difference in performance from the edited headlines input is insignificant. This means that the information contained in whole headlines is not valuable to the model in the way that it is presented to it.

The potential improvements to our approach are vast. Better pre-trained embeddings encapsulating the humourist sense of words could be used. Better tokenisation including lemmatisation and subword conversion could be implemented to deal with unknown words. A better way to feed the model with both the edited headline and the word that was edited out while conveying that it was edited out should be explored. Having observed that the current model responds well to information being abstracted away, (e.g. feeding just replaced/replacement words), the list of stop words could be increased to contain humouristically neutral words. Due to time limitations, the aforementioned ideas where not explored here.

2.6 Predictions from the test set

A model taking full edited headlines as input, pretrained using the tuned hyperparameters from section 2.3 was fed with the test data containing no mean funniness scores and produced funniness score predictions. These can be found in our code.

3 Approach 2

In this approach, the use of models and pretrained representations of tokens was not allowed. Therefore, to tackle this obstacle, we implemented two separate methods and compared the results. Namely, we first addressed the problem by training our custom word embeddings on the given corpus. The second and more creative approach, did not include word embeddings at all, and therefore our main task was to find ways other than vector representations to describe our corpus.

3.1 Word2Vec

In this subsection we analyse our first approach to the problem. As discussed above, we initially attempted to train our own word embedding, using the given corpus. After obtaining the custom vector representations, we would follow the same steps as in part one. Due to our relatively small corpus, we chose to implement the Word2Vec algorithm in order to vectorise our tokens. We have tried both implementing the algorithm from scratch, and using the `Gensim` python library. As the former was slow to train, we chose to proceed with the library.

The first issue we faced when implementing Word2Vec, was the fact that almost half of our vocabulary was represented by vectors of zeros. This is because the algorithm (by default), only embeds tokens that appear more than 5 times in the corpus. Intuitively, this makes sense, as there is not enough information to train reliable embeddings of words appearing fewer than 5 times. However, because our corpus is small, we decided lowering this bound from 5 words to 2 words. Further, we used the concatenated data set (Original + Funlines), as an input to the algorithm.

After training on the corpus, we experimented with the results, by plotting some of the representations and searching for similarities between words.

Naturally, the next step was to train our model using the custom embeddings. In order to directly compare results to pre-trained representations, We did not want to deviate from the implementation of approach 1. Therefore, using the custom embeddings, we constructed `word2idx`, `idx2word` dictionaries, `wvwecs` and `vectorised_seqs` and fed them to the BiLSTM.

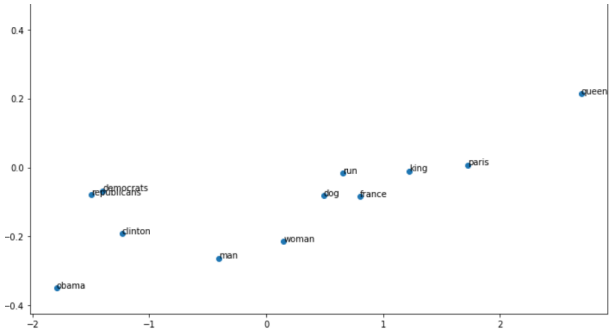

Figure 1: Visualisation of embeddings

Input Type RMSE on validation set
edited headlines 0.592
edited headlines plus replaced word 0.602
replaced/replacement word pair 0.585

Table 2: Performance on validation set of approach using embedding trained on the training corpus.

Comparison of 1st test funniness prediction, using edited headline as input: "Thousands of gay and bisexual swans convicted of long-abolished sexual offences are posthumously pardoned" : pre-trained word embeddings: 1.382, our-trained word embeddings: 1.004.

With our custom embeddings and model now trained, we wanted to further explore the syntactic structures in our dataset. After reading the paper: *Exploiting Syntactic Structures for Humor Recognition* [3], we wanted to test the incongruity theory on our data. This theory suggests, that sentences with pairwise dissimilar words tend to be funnier than consistent structures. In practice, this required calculating similarity metrics on the token pairs of our corpus, using our custom embeddings. Each sentence was given an average similarity score (total similarity over sentence length), to avoid longer sentences being more similar. It is worth noting that words that had no embeddings were given a similarity score 0. These averages were then plotted against the mean grade of each sentence. Surprisingly, even with a small corpus, there seems to be a weak correlation between similarity and humour. The plot can be found in the attached notebook.

3.2 No trained representations

After experimenting with word embeddings, we wanted to explore approaches with no word representations at all. We expected these approaches to have a lower accuracy, since a significant proportion of the semantic meaning would be discarded.

Our first, naive approach made use of POS tagging. Specifically, we wanted to explore how the parts of speech of each sentence affected humour score. In order to investigate this, we used the `nltk` library to tag each token (word) in our corpus. We then represented each sentence as its most common tag. This was achieved by creating a frequency dictionary for each sentence, and adding the most common tag of this dictionary to the corresponding column of our data frame. To train on the edited data set, the given linear regression approach was used. That is, we fed the corresponding input-output (`similarity score`, `meanGrade`) pairs to the regressor. As expected, this naive approach gave an accuracy similar to that of the baseline.

A second approach that we considered, which does not require word embeddings, was to manually construct a list of supposedly funny words, (e.g. Trump, titties) and to represent each edited headline by a score based on how many and potentially which funny words it contains. Then, a simple regression between our constructed scores and the mean funniness scores of headlines could be trained. Furthermore, the words included in the constructed list, as well as their embeddings could be themselves learnable. This approach was not implemented for lack of time.

4 Conclusion

Overall, performance is lower when using custom embeddings or no embeddings at all. This behavior is expected, since the corpus is very small and therefore it fails to capture semantic representations accurately. With a better embedding, the exact same pipeline as in part 1 could be followed. Perhaps, a distance metric between sentences could be defined, with the training of the model based on how similar each sentence is to the 5 funniest ones.

References

[1] Nabil Hossain, John Krumm, Tanvir Sajed, and Henry Kautz. Stimulating creativity with funlines: A case study of humor generation in headlines. *arXiv preprint arXiv:2002.02031*, 2020.

[2] Y. Bengio J. Bergstra. Random search for hyper-parameter optimization. *Journal of Machine Learning Research*, 2012.

[3] Lizhen Liu, Donghai Zhang, and Wei Song. Exploiting syntactic structures for humor recognition. pages 1875–1883, 2018.