Pun Detection and Interpretation: a Recurrent Neural Network with Long Short-Term Memory cells approach

Dung Le
Bennington College / Bennington, VT 05201
dungle@bennington.edu

Abstract

Pun is one of many instances of natural language that poses certain difficulty for artificial intelligence system to comprehend. It addresses a compelling question of whether or not a computer system can interpret the connotation of a word given its context. This research aimed to tackle the task of pun detection using Recurrent Neural Network (RNN) with Long Short-Term Memory (LSTM) cells. In addition, it also presented a knowledge-based method for locating and interpreting pun word using vector representation of sense and the lexical database WordNet.

1 Introduction

From Shakespeare's literature to daily conversation, pun - a form of wordplay that exploits the polysemy of a word (homographic pun) or the fact that there are words that sound alike but have different meanings (heterographic pun) - has been used intensively to elicit humor and laughter. Pun, in its pure nature, has a subtlety that makes it inherently difficult even for human to comprehend. The two types of pun are expressed in the following sentences:

1. I used to be a banker but I lost interest
2. The doctor has a bad day at the orifice

In the first instance, the word 'interest' can be understood in two ways: i. the state of wanting to know or learn about something, or ii. the money paid regularly at a particular rate. In the latter example, the word 'orifice' sounds similar to the word 'office', and therefore can be punned as heteronyms.

Recurrent Neural Network, on the other hand, has started to gain popularity in the recent years, especially in the tasks concerning natural language and speech processing. This is due to the model's ability to preserve the sequential characteristic of words in a sentence. This paper aims to tackle the task of pun detection using Recurrent Neural Network with Long-Short Term Memory cells. In addition, the paper also presents a knowledge-based method for locating and interpreting pun word using vector representation of sense and the lexical database WordNet.

The subtasks presented in this paper were first introduced by Miller et al. (2017) in the SemEval-2017 Task 7: Detection and Interpretation of English Puns.

Subtask 1: Pun detection. Given a document of sentences that either contain a pun word or not, classify sentences with pun word from those without.

Subtask 2: Pun location. For each context that has pun word, locate the pun word.

Subtask 3: Pun interpretation. For each context that contains pun, the system needs to annotate the two different senses being used in WordNet.

There are two important assumptions in all of these tasks: i, for each sentence that contains pun word, there is exactly one word intended for pun, and ii, the three tasks are independent from each other.

The data set is made available to the research community by Miller et al. (2017), and had already been used to evaluate systems participated in SemEval-2017 Task 7. It consists of 2250 contexts, of which 1607 contain a single homographic pun, and a further 1780 English one-liners, of which 1271 contain a single heterographic pun. Each pun word is annotated with their WordNet 3.1 senses (Princeton University, 2010). This research only presents the three subtasks in relation to homographic pun.

2 Related Works

My work is inspired by the use of distributed word embeddings as input features for pun detection as proposed by Indurthi and Reddy (2017). They trained a bi-directional recurrent neural network model by using the pre-trained 50 dimensional GloVe embeddings (Pennington et al., 2014). The embeddings were pre-trained on about 6B words from Twitter using the Continuous Bag of Words architecture. My approach is analogous in the way that it also leverages on the use of distributed word embeddings which capture the lexical and semantic features of each word in the sentence. However, the differences are that in lieu of pre-training the words using the Continuous Bag of Words architecture, my word2vec embeddings utilize the Skip-gram model with negative sampling, which works better with small amount of training data and represents well even for rare words (Mikolov et al., 2013b).

Several other related systems were built based on different observations of English pun. However, none of them used word embeddings as features to train their models. JU-SCE-NLP (Das and Pramanick, 2017) used syntactic features from a part-of-speech tagger and a syntactic parser to train their hidden Markov Model. PunFields (Mikhalkova and Karyakin, 2017) represented each word as a vector of frequency that counted the number word in the same context appeared in *Roget's Thesaurus*. Idiom Savant (Doogan et al., 2017) and UWAV (Vadehra, 2017) did train their classifiers on lexical-semantic and word embedding features. However, instead of training on a deep neural network model, UWAV model took votes from three classifiers (support vector machine, naive Bayes and logistic regression) while Idiom Savant detected pun by calculating the maximum score from all possible n-gram pairs and classified as pun if the score exceeded a certain threshold.

3 Subtask 1 - Pun Detection

In this subtask, given a context, the system needs to classify sentences with pun from those without. Even though the task was designed to make use of unsupervised method, my system - LSTM_Pun - utilized supervised method using word embedding and a deep neural network as binary classifier. For word embedding, in lieu of implementing the word2vec algorithm from scratch, a Google Pre-trained word2vec model, which had its data trained on 3 million words and phrases with 300 hidden layers in the neural network, was used. This decision was made considering the fact that word2vec algorithm generalized well when being trained on a large data set (Mikolov et al., 2013a). This decision, however, had a drawback that I would discuss later when evaluating the results of subtask 1. For the binary classifier, Recurrent Neural Network was chosen due to its ability to capture the sequential characteristic of natural language. In addition, since the pun word is semantically related with at least one previous word in the sentence, Long Short-Term Memory cells were integrated as a way to encapsulate the long-term dependency of words in sentence.

The data set was split into train and test set using two settings. In Setting 1, the homographic pun data set was divided into a training set, which contained 1800 (80%) contexts, and a test set with 450 contexts. The sizes of training set and test set in Setting 2 were reversed, with the former having 450 (20%) contexts and the latter containing 80% of the total contexts. Divided this way, the system was thoroughly examined under both conditions where the training set is scarce and prolific.

3.1 RNN with LSTM cells Model

The model contains three layers as presented below:

- *Embedding Layer*: The sentence was first tokenized and lowercased. Words that do not appear in the Google Pre-trained vocabulary were removed since every word needed to be represented as sense vector. These sense vectors were then used as inputs to the hidden layer.

- *Hidden Layer*: Each sense vector was passed into a neural network cell with four hidden layers, in which information was decided to be thrown away, to get store in the current cell state, and to output (Olah, 2015). The first decision was made by a sigmoid layer called the "forget gate layer." It outputted a 1 as to "keep" or 0 as to "get rid of" the information. The next two layers, a sigmoid followed by a tanh function, were in charge of determining what new information should be stored in the current cell state. Finally, the last layer decided what to output. This sequence was

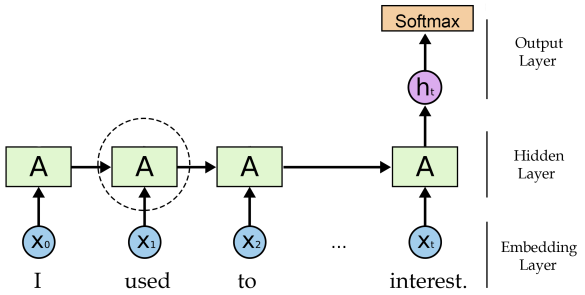

Figure 1: RNN with LSTM cells.

repeated until the last word of the sentence, with the output of the previous word being the input of the next word.

- *Ouput Layer*: The final output was passed into a softmax function, which returned a 2-dimensional vector representing the probabilities of the sentence containing pun or not.

3.2 Evaluations and Results

Evaluation metrics include precision, recall, accuracy and F1 score as used in regular classification task.

$$P = \frac{TP}{TP + FP}$$
$$R = \frac{TP}{TP + FN}$$
$$A = \frac{TP + TN}{TP + TN + FP + FN}$$
$$F_1 = \frac{2PR}{P + R}$$

with TP, TN, FP, and FN being *true positive*, *true negative*, *false positive*, and *false negative*, respectively.

In addition to the random baseline, which randomly selected 0 as non-pun or 1 as pun for each sentence, I also selected Fermi system (Indurthi and Reddy, 2017) to compare the results against since this system used the same approach as my system does. The results can be found in Table 1.

3.3 Discussion

In both settings, LSTM_Pun yielded a higher recall than the random baseline and Fermi system, which meant that among all the contexts that were known to have pun, more than 98% was correctly predicted. However, the trade-off between precision and recall was visible in both settings of LSTM_Pun and Fermi. The results of the two systems suggested that bi-directional RNN preserved high precision while RNN with LSTM cells emphasized on high recall. However, a high recall - low precision result also meant that the model was biased toward the classification of pun over non-pun.

Except for precision, all three other measurements of LSTM_Pun Setting 2 were significantly higher than that of Fermi Setting 2, meaning that the system generalized well even under the scarcity of the training set. This could be rationalized by the increase in the size of hidden layers (from 150 layers in Setting 1 to 300 layers in Setting 2.) However, as mentioned earlier, the use of Google Pre-trained word2vec model also had drawback, in which the vector representation of sense might not capture both senses of the pun word equally (since some senses were more popular than others depending on the pre-trained data.)

4 Subtask 2 - Pun Location

Subtask 2 asked participated systems to locate the one word intended for pun. The method discussed below was built upon these following observations of English pun.

- One sense of the intended pun word would be semantically related to at least one previous word in the sentence. For example, in (1) the pun word "interest" has one sense that is related to money and "banker" in general.
- Stopwords and words with singular meaning (if any) cannot be pun word
- Pun word is more inclined to appear toward the end of the sentence (as a way to elicit surprise or humor.)

With that being said, I computed the cosine similarity between every single pair of words using their vector representation of sense. The top three highest scores were chosen. Weights were then applied to each word within these pairs. After this step, a pair with the highest weighted similarity score was filtered out, and word that was closer to the end of the sentence was then selected as pun word.

4.1 Evaluations and Results

The evaluation metrics for subtask 2 are similar to the first one in that both used precision, recall and

Systems	Precision (P)	Recall (R)	Accuracy (A)	F_1 score
Random	0.7142	0.5000	0.5000	0.5882
Fermi Setting 1	**0.9697**	0.7953	**0.8360**	**0.8738**
Fermi Setting 2	0.8918	0.6876	0.7173	0.7765
LSTM_Pun Setting 1	0.7320	**0.9969**	0.7333	**0.8442**
LSTM_Pun Setting 2	**0.7384**	0.9836	**0.7411**	0.8425

Table 1: Subtask 1 - Pun Detection results.

F_1 score to measure the correctness of the method. However, while subtask 1 assumed that all systems yielded a 100 percent coverage, the coverage of a system in subtask 2 was the target of investigation since this result varied based on different approach.

$$Coverage(C) = \frac{num\ of\ guesses}{num\ of\ contexts}$$

The three baselines for subtask 2 were random, last word, and max polysemy. The random baseline impulsively selected one word in the sentence as pun. Meanwhile, the last word baseline always assigned the last word as pun. The third baseline was slightly more complicated as it was inspired by Mihalcea et al. (2010). In that study, pun word could be detected by finding the candidate whose words have the highest mean polysemy. As a result, the third baseline classified word with the highest amount of senses in WordNet as pun. The results can be observed in Table 2.

4.2 Discussion

All measurements of LSTM_Pun system return better results in comparison to the three baselines. The fact that the system's scores were not significantly different from that of the last word baseline reinforced the observation that pun word was inclined to appear toward the end of a sentence. However, it is worth noting that the actual cosine similarities measured between each pair of word in the sentence did not significantly differ from each other. This again could be explained that a 300 dimensional vector might contain too many noises, and thus, failed to represent the meaning of a word concisely. The system can be improved by adapting Latent Dirichlet Allocation (LDA) method, which can be used to cluster words from the same topics.

5 Subtask 3 - Pun Interpretation

In this subtask, the system needs to disambiguate the two senses intended by the pun word given its context and location. The contexts in this subtask was selected so that only pun words with both senses being annotated in WordNet 3.1 were tested. The two senses of pun word were achieved using the following methods:

- Each sentence was first passed into a Part-Of-Speech (POS) tagger, which returned the corresponding POS for each word. Knowing the POS of the pun word, I searched for the most common sense of this word given its POS. If there existed no sense for this POS, I tried to stem the word, assigned a new POS, and looked up the word again. This approach yielded the *first sense* of pun word.

- The *second sense* was achieved using a variation of Lesk algorithm, which has been used intensively in the task of word sense disambiguation (Jurafsky and Martin, 2008). Given the pun word and its sense vector, I computed the cosine similarities between this word and every other words in the sentence. The pair with highest similarity score contained the pun word and another word that was semantically related to this pun word. The latter would be used to search against the gloss of each sense of the pun word.

- However, since this approach yielded a significantly low coverage (7.68% out of 1298 contexts), the fallback used a simplified version of Lesk algorithm, which chose a context window and searched for every word within this window in WordNet.

- Only pun words with both senses obtained from the aforementioned methods were counted toward the final results.

5.1 Evaluations and Results

Analogous to those in subtask 2, subtask 3 measured coverage, precision, recall and F_1 score to

Systems	Coverage (C)	Precision (P)	Recall (R)	F_1 score
Random	1.0000	0.0946	0.0946	0.0946
Last word	1.0000	0.4400	0.4400	0.4400
Max polysemy	1.0000	0.2427	0.2427	0.2427
LSTM_Pun	1.0000	**0.4698**	**0.4698**	**0.4698**

Table 2: Subtask 2 - Pun Location results.

Systems	Coverage (C)	Precision (P)	Recall (R)	F_1 score
Random	**1.0000**	0.0931	0.0931	0.0931
Most frequent sense	**1.0000**	**0.1348**	**0.1348**	**0.1348**
LSTM_Pun	0.8629	0.0464	0.0401	0.0430

Table 3: Subtask 3 - Pun Interpretation results.

determine the efficiency of the algorithm. In addition to the random baseline, subtask 3 introduced the most frequent sense baseline, in which the two most frequently used senses of the word would be chosen. The results of these baselines and method are represented in Table 3.

5.2 Discussion

LSTM_Pun system did not performed well in this subtask. The reasons might arrive from both internal and external factors. The external factors varied from the fact that WordNet synsets posed a problem of consistency to the non-transcribed method of mapping from manually annotated sense to WordNet lemma. However, this discussion would reflect on the system as the dominant factor of the poor performance. For the *first sense*, the Part-Of-Speech tagger can be improved to correctly tag sentences written in present perfect tense. More importantly, however, the Lesk algorithm used for the *second sense* must be modified so that it can cover a large gloss of not only the pun word but also its synonyms.

6 Conclusions

In conclusion, recurrent neural network is not only a plausible but also the current state-of-the-art approach to many natural language processing tasks - pun detection included. This paper presented a variation of recurrent neural network in which a memory cell was integrated within each layer to decide which information to preserve and which to discard. The results were significant and comparable to the current state-of-the-art method (Indurthi and Reddy, 2017), which relied on bi-directional recurrent neural network to detect pun.

Regarding the question of whether or not an artificial intelligence system is capable of understanding the connotation of a word given its context, the results from subtask 2 and subtask 3 shed light on the possibilities, yet, insinuated certain difficulties. The paper suggested that the cosine similarity between two sense vectors was not adequate to measure semantic resemblance between two words.

As future work I plan to adapt Latent Dirichlet Allocation (LDA) for topic classification and Latent Relational Analysis (LRA) for semantic similarity measurement.

References

Dipankar Das and Aniket Pramanick. 2017. Ju_cse_nlp at semeval 2017 task 7: Employing rules to detect and interpret english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*. Association for Computational Linguistics, pages 432–435.

Samuel Doogan, Aniruddha Ghosh, Hanyang Chen, and Tony Veale. 2017. Idiom savant at semeval-2017 task 7: Detection and interpretation of english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*. Association for Computational Linguistics, pages 103–108. http://www.aclweb.org/anthology/S17-2011.

Vijayasaradhi Indurthi and Oota Subba Reddy. 2017. Fermi at semeval-2017 task 7: Detection and interpretation of homographic puns in english language. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*. Association for Computational Linguistics, pages 457–460. http://aclweb.org/anthology/S17-2079.

Daniel Jurafsky and James H. Martin. 2008. *Speech and Language Processing: An Introduction to Nat-*

ural Language Processing, Computational Linguistics, and Speech Recognition. Prentice Hall, Upper Saddle River, NJ.

Rada Mihalcea, Carlo Strapparava, and Stephen Pulman. 2010. Computational models for incongruity detection in humour. *Computational Linguistics and Intelligent Text Processing: 11th International Conference, CICLing 2010* pages 364–374. https://doi.org/10.1007/978-3-642-12116-6_30.

Elena Mikhalkova and Yuri Karyakin. 2017. Punfields at semeval-2017 task 7: Employing rogets thesaurus in automatic pun recognition and interpretation. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*. Association for Computational Linguistics, pages 426–431.

Tomas Mikolov, Kai Chen, Greg S Corrado, and Jeffrey Dean. 2013a. Efficient estimation of word representations in vector space. In *Proceedings of Workshop at ICLR*.

Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeffrey Dean. 2013b. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*. pages 3111–3119.

Tristan Miller, Christian F. Hempelmann, and Iryna Gurevych. 2017. Semeval-2017 task 7: Detection and interpretation of english puns. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*. Association for Computational Linguistics, pages 58–68. http://aclweb.org/anthology/S17-2005.

Christopher Olah. 2015. Understanding lstm networks. *Colah's Blog* .

Jeffrey Pennington, Richard Socher, and Christopher D. Manning. 2014. Glove: Global vectors for word representation. In *Empirical Methods in Natural Language Processing (EMNLP)*. pages 1532–1543. http://www.aclweb.org/anthology/D14-1162.

Princeton University. 2010. Wordnet. *Princeton University "About WordNet"* .

Ankit Vadehra. 2017. Uwav at semeval-2017 task 7: Automated feature-based system for locating puns. In *Proceedings of the 11th International Workshop on Semantic Evaluations (SemEval-2017)*. Association for Computational Linguistics, pages 449–452. http://www.aclweb.org/anthology/S17-2077.