1003376438 Yan (Tina) He
1002907090 Yu Shen (Lucy) Lu
Submission date: Dec 02, 2018
Word count: 1988 words
Penalty: 0%

MIE324 Final Report – SPINN

Simple Pun Investigation Neural Net

Introduction

This document describes the design and implementation of the Simple Pun Investigation Neural Net (SPINN) to detect and locate English homographic puns. Subtle wordplay is difficult for a computer to process; thus, we use a divide and conquer strategy to break down this problem into two subtasks: pun detection and pun location. SPINN utilizes the traditional neural net models as well as SenseGram [1], a word sense disambiguation (WSD) tool that evaluates the likelihood of a given word having a certain meaning based on the context.

For language learners, puns are one of the hardest concepts to grasp. One would need to understand multiple meanings of a word, in order to realize that it is being used in an intentionally ambiguous and humorous way. Puns demonstrate how a word can be used in different contexts, so studying puns is a useful way to learn new vocabulary. For people who are learning English as a second language, puns can cause confusion and lead them to misunderstand sentences.

There are mainly two types of puns in the English language, homographic and homophonic. Homographic puns involve a play on meaning, and homophonic puns involve a play on pronunciation. The latter is much more difficult to detect, because homophonic puns often rely on approximate pronunciation and irregular spelling. Therefore, SPINN will focus solely on homographic puns, which involve the same word being used in two contexts. An example of a homographic pun is: “I used to be a banker, but I lost **interest**.” In this example, the word “interest” is the pun and can be interpreted as both “dividends” and “enthusiasm”. We would like SPINN to check if a sentence contains a homographic pun and locate the punning word so that the user can find the meanings behind the pun.

SPINN can have many potential applications. For instance, it can be used to create more sophisticated AI chatbots, by enabling the extraction of multiple contexts of words. In addition, SPINN can be further developed in to an entertaining pun generator.

Description of overall software structure

Figure 1 below demonstrates the implementation of subtasks 1 and 2. Through the user interface, the user can input a sentence and obtain prediction results for pun detection and location using various models, as shown in Figure 2. A link is provided so that the user can look up the meaning of the likely punning word in the input sentence.

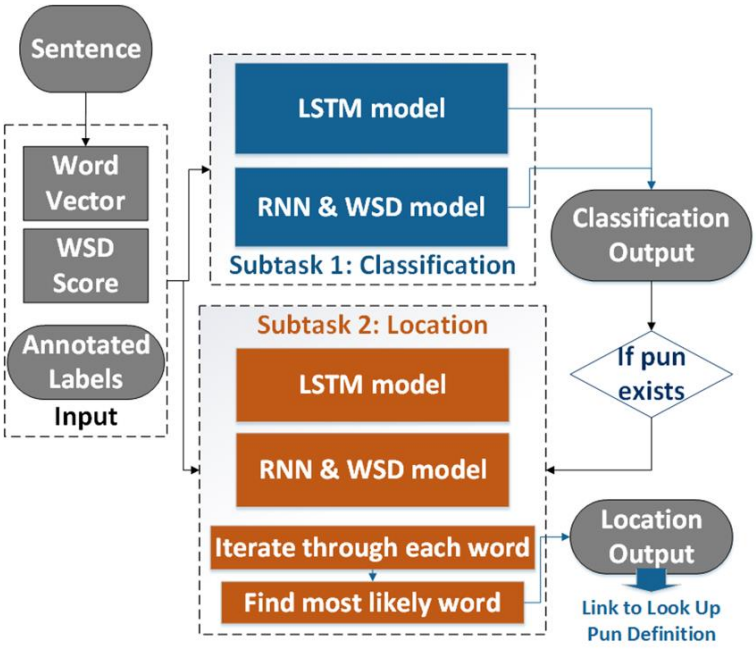

Figure 1: Block diagram of overall software structure

```
Enter a sentence:
 We dressed up as almonds for Halloween . People say we ' re nuts !
RNN model thinks the sentence is not pun, with 99.5% confidence
WSD+RNN model thinks the sentence is pun with 56.6% confidence
Rnn model thinks the punning word is "nuts".
Sensegram_rnn model thinks the punning word is "nuts".

To look up the definition of nuts, you can go to: https://www.google.ca/search?q=nuts%20definition
```

Figure 2: Screenshot of user interface

Sources of data

Most of the training data is sourced from the SemEval-2017 Task 7 dataset [2], which contains English homographic puns. There are 2250 contexts, and 1607 contexts (71%) contain puns. Each context contains a maximum of one pun, which is restricted to a single word. Multi-word expressions are not considered. The dataset includes "punning and non-punning jokes, aphorisms, and other short, self-contained contexts sourced from professional humorists and online collections" [3].

Approximately 20% of the puns in the dataset are created from templates and contain the same keywords or phrases, as shown in Table 1 below. In order to diversify the dataset, we manually altered sentences and eliminated repetition. To create a balanced dataset, we added more puns

and non-punning jokes from the Reddit clean jokes page and the Kaggle Short Jokes dataset. After processing, our dataset contains 2510 contexts, half of which contain puns.

Table 1: Puns with repeated keywords in the SemEval dataset

Keywords	Count
"My name is xxx"	55
"OLD xxx never die"	178
"She was only a xxx' s daughter"	47
"Tom"	57

Machine learning models and performance

Subtask 1: Pun detection

(1) Initial Recurrent Neural Net (RNN) and Convolutional Neural Net (CNN) models:

The RNN utilizes Long Short-Term Memory (LSTM) cells rather than Gated Recurrent Unit (GRU) cells to deal with long sentences. We used pretrained GloVe word embeddings of dimension 300. The model has an embedding layer that converts a tokenized sentence into an array of word vectors, which is then passed into the LSTM with hidden size 100. The output is passed through two linear layers, and the final sigmoid activation function outputs a single probability.

The CNN model has 1 convolution layer that uses 100 kernels of size 2 and 4 to scan each sentence; the output is passed to a linear layer which then calculates a single probability. The motivation for this is to let CNN examine word combinations, since in our dataset there are a lot of puns that involve idioms.

As shown in Figure 3, after a few epochs the validation and training accuracy start to diverge because the model is overfitting. The confusion matrix in Figure 4 shows that the initial RNN model is not biased towards either class (pun or not pun).

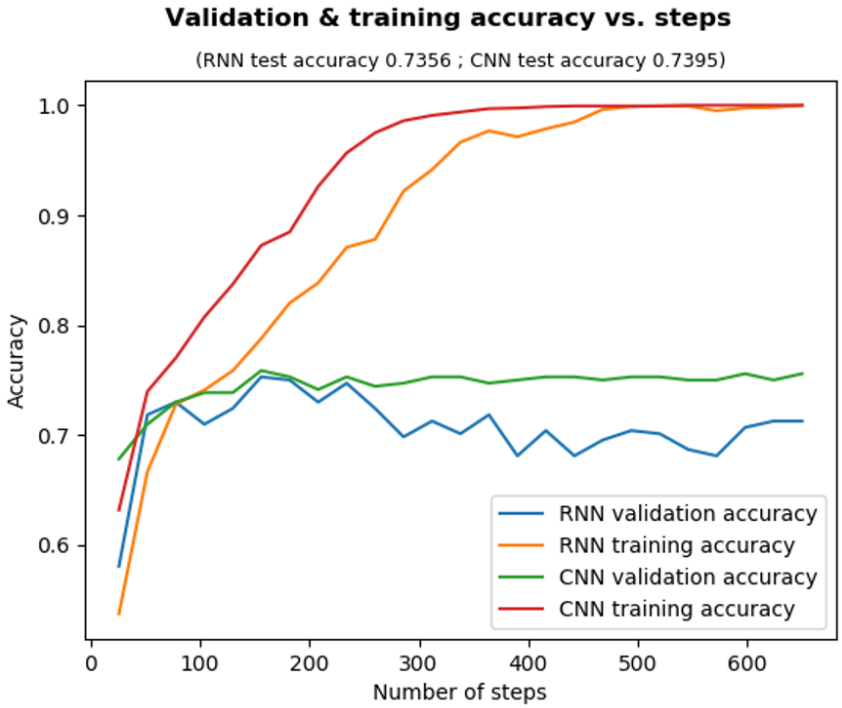

Figure 3: training, validation and test set results for subtask 1 initial models

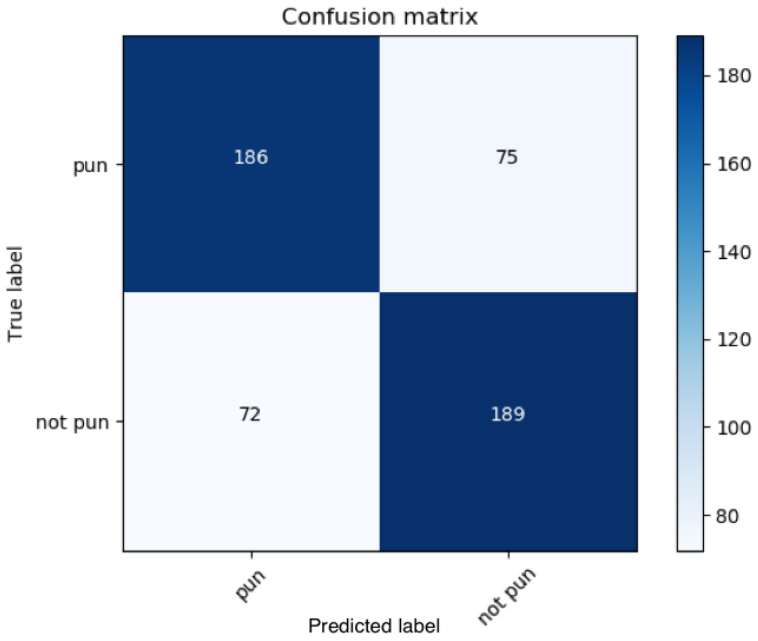

Figure 4: Confusion matrix for subtask 1 RNN model[1]

[1] Note: Only the RNN model's confusion matrix is shown here. In fact, all RNN/CNN/CRNN/etc. models in subtask one perform similarly, and their confusion matrices all look similar to this one shown in the report.

(2) RNN+WSD model

We experimented with drop-out and bidirectional RNN with various parameters; however, none of the attempts mitigated the overfitting of the model. We realized that the model is trying to memorize high frequency punning words. Attempts to balance the word frequency is futile since it is almost impossible to have all words appear equally frequently in both puns and non-puns. Based on Miller's hypothesis [4] on pun identification, WSD can help identify puns. WSD techniques are used to identify the meaning of a word in a specific sentence, when the word itself has multiple meanings. If WSD yields an ambiguous prediction for a word, it may mean that this word is associated with more than one meaning in the given sentence, which fits the description of homographic puns.

We used the open-source toolkit SenseGram to incorporate WSD into our models. Given a word and its context, SenseGram calculates a score ranging from -1 to 1 for each meaning of the given word. For a punning word, we would expect high and similar scores for two meanings. We decided to take the 10 largest scores for each word, so that each word has an embedding size of 10. For words with fewer than 10 scores, we padded the embedding with -1. The hidden size for the LSTM is also 10. Results are passed through 3 linear layers, compared to 2 layers previously.

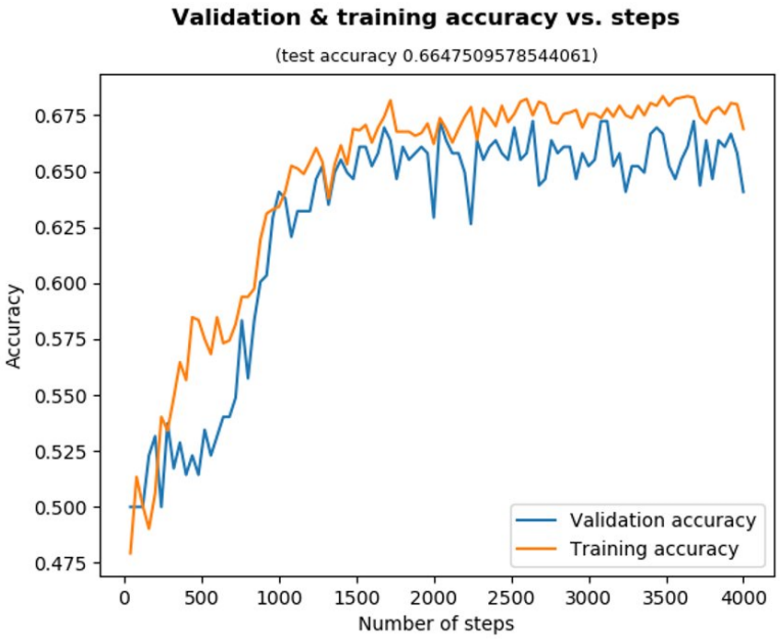

Figure 5: training, validation and test set results for subtask 1 RNN+WSD model

As seen in Figure 5, overfitting is no longer a problem; however, the WSD+RNN model also has a lower test accuracy than the RNN model. We conclude that the WSD+RNN model is more

generalizable and is not just memorizing high-frequency punning words. For example, a few puns in our dataset contain the word “glue”. Whenever the input contains “glue”, the RNN model outputs a high punning probability, as shown in Figure 6.

Compared to the RNN, the WSD+RNN model has a much larger number of false negatives, as seen in Figure 7. Unlike the RNN model, the WSD+RNN model is biased towards the negative label, which may be caused by SenseGram’s output. Instead of getting an ambiguous score for only the punning word, SenseGram may produce ambiguous scores for other words as well, which misleads the model.

```
Enter a sentence:
 i love glue glue glue
RNN model thinks the sentence is pun, with 99.9% confidence
WSD+RNN model thinks the sentence is not pun, with 52.6% confidence
```

Figure 6: Example of the RNN memorizing high-frequency punning words, while the WSD+RNN model can avoid this misclassification

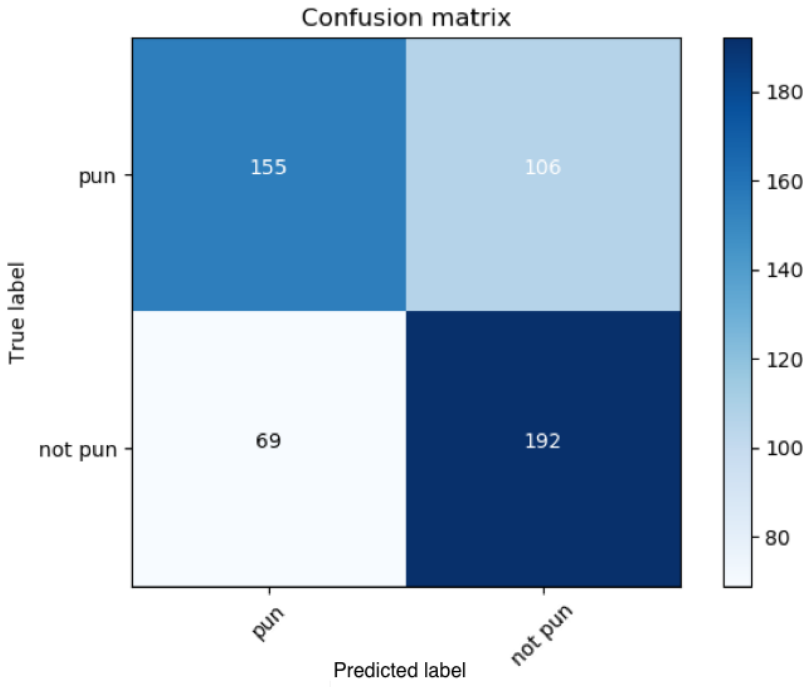

Figure 7: Confusion matrix for subtask 1 RNN+WSD model

Subtask 2: Pun location

Two models are implemented to locate the punning word, given that the sentence of interest contains a homographic pun.

1) RNN model:

We kept the RNN LSTM model from subtask 1, since the LSTM outputs a hidden state for every word along the sentence, so that all these hidden states can be passed into the linear layers. In the original SemEval dataset, 47% of the puns are located at the end of the context. If we use CNN or linear models, the entire sentence would be the input, so it is very likely that the model would be biased towards the end of a sentence. Since the RNN model can only examine one word at a time, it is more likely to be unbiased in terms of location.

Although memorization of keywords was problematic in subtask 1, it can be useful for subtask 2. This is another reason that we decided to apply the RNN model to pun location. Since some words are just more likely to be punning words, it would be helpful to memorize these high-frequency punning words, in order to locate them in sentences.

2) RNN+WSD model:

The motivation for using WSD is to make the model less dependent on the vocabulary of the dataset. The LSTM model's entire vocabulary is constructed from our dataset, so if the user input contains new vocabulary, the model becomes unreliable. In contrast, the WSD model uses scores calculated by SenseGram instead of word vectors. The score for each word indicates the distribution of meanings for this word. This distribution pattern is more dependent on the context, as opposed to the word's own unique word vector. Therefore, the WSD model would be applicable to words outside of our dataset vocabulary.

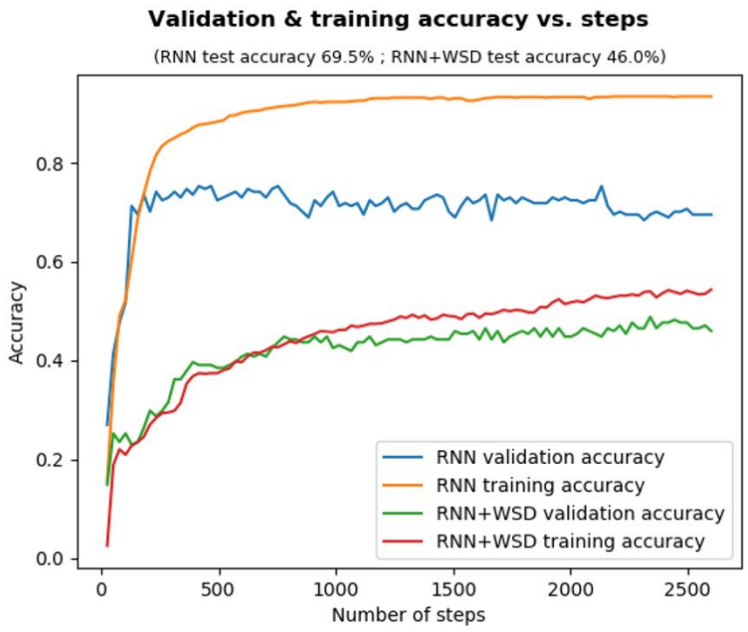

Figure 8: training, validation and test set results for subtask 2 models

Overall summary of models' performance:

Table 2 contains the summarized performance results of our models for both subtasks, measured against other models that participated in the SemEval challenge. However, it should be noted that we used a modified dataset.

	SemEval Contestants (original dataset) [2]			**SPINN (modified dataset)**	
	Max accuracy	Min accuracy	Mean accuracy	RNN	WSD+RNN
Subtask 1	85.3%	46.7%	67%	73.56%	66.48%
Subtask 2	66.4%	27.8%	43%	69.5%	46%

Table 2: Subtasks 1 and 2 performance of SPINN compared to other models

Ethical Issues

While the goal of SPINN is to detect puns, this project can be potentially used to develop a pun generator or to make chatbots more life-like. There are still some offensive and sexist puns in the dataset despite our manual filtering. Using these puns to train the neural net may therefore cause the model to generate offensive puns.

If SPINN is used to improve an AI's understanding of the complex, underlying meanings of human

language, it is possible that the AI will misinterpret unintended puns and ambiguous sentence inputs. This can be problematic if the AI then acts accordingly. While SPINN can be useful for the AI to understand wordplay, it should not be used when the AI still has trouble interpreting the literal, or most "obvious" meanings of human language.

Reflections and future directions

Due to our small and biased dataset, overfitting was a major obstacle for our models. Incorporating WSD reduced overfitting, but also lowered the prediction accuracy. The confusion matrix of the WSD model indicated that SenseGram brought some problems to the model while reducing memorization, but we have yet to identify the root of the problem. A potential next step would be to experiment with some other WSD algorithms and hopefully increase the accuracy of WSD+RNN model. Some studies show that SEL+cluster [5] yields better results than other WSD algorithms. To compensate for our small dataset, another alternative solution would be transfer learning using pre-trained language models, such as BERT [6].

We also observe that high accuracy does not necessarily imply generalizability. When evaluated on a small dataset, some of our models can achieve high accuracy by memorizing high frequency words. However, accuracy as a metric does not reflect the model's vocabulary limitations.

From the performance results of our models, we conclude that neural networks may not be the best tool to interpret wordplay. We saw that understanding puns often requires "common sense", and high-level interpretation of language is still a difficult task given that state-of-art technology still cannot understand natural language perfectly, let alone wordplay.

References

[1] “SenseGram,” *GitHub*, 29-Aug-2018. [Online]. Available: https://github.com/tudarmstadt-lt/sensegram. [Accessed: 30-Oct-2018].

[2] “SemEval-2017 Task 7: Detection and Interpretation of English Puns.” [Online]. Available: http://alt.qcri.org/semeval2017/task7/index.php?id=data-and-resources. [Accessed: 28-Oct-2018].

[3] T. Miller, C. Hempelmann, and I. Gurevych, “SemEval-2017 Task 7: Detection and Interpretation of English Puns,” *Proceedings of the 11th International Workshop on Semantic Evaluation (SemEval-2017)*, 2017. [Online]. Available: http://www.aclweb.org/anthology/S17-2005. [Accessed: 28- Oct- 2018]

[4] T. Miller and M. Turković, "Towards the automatic detection and identification of English puns", 2018. .

[5] “BERT,” *GitHub*, 01-Nov-2018. [Online]. Available: https://github.com/google-research/bert. [Accessed: 02-Dec-2018].

[6] T. Miller and I. Gurevych, 2015, “Automatic disambiguation of English puns”, in Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (ACL-IJCNLP 2015), Stroudsburg, PA: Association for Computational Linguistics, pp. 719–729.