Sarcasm Detection

Ratnadeep Mitra
Graduate School of Arts & Sciences
Georgetown University
rm1773@georgetown.edu

Abstract

Sarcasm detection is a prevalent linguistic phenomenon. Detection of sarcasm is of great importance and it is beneficial to many NLP applications, such as sentiment analysis, opinion mining and advertising. The goal of this research was to use deep learning with Recurrent Neural Networks (RNN) – more specifically Long-Short-Term-Memory (LSTM) networks that classifies tweets as sarcastic or not.

1. Introduction

Sarcasm is a linguistic tool – a mode of satirical wit, depending for its effect on bitter, caustic and often ironic language. Due to its figurative nature, sarcasm poses a great challenge for systems performing sentiment analysis. For example, let's take the sentence – "*I want to kill myself now*". There is absolutely nothing positive about this sentence, but I found this sentence in the comment section of an endearing video of kittens on YouTube. This is a classic case of the use of sarcasm, where the speaker does not actually want to kill himself, but he is in fact expressing how much the video affected him, albeit in a weird way. Surprisingly such type of comments or tweets are not uncommon where the speakers use negative words to express positive emotion and vice-versa. Hence, it is obvious that the detection of sarcasm in a text is very important – the failure of which will lead to misleading and inaccurate comprehension of sentences.

2. Related Work

In approaching the problem, the findings of several papers that provided a good understanding of the research already done was considered. Most of the research work used lexical feature-based classification such unigrams, bigrams and n-grams with polarity detection using libraries such as NLTK. The most popular machine learning algorithm used appeared to Naïve-Bayes. However, more recent research works have focused on Convolutional Neural Networks (CNN) or Recurrent Neural Networks (RNN) for sarcasm and the accuracy achieved is noticeably than Naïve-Bayes algorithm.

3. Datasets

The dataset used for this research is a collection of datasets graciously made available on GitHub by researchers who have worked earlier on sarcasm detection. The source of data is twitter and tweets with the hashtag "#sarcasm" where annotated as sarcastic. The dataset used contained 41613 unique annotated tweets. The distribution of sarcastic (class 1 - *orange*) and non-sarcastic tweets (class 0 - *blue*) is almost equal. Table 1 and Figure 1 show the distribution of sarcastic and non-sarcastic tweets in the dataset (1 denotes sarcastic tweet, while 0 denotes non-sarcastic tweet).

	Count
Sarcastic	19438
Non-sarcastic	22175

Table 1: Tweets Distribution

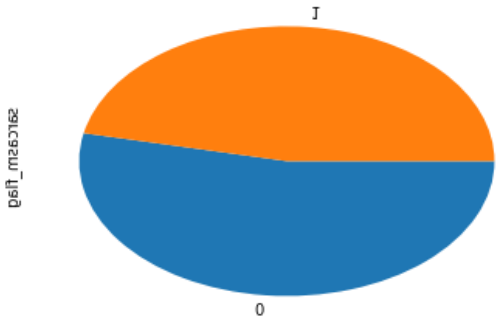

Figure 1: Tweets Distribution

4. Methods

The first step was checking for duplicate tweets. From an original dataset of 41755 tweets, duplicates were removed, and 41613 tweets were kept for analysis.

4.1 Data Pre-Processing

Several pre-processing tasks were performed to prepare the data for analysis. The standard tweet cleaning steps were then performed –

1. Removal of hyperlinks, website addresses or URLs
2. Removal of hashtags, preceded by '#'
3. Conversion of tweets to lowercase
4. Removal of remaining punctuations and special characters

The data cleaning steps were performed in the order mentioned above. It is important that Step 5 comes after Steps 2 and 3. Steps 2 and 3 locates and removes the entire unimportant words from the tweet. If Step 5 was performed before, then the "#" and "@" would be removed but the bogus word would remain in the tweet. Hence it is essential that Step 5 is performed after Steps 2 and 3. Stop-words were not removed because it was felt that removal of stop-words might affect the sentence construction and consequently the indicators for sarcasm.

After data cleaning, the raw tweets were encoded and stored as integers. The maximum length of each tweet was taken as 250, which is the maximum number of characters allowed in a tweet by Twitter. Tweets shorter than the maximum length were padded with zeroes so that every tweet has the same length. The target variable (0 and 1) were converted to categorical variable rather than integer, so that 0 and 1 are treated as classes and not just numbers.

4.2 Model Architecture

Previous research works suggested that recurrent neural networks and word embeddings give much better results than traditional feature engineering, sentiment analysis and Naïve-Bayes classification. RNN is a sequence of neural network blocks that are linked to each other like a chain. Each one is passing a message to a successor. They overcome traditional neural networks' shortcoming when dealing with sequential data, be it text, time-series or even DNA and are very capable of handling "long-term dependencies". Recurrent neural network uses its reasoning about previous events to inform the later ones. They are networks with loops in them (as shown in Figure 2), allowing information to persist.

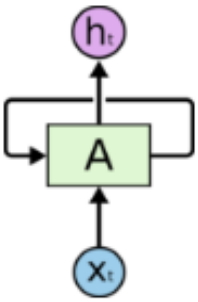

Figure 2: Recurrent Neural Network [3]

In the above diagram, a chunk of neural network, A, looks at some input x_t and outputs a value ht. A loop allows information to be passed from one step of the network to the next. Thus, like feedforward neural networks, RNNs can process data from initial input to final output. But, unlike feedforward neural networks, RNNs use feedback loops such as Backpropagation Through Time or BPTT throughout the computational process to loop information back into the network. This

connects inputs together and is what enables RNNs to process sequential and temporal data.

One drawback of the standard RNNs is the vanishing gradient problem, which affects the performance of the neural network because it cannot be trained properly. This happens with deeply layered neural networks, which are used to complex data, for example long bodies of text. Standard RNNs use a gradient-based learning which degrades with increasing complexity.

Long-Short-Term-Memory (LSTM) networks are a special kind of RNN, capable of learning long-term dependencies. RNNs built with LSTM units categorize data into short-term and long-term memory cells. Doing so enables RNNs to figure out what data is important and should be remembered and looped back into the network, and what data can be forgotten. LSTM networks manage to keep contextual information of inputs by integrating a loop that allows information to flow from one step to the next. LSTM predictions are always conditioned by the previous experience of the network's inputs.

On the other hand, the more time passes, the less likely it becomes that the next output depends on a very old input. This time dependency distance itself is as well as contextual information to be learned. LSTM networks manage this by learning when to remember and when to forget, through their forget gate weights. [medium]. A forget gate is responsible for removing information from the cell state. The information that is no longer required for the LSTM to understand things or the information that is of less importance is removed via multiplication of a filter. Figure 3 shows the picture of a simple LSTM memory cell with various gates.

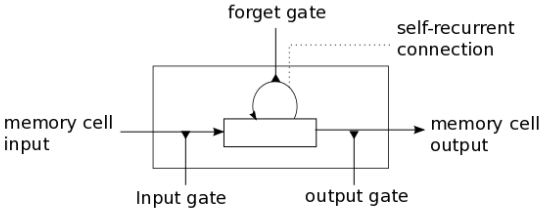

Figure 3: LSTM Memory Cell [4]

Although the text of a tweet is not expected to be very long (250 characters per tweet), the advantages of RNN and LSTM when it comes to processing sequential data made it the model of choice.

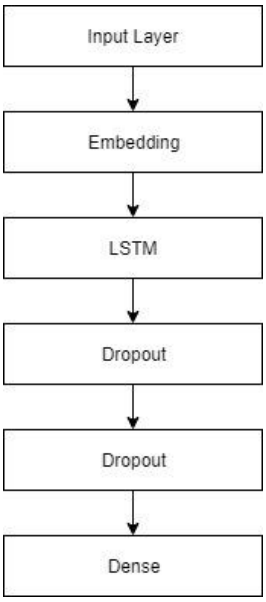

Figure 3: RNN-LSTM Model

The deep learning model shown in Figure 3 put together embedding layer, followed by LSTM layer with 2 dropout layers and finally a fully connected dense output layer. The loss function used was binary cross-entropy.

5. Results

During the initial phase, the model was trained for 50 epochs, but it was found that it the loss and accuracy starts asymptoting after about 10 epochs. So, the model was then trained for 12 epochs and the loss and accuracy were tracked which are shown is figures 4 and 5 below respectively.

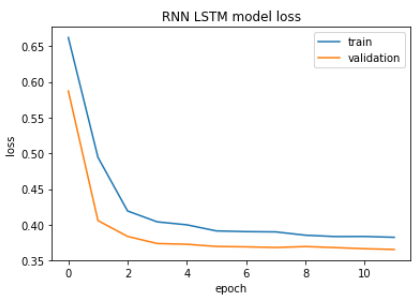

Figure 4: RNN-LSTM Model Loss

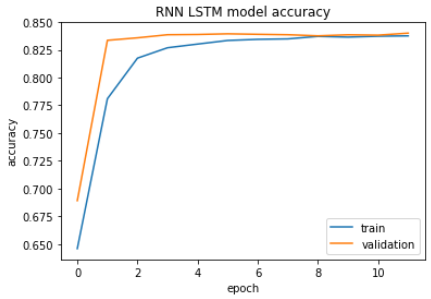

Figure 5: RNN-LSTM Model Accuracy

It was observed that the loss decreased nicely and rapidly but there might still be room for improvement. The validation accuracy reached about 85% near the 10th epoch, which is quite high. The model had two dropout layers, which introduces some sort of randomness of the network and results in training a slightly different model every epoch. Despite that, we did not observe any fluctuations in the accuracy or loss. It's a usual way to stop training when some metric computed on validation data starts to grow. This is a usual indicator of overfitting but fortunately, for our model, the model accuracy stayed almost constant near 80% and the loss very gradually decreased with increasing epochs.

The model performed well on the validation set and this configuration was chosen as the final model. Then the model was evaluated on the test set. The loss on the test set was 0.371 and the accuracy achieved was around 84%, which was very satisfactory.

6. Conclusions

Although the LSTM network gave very high accuracy, there are still ways to improve our model further. One of the important things while detecting sarcasm, is the context. Without the context it become difficult very to identify when a text is sarcastic and when it is not. One of the helpful things for identifying the context is Attention. To pay attention to a part of the source sequence, is to assign a high attention weight relative to the rest of the sequence. It means that the dynamic context contains more information corresponding to the parts of the source sequence that are interesting. In other words, attention-based LSTM mechanism allows the network to refer to the input sequence, instead of forcing it to encode all information into one fixed-length vector. Two other relatively new concepts that can be considered as a future work on sarcasm detection are emoji embedding and phrase embedding. Emojis play a major role in conveying sentiments nowadays and if we could utilize the co-occurrence and their meanings in the context of social media posts, it is going to be very helpful. Phrase embedding can learn phrase representations and they can be useful in detecting the commonly occurring sarcastic phrases.

Ultimately, RNN-LSTM model is well-suited for sarcasm detection, but a more complex model might give an even higher rate of accuracy.

References

[1] Husain, Hamel. "How To Create Data Products That Are Magical Using Sequence-to-Sequence Models." Towards Data Science, 18 Jan. 2018.

[2] Banerjee, Suvro. "An Introduction to Recurrent Neural Networks." Medium, 23 May 2018.

[3] Olah, Christopher. "Understanding LSTM Networks." Colah's blog, 27 Aug. 2015.

[4] Carrier, Pierre Luc. "LSTM Networks for Sentiment Analysis." Deeplearning, 15 Jun. 2018.

[5] Moawad, Assaad. "The magic of LSTM neural networks." Medium, 2 Feb. 2018.

[6] Srivastava, Pranjal. "Essentials of Deep Learning: Introduction to Long Short Term Memory." Analytics Vidhya, 10 Dec. 2017.