RNN Clickbait

Daniel Alemu, A13874961

Abstract

RNN's, or recurrent neural networks, are a reliable way to process sequential data. It uses a hidden layer to hold on the previous output, to make predictions based on past input. This works well on text based data where grammar is very important. We will be looking at a form of text called clickbait. However I plan to try a more improved model of RNN taken with inspiration from LSTMs. LSTMs (long short term memory) are a more complicated form of RNN. LSTMs have forget gates that decide whether or not the input is important enough to update the hidden layer. The LSTM has a reported high accuracy on grammar-related textual data. So I plan to use just the forget gate from LSTMs in my revised RNN model.
Other methods I will bring in are the use of word embeddings, dropout, and stacking RNNs. The revised RNN model will produce a clickbait sample Hopefully by these additions to the original RNN model, it will produce better samples than the RNN baseline.

WHAT HAPPEND NEXT WILL BLOW YOUR MIND
WHAT POKEMON CHARACTER ARE YOU?
16 WAYS TO
YOUR
IT'S A TRAP!
RESTORE YOUR FAITH IN HUMANITY !!!
GAME OF THRONE
O YOU BELONG
Six Corgi Pup

Introduction

Clickbait is very common on YouTube and any sort of social media out there. There's many reasons for creators to use clickbait. To promote their brand or to get paid by how many people click their stuff. Ads are doing it, YouTubers are doing it, Redditors, etc. Now that everything's online, it is imperative that companies build a catchy online persona to get business.

However, online businesses are not good at making clickbait. They miss trendy, use a meme inappropriately, use a meme from 7 years ago, the list goes on. And it's not easy, clickbait is subjective and changes very quickly with the times. A meme that was made a month ago is old now. Minecraft, the popular game, used to guarantee YouTubers a stable amount of views just for putting the name “Minecraft” in their title. Until Minecraft fell in terms of popularity, only to rise again recently. It requires constant attention to stay informed about what’s popular.

Despite that, I think clickbait titles are original and easy to manipulate. I believe my model will be able to make convincing clickbait (assuming the dataset is actually easy)..

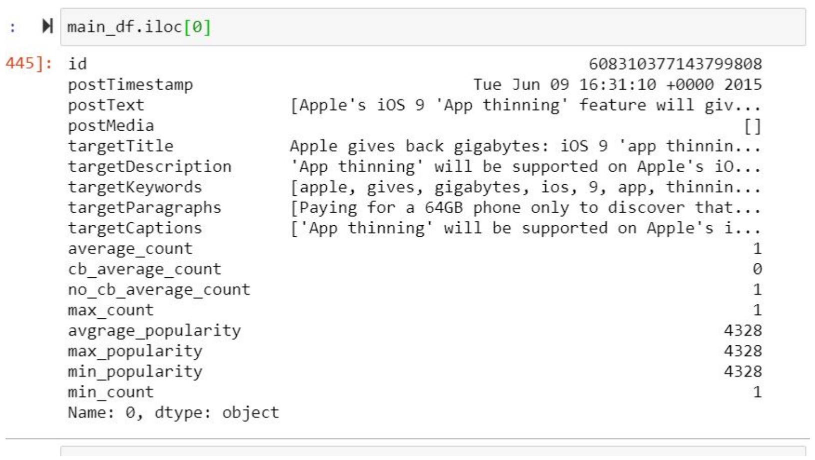
Method

The dataset comes from here: https://www.clickbait-challenge.org/. A clickbait challenge with the goal of identifying clickbait. It comes with a couple of json files including 2459 training data points and 80013 untrained data points. Each data point comes with relevant text data like the title and keywords that give context the title context. A separate json file contained the labels of each data point in the same order, so for the label column the training had “clickbait” or “no-clickbait”. There were 762 data points on clickbait and 1697 data points that weren’t clickbait.

The data I used was only from the training set. Two files on the training were read into a pandas dataframe and then merged so that the new dataframe had the relevant data and the labels.

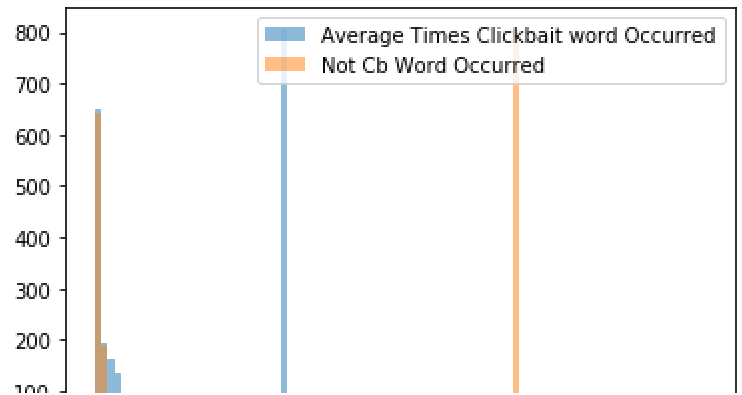
EDA

The RNN side of the model only needed the title data for each data point. However would also later include word embeddings, so statistics were applied to the keywords. One of these stats was the popularity of every keyword in the dataset. The most popular words were ‘news’ and ‘and’. For the clickbait, the two most popular words were ‘uncategorized’ and ‘news’. From non-clickbait the most popular words were ‘news’ and ‘breakingnews’.



```
main_df.iloc[0]

445]: id                                              608310377143799808
      postTimestamp                        Tue Jun 09 16:31:10 +0000 2015
      postText              [Apple's iOS 9 'App thinning' feature will giv...
      postMedia                                                         []
      targetTitle           Apple gives back gigabytes: iOS 9 'app thinnin...
      targetDescription     'App thinning' will be supported on Apple's iO...
      targetKeywords        [apple, gives, gigabytes, ios, 9, app, thinnin...
      targetParagraphs      [Paying for a 64GB phone only to discover that...
      targetCaptions        ['App thinning' will be supported on Apple's i...
      average_count                                                      1
      cb_average_count                                                   0
      no_cb_average_count                                                1
      max_count                                                          1
      avgrage_popularity                                              4328
      max_popularity                                                  4328
      min_popularity                                                  4328
      min_count                                                          1
      Name: 0, dtype: object
```

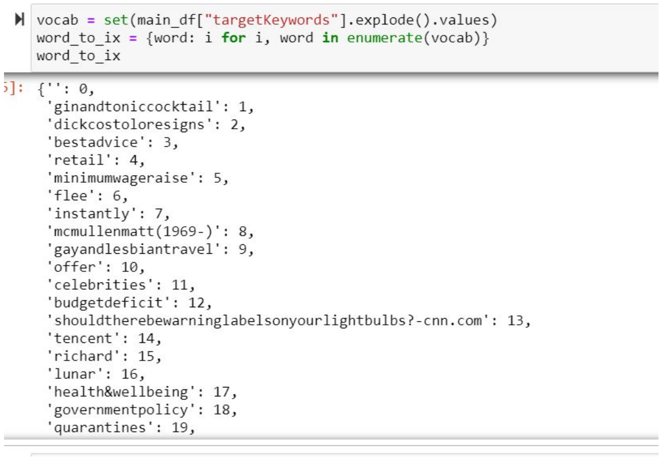


Interestingly, the non-clickbait more commonly included news and sports related terms.

The second statistic was the count of each word in the entire dataset. The count was first taken from the dataset as a whole, then applied to each data point where each keyword was replaced with the amount of times it appeared. Now that each data point has a count for each keyword, it was then averaged. This average count was used for the count of how many times that keyword appeared in general, how many times that keyword appeared in clickbait, and how many times it appeared in non-clickbait. I tried the same thing with min and max but average visualized the best. As you can see some words appear more often in clickbait, some more often in non-clickbait. So it shows that clickbait keywords can actually help the model differentiate its replication from non-clickbait.

Word Embeddings

As mentioned above, the keywords are a good feature to include in the model. Pytorch cannot handle string values in its tensors. So pytorch has a embedding package where you pass in the entire vocabulary of the keywords with an integer id for every word. The embedding package then assigns values for multiple meanings or semantics for every id passed in. So it learns multiple meanings for every word. For example, for the word "professor" the id will be passed in and the model will give values to relation of "professor" to "human" or "school", etc.

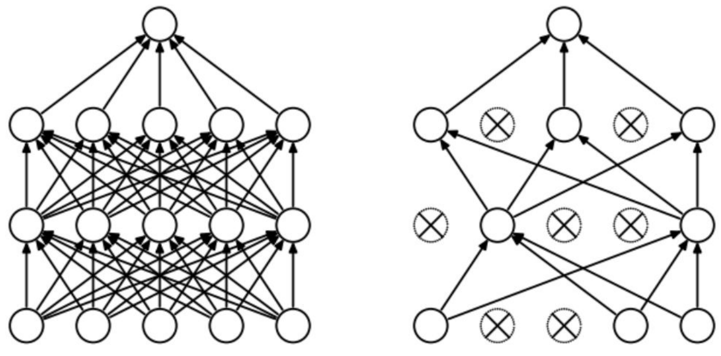


```
vocab = set(main_df["targetKeywords"].explode().values)
word_to_ix = {word: i for i, word in enumerate(vocab)}
word_to_ix
```

```
5]: {'': 0,
     'ginandtoniccocktail': 1,
     'dickcostoloresigns': 2,
     'bestadvice': 3,
     'retail': 4,
     'minimumwageraise': 5,
     'flee': 6,
     'instantly': 7,
     'mcmullenmatt(1969-)': 8,
     'gayandlesbiantravel': 9,
     'offer': 10,
     'celebrities': 11,
     'budgetdeficit': 12,
     'shouldtherebewarninglabelsonyourlightbulbs?-cnn.com': 13,
     'tencent': 14,
     'richard': 15,
     'lunar': 16,
     'health&wellbeing': 17,
     'governmentpolicy': 18,
     'quarantines': 19,
```

Dropout

RNNs sometimes implement dropout in order to avoid overfitting. Normally, models do what's called coadaptation. Which is making connections between different inputs and words and assuming that two inputs must be paired together based on what it's seen from the training data. So I'll be using dropout to generalize to other clickbait.


(a) Standard Neural Net (b) After applying dropout.

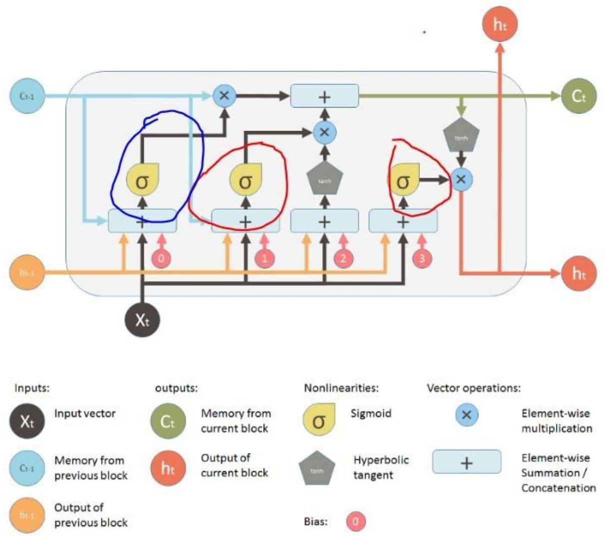
Implementing in RNN is when the hidden layer is zeroed out with a certain given probability. So the RNN's layer will occasionally be inactive during some inputs, requiring some layers to also handle inputs that the zeroed out neurons had handled.

Stacked RNNS

This addition was simple. Rather than adding 1 nn.RNNCell from pytorch, I added 2. So there are two hidden layers that are updating in every forward propagation corresponding to each nn.RNNCell. This is similar to the structure of the LSTM but this idea came from this blog: http://karpathy.github.io/2015/05/21/rnn-effectiveness. This will make the model fit the data better (by adding another layer) but risk overfitting.

LSTM Forget Gates

These gates are similar to the dropout where instead of neurons, for 1 data point the hidden layer will be zeroed out. This prevents updating when the model decides a certain input should be forgotten. How it works is that after getting the output from the RNNs and the hidden layers have new values for the next iteration, the input and the hidden layers will be flattened to a 1D tensor and concatenated to each other. This is fed to a small sigmoid NN where whatever is outputted is multiplied by the hidden layer for the next iteration. So since the output of a sigmoid function is either 0 or 1, either the hidden layer for this iteration will be zeroed out and not updated, or it will be kept and multiplied by 1.



Tuning Parameters

The following parameters were checked and changed:

1. Dropout rate at 50%, 80%, or 100%
2. Number of keywords used
3. Hidden layer size
4. Count features (had to remove because it took too long)

Then, compared the cross entropy loss rate between these parameters and picked the smallest loss between each.

Experiment

Dropout Rates

	.5	*.8*	*1*
Entropy Loss	*2.8155877590179443*	*2.8498024940490723*	*2.944298267364502*

So I decided to use the 50% dropout parameter. Also for hidden layers, I use 100 dimensions. The other parameter was 1000 dimension, but that didn't run very fast.

Baseline RNN and Stacked RNN

	Baseline	*Stacked*
Entropy Loss	*2.743119239807129*	*2.79838490486145*

After a couple of times, the loss values II got were the following in the tables above. We don't know for sure but I believe both the models are bad for this data format. ~.03 loss worth of distance.

Conclusion

In conclusion I found that the standard RNN performed better than my model. I assume that this is because either the keywords were a bad addition to the model or that the model thought clickbait and non-clickbait were too similar. The LSTM and its forget gate are better suited for my grammar related text so perhaps this isn't the right dataset for this model. Maybe I can later check if the accuracy of this model with a new grammar based dataset.

Clickbait is a pretty hard idea to research since models have a hard time catching it. I have recently heard that companies have begun switching from RNN models and LSTMs to Attention based models. The old RNN model is outdated and unreliable. Most likely it was made for this kind of data.

Further Work

There are a couple of ways to improve accuracy that, after seeing the results, I have thought about later implementing. Including trying out a new grammar based, textual dataset.

Removing unimportant keywords like 'and' and 'a'. These words don't represent clickbait or non-clickbait, so they don't mean anything to us. Having words like this with a higher count would probably mess with the weights during training. Hence removing these outliers may improve the model's accuracy.

Trained without outlier data. I made 3 versions of the dataset where rows with an average count for keywords < 5, 10, 15 were removed. From the EDA, the low counts seemed to contain most of the data points but the most differentiation between clickbait and not was in the higher counts. Therefore it might result in better generated clickbait if the model only read the data points with high average counts.

https://www.clickbait-challenge.org/
http://karpathy.github.io/2015/05/21/rnn-effectiveness