Automatic Voice Title Generation by Extractive Summarization of Long Product Titles using Bi-LSTM

Snehasish Mukherjee
Stanford University
mukherji@stanford.edu

Phaniram Sayapaneni
WalmartLabs, Sunnyvale
Phaniram.Sayapaneni@walmartlabs.com

Abstract

The proliferation of voice enabled smart assistants has created a huge market for Voice commerce but the user experience suffers due to the lack of short voice friendly titles. In this paper we investigate supervised techniques for automatic generation of short titles given, the long one. We show that even with a small parallel dataset of around 19,000 title compression examples, LSTM based architectures work well without over-fitting. To improve the fluency of the generated short titles we propose and implement a modified Kneser-Ney based trigram language model trained on 1 Million grocery search queries. We also finetuned SOTA architectures like BERT, GPT-1 and XLNet and applied to our problem but found that relatively simpler architectures outperformed them. Our self-attentive BiLSTM + CRF model achieved a ROUGE-1 F1 score of 0.85 which beat BERT by a huge margin.

1 Introduction

Voice Commerce is already a 2 Billion dollar industry and still growing very fast. In voice commerce, the product titles need to be read out to the user at various stages of the user journey like search, browse, add to cart etc. This presents a unique problem since e-commerce product titles are long and not suitable for voice based interaction. Reading out a title like “Puffs Plus Lotion Facial Tissue, 4 Mega Cubes, 72 Facial Tissues Per Cube (288 tissues total)” takes too long and deteriorates the user experience which is a major hindrance in the adoption of this technology. The e-commerce industry gets around this problem by manually generating voice friendly titles, or voice titles in short, for each product. This is not scalable for large assortment sizes of 50 Million items or more and hence majority of these products do not have voice friendly titles.

In this project we aim to solve this problem by leveraging Deep Learning techniques to automatically compress a long product title to generate the corresponding short title. Given a long product title string, we feed it as the input to a RNN based neural network to predict the output short title. For example, given the long title “Puffs Plus Lotion Facial Tissue, 4 Mega Cubes, 72 Facial Tissues Per Cube (288 tissues total)” the network should generate the short title “facial tissue”.

Formally, we want to build a model $M$ that given the long title $L = \left( L_i \right)_{i=1}^{n}$ containing $n$ words $L_1, ..., L_n$ as input, can generate the corresponding short title $S$ as the output, so that

$$S = M(L)$$

where $S = \left( S_i \right)_{i=1}^{m}$, has $m$ words $S_1, ..., S_m$, $m \leq n$ and $S_i := L_{L(i)}$, i.e., $S$ is a subsequence of $L$. Additionally, $S$ should be reasonably descriptive and retain the root product identity i.e., the short title for “Almond milk” should be “milk” and not “Almond”.

CS230: Deep Learning, Winter 2018, Stanford University, CA. (LateX template borrowed from NIPS 2017.)
2 Related work

Our work can be classified under the broad category of text summarization. This is an area of active research and has evolved quite a bit with the advent of RNNs. [21, 7, 2, 12] present comprehensive surveys of neural as well as classical text summarization techniques. While document level summarization [5, 4, 19] deals with the problem of generating document level summaries of the content, our work is best described as sentence compression [27, 26, 6, 16, 30, 9, 10] which involves summarizing long sentences to shorter ones while preserving the core intent.

There are 2 distinct flavors of summarization independent of the source granularity; Abstractive and Extractive. Abstractive summarization [6, 20, 25, 4] can produce summaries consisting of words not present in the input, while Extractive summarization [10, 9, 30, 26] aims to generate summaries using words or sentences extracted from the original input. We will focus on neural approaches to the extractive flavor of sentence compression for the remainder of this section.

A popular approach to extractive sentence compression is to model the problem as a sequence-to-sequence learning problem using an encoder-decoder based architecture [10, 9, 16]. The encoder learns a distributed representation of the input sentence which is then fed to the decoder which is trained to produce a binary 0/1 label for each input word denoting whether to delete or keep that word in the output summary. Beam search can be used [10, 27, 28] to sample from the decoder output probability distribution to generate the most likely compression.

[26] and [30] present 2 approaches different from the encoder-decoder based approaches. [30] uses Reinforcement Learning with KEEP and DELETE policies. Words are kept or deleted based on the policy while a pre trained language model provides feedback on the generated summary. This results in the system learning the optimal policy over time. Like all the other deletion based compression models, [26] still tags words in the input sequence with a 0/1 label, but treats it as a sequence labelling problem, instead of a sequence generation problem. Hence it does away with the decoder and employs stacked BiLSTM layers with a final CRF layer at the output that does the 0/1 classification. Our work in this project closely follows this approach with minor modifications.

Most of the techniques discussed here, with the exception of [26], require considerable amount of training data. While [16, 9, 30] are trained on the Gigaword corpus, [10] is trained on 2 Million news headline and summary pairs. Though these 2 Million examples were synthetically generated following the method in [11], it is based on syntactic structure of the English language and its parse trees. [9] also proposes an unsupervised approach involving generating training data by adding noise, but the results lag behind supervised approaches. Since the distribution of the words in the product titles, and in general the vocabulary of our problem domain, is different from general English, data sparsity and unavailability of e-commerce specific embeddings pose a challenge for us. Interestingly, [26] reports state of the art performance with only 10000 pairs of original and compressed sentences, which is the reason we chose to implement a modified version of this approach to solve our problem of product title compression.

3 Dataset and Features

We use 2 different private data sets from WalmartLabs. The first dataset consists of 19,254 human generated short product titles and their original long counterparts. These (short query, long query) pairs correspond to the top selling grocery items and are the most heavily used in voice shopping. We call this the “compression dataset”. The second data set consists of around 12 Million anonymous online grocery search queries, and their frequencies taken over a 5 month period. Each (query, frequency) pair is what was observed in a day, hence multiple rows exist for the same query, one for each day. We call this the “search query dataset”.

**Normalization:** We normalize the product titles and the search queries by converting to lower case, removing all non-ascii characters, padding commas with spaces, replacing & with “and”, squeezing consecutive spaces and removing all non-alphanumeric characters with the exception of dot(“.”), percent(“%”), single quote(“’”) and comma(“,”). The commas were padded with spaces so that they can be extracted as tokens. Next we filter out all those samples from the compression dataset which are not purely extractive. After the last step we were left with 15,856 parallel title compression examples. From the search query dataset we remove all queries with total frequency less than 5. This leaves us with 1,108,043 top searched queries.
Training/Validation/Test: We use a 90/10 split of the compression dataset for the training and test set respectively. Additionally before every epoch another randomly chosen 10% data is set aside as the validation set. The search query dataset is not directly used for training. Table 1 lists a few statistics of the datasets, while figure 3 shows that short title lengths and the search query lengths are similarly distributed with a mode around 2. Table 2 lists a few examples of long-short title pairs from the compression dataset.

Table 1: Dataset Statistics

<table>
<thead>
<tr>
<th>Statistic</th>
<th>Long Titles</th>
<th>Short Titles</th>
<th>Search queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>No of samples</td>
<td>15,856</td>
<td>15,856</td>
<td>1,108,043</td>
</tr>
<tr>
<td>Vocabulary size</td>
<td>8,032</td>
<td>1,833</td>
<td>159,994</td>
</tr>
<tr>
<td>Median length</td>
<td>10</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>95 percentile length</td>
<td>15</td>
<td>5</td>
<td>6</td>
</tr>
</tbody>
</table>

Table 2: Examples from compression dataset

<table>
<thead>
<tr>
<th>Long Title</th>
<th>Short Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>great value organic extra virgin olive oil , 25.5 oz</td>
<td>olive oil</td>
</tr>
<tr>
<td>welch’s bottle 100% juice coconut water tropical grape berry , 64 oz</td>
<td>grape berry juice</td>
</tr>
<tr>
<td>huy fong foods chili garlic sauce , 18 oz</td>
<td>chili garlic sauce</td>
</tr>
<tr>
<td>great value crunchy granola bars , oats and honey , 1.4 oz , 6 count</td>
<td>granola bars</td>
</tr>
</tbody>
</table>

4 Methods

We model the problem of generating voice friendly titles as a supervised sequence labelling problem where we are given long product title and we have to label each token with a 0 or 1. The words tagged 1 by the model will be a part on the summary. Formally, given the input training pair $(X,Y)$ the model tries to learn the optimal parameter $\theta'$ such that

$$\theta' = \arg\max \sum \log p(Y|X; \theta)$$

(1)

where $Y_i \in \{0,1\}$ and $X_i \in \mathbb{R}^d$ is an embedding of a word in a d-dimensional vector space. Additionally in our problem to we keep $|X| = |Y|$ and use padding to guarantee this.

Since our inputs and outputs are sequential, we use RNNs, and since our input and out lengths are the same we choose not to use any encoder decoder based architecture. Instead we use an architecture...
A broad level view of our model architecture is shown in TODO. Fixed length input passes through an embedding layer to enter a stack of (Bi)LSTM layers. Describe your learning algorithms, proposed algorithm(s), or theoretical proof(s). Make sure to include relevant mathematical notation. For example, you can include the loss function you are using. It is okay to use formulas from the lectures (online or in-class). For each algorithm, give a short description of how it works. Again, we are looking for your understanding of how these deep learning algorithms work. Although the teaching staff probably know the algorithms, future readers may not (reports will be posted on the class website). Additionally, if you are using a niche or cutting-edge algorithm (anything else not covered in the class), you may want to explain your algorithm using 1/2 paragraphs. Note: Theory/algorithms projects may have an appendix showing extended proofs (see Appendix section below).

Finally, since we are tied down by the lack of parallel training examples from our specific domain, we try out transfer learning by finetuning state-of-the-art pre-trained models, namely BERT [8], GPT-1 and XLNet [29].

5 Experiments/Results/Discussion

We use the 15,856 rows of data for training BiLSTM based neural networks and fine-tuning state-of-the-art pre-trained models. We use ROUGE-1 [14] scores for precision, recall and F1. Additionally,
we also look at the number of unseen test cases where the model outputs a summary of all 0s.

The results for the various model architectures are presented in Table 3. We train various different combinations. You should also give details about what (hyper)parameters you chose (e.g. why did you use X learning rate for gradient descent, what was your mini-batch size and why) and how you chose them. What your primary metrics are: accuracy, precision, AUC, etc. Provide equations for the metrics if necessary. For results, you want to have a mixture of tables and plots. If you are solving a classification problem, you should include a confusion matrix or AUC/AUPRC curves. Include performance metrics such as precision, recall, and accuracy. For regression problems, state the average error. You should have both quantitative and qualitative results. To reiterate, you must have both quantitative and qualitative results! If it applies: include visualizations of results, heatmaps, examples of where your algorithm failed and a discussion of why certain algorithms failed or succeeded. In addition, explain whether you think you have overfit to your training set and what, if anything, you did to mitigate that. Make sure to discuss the figures/tables in your main text throughout this section. Your plots should include legends, axis labels, and have font sizes that are legible when printed.

### Table 3: Results

<table>
<thead>
<tr>
<th>Model</th>
<th>Precision</th>
<th>Recall</th>
<th>F1</th>
<th>Num 0</th>
</tr>
</thead>
<tbody>
<tr>
<td>3-LSTM</td>
<td>0.831</td>
<td>0.849</td>
<td>0.816</td>
<td>43</td>
</tr>
<tr>
<td>3-BiLSTM</td>
<td>0.883</td>
<td>0.876</td>
<td>0.862</td>
<td>25</td>
</tr>
<tr>
<td>3-BiLSTM+SA</td>
<td>0.878</td>
<td>0.882</td>
<td>0.862</td>
<td>23</td>
</tr>
<tr>
<td>3-BiLSTM+SA+KN</td>
<td>-</td>
<td>-</td>
<td>0.835</td>
<td>23</td>
</tr>
<tr>
<td>3-BiLSTM+SA+CRF</td>
<td>0.869</td>
<td>0.872</td>
<td>0.852</td>
<td>8</td>
</tr>
<tr>
<td>BERT</td>
<td>0.797</td>
<td>0.780</td>
<td>0.750</td>
<td>-</td>
</tr>
<tr>
<td>Open AI GPT-1</td>
<td>0.730</td>
<td>0.656</td>
<td>0.656</td>
<td>-</td>
</tr>
<tr>
<td>XLNET</td>
<td>0.690</td>
<td>0.655</td>
<td>0.640</td>
<td>-</td>
</tr>
</tbody>
</table>

### 6 Conclusion/Future Work

We had started out with the problem of automatically generating voice friendly product titles under the constraint of a small training data set and showed that many different variations of LSTM based network architectures work well for this problem. Additionally, we also used pre-trained SOTA models like BERT, GPT-1 and XLNet to solve our problem. However contrary to our expectations, their results were lagging behind relatively simpler techniques. This needs further investigation. We also found that our solution can be sometimes prone to returning all Os. Our work did not investigate this problem in detail and we will take this up in future. Finally, we will try to scale up this solution to deploy this for hundreds of millions of product titles.
7 Contributions

Both members contributed significantly towards successful completion of the project. Snehasish (SUNetId: mukherji) led the project. Snehasish came up with the project idea, did the literature review, collected cleaned and pre-processed the training data, tested some warm-up models on the Google Research news dataset, designed and trained all the BiLSTM, Attention and CRF based models, built the Kneser-Ney smoothing based Beam search, and deployed custom evaluation metrics for Keras.

Phaniram trained a word2vec skip-gram based word embedding, obtained the t-SNE visualization, implemented a synthetic training data generation model, researched and tried out several state-of-the-art models for our problem including BERT, GPT-1, and XLNet.

Code The code for all the ideas implemented here, and all other associated experiments, can be found at the github url: https://github.com/isnehasish/title_sum. Detailed instructions on how to use the notebooks to reproduce the results will be uploaded in the README.md. Please note that the data set used for this project is not public.

References


