Playlist Title Generation Using Sequence to Sequence

Sofía Samaniego de la Fuente
Department of Statistics
Stanford University
`sofiasf@stanford.edu`

Advisor: Richard Socher

Abstract

In this project we propose a model that generates the name of a playlist given information about its tracklist. We first use a standard sequence-to-sequence model with unidirectional single-layer LSTM cells for both the encoder and the decoder and GloVe embedding initialization. We then add drop-out for regularization and introduce an attention mechanism to establish alternative connections between relevant words in song titles and playlist names. Finally, we provide an interactive console that generates playlist name suggestions given an input tracklist, which could serve as a useful automated suggestion system for digital music services users.

1 Introduction

Coming up with an original and catchy title for a playlist is most definitely a challenging task. In this project we propose a neural model dedicated to making the name search easier for those in the hunt for the perfect playlist title. In particular, we attempt to construct a network that builds an internal representation of the names of the tracks included in a playlist and then uses natural language techniques to generate a title that could be close to what a human might devise.

Generating a playlist name is somewhow related to the task of creating a condensed representation of an input text; that is, to the task of summarization. Specifically, this problem is similar to abstractive summarization, as a title generally captures the gist of a playlist without explicitly using segments of the song titles included in it, as an extractive method does. We hence approach this problem through a neural sequence-to-sequence model, the most viable and promising approach for abstractive summarization available at present. These models, first introduced by Sutskever et al. [1], typically use an LSTM cell to map an input sequence to a vector of fixed dimensionality and then another LSTM to decode the target sequence from the vector. They have proven successful in a variety of tasks such as machine translation, question answering, bot interaction, and automatic summarization.

There are numerous challenges to overcome in this project. In particular, it is hard to quantify how well a title conveys the relevant information of a playlist, aspects of which may not appear in the input text. Furthermore, deciding whether a playlist name is "good" is a highly subjective task that is difficult even for humans, so coming up with a good automatic evaluation metric is particularly challenging. To evaluate the final performance of our model, we use both a quantitative measure, namely the ROUGE score, as proposed in NITS's annual Document Understanding Conferences [2], and a qualitative one, namely the analysis of example results.

The rest of the paper is organized as follows. In Section 2 we discuss relevant literature for this project. In particular, we provide a brief review on previous work in sequence-to-sequence models for abstractive summarization. We note, however, that, to our knowledge, no one has ever proposed the specific task of generating playlist titles using neural natural language processing techniques. In Section 3, we delineate the architecture of our standard and extended neural models. We then describe and analize our dataset and summarize our experimental results in Section 4. We show that our approach yields in human-like and creative results for most input playlists. Finally, we conclude the project with ideas for future work in Section 5.

2 Background

Automatic summarization refers to the process of creating a condensed, fluent, and accurate representation of an original input text using natural language processing techniques. While this problem has long been a research topic of great interest in computer science, in recent years this interest has spiked due to an increase in availability of text data from a variety of sources, including a huge amount of online articles and documents. There are two general approaches for this problem: extraction and abstraction. Extractive methods work by assembling summaries from important sections taken verbatim from the source text; meanwhile, abstractive methods, which will be our main focus, aim to convey relevant information by generating novel words and phrases which may or may not appear in the source text [3].

In 2015, Rush et al. [4] proposed a local attention-based model that generates each word of the summary conditioned on an input sentence. Their model achieved significant performance gains on the DUC-2004 evaluation dataset, consisting of 500 document-summary pairs. Nallapati et al. [5] modeled abstractive text summarization using attentional encoder-decoder RNNs in 2016 and outperformed several state-of-the-art results on two different datasets: the DUC-2004 corpus and the CNN/Daily Mail corpus. Finally, See et al. [6] proposed an extension to the standard sequence-to-sequence architecture for abstractive text summarization in 2017. Their model tackled two shortcomings of the standard architecture by introducing the use of a hybrid pointer generator network to aid in the liable reproduction of factual details and coverage to discourage repetition. They outperformed the current abstractive state-of-the art by over 2 ROUGE points in the CNN/Daily Mail dataset.

Text summarization is without a doubt a non-trivial task for computers, since this process implies the use of human knowledge and language capability, both of which computers lack. While a lot of progress has been made in this area with the recent success of neural models and other data driven approaches, this remains a very challenging task and attaining high levels of abstraction remains an open research question.

3 Methods

In this section we describe our standard baseline sequence-to-sequence model, the regularized version using dropout, and the extended attention-based model, modeled off the attention mechanism introduced by Bahdanau et al. in 2015 [7].

3.1 Standard sequence-to-sequence model

Our baseline model, depicted in Figure 3.1, is similar to that presented by Luong et al. [8] in their Neural Machine Translation (seq2seq) Tutorial.

We start by tokenizing the words in the song names of our playlists and their corresponding titles into a sequence of tokens which roughly correspond to words. We will represent the tokens of the tracklist of each playlist as a sequence of integers

$$x_1, x_2, \dots x_T$$

where T is the arbitrary input length, each $0 \leq x_t \leq |V|$ is the index of a token in the vocabulary, and $|V|$ is the vocabulary size. The top $|V|$ most frequent tokens in our data are treated as unique, while all other are converted to an "unknown" token and get the same embedding.

These tokens are fed into the encoder (a single-layer unidirectional LSTM), who looks for the corresponding word representations E_{x_t} in the embedding matrix $E \in \mathbb{R}^{|V| \times d}$ (embedding layer) and then uses them as input to produce a sequence of hidden states $h_1, h_2, \dots, h_T$.

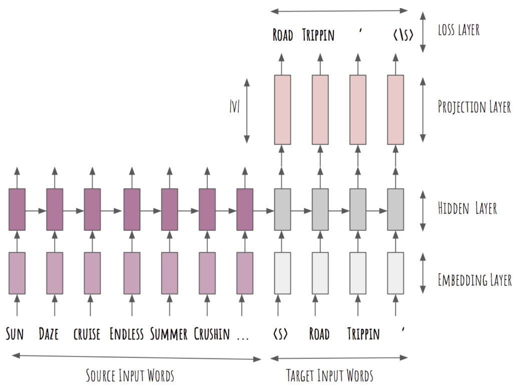

Figure 1: Standard sequence-to-sequence network architecture at train time. During training, the decoder receives as input the sequence of tokens in the target playlist name with an additional start-of-sentence token appended on the right. Here, '<s>' marks the start of the decoding process while '<\ s>' tells the decoder to stop.

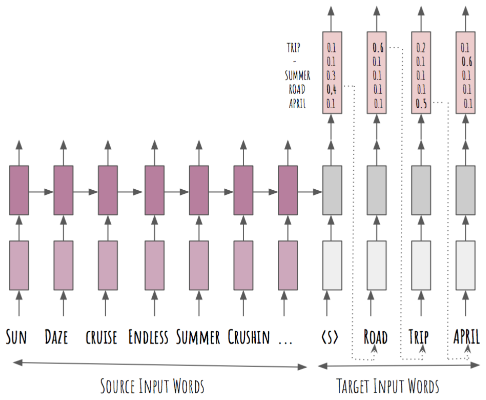

Figure 2: Standard sequence-to-sequence network architecture at test time. Here, the decoder is initially fed a starting symbol to indicate the start of the decoding process; subsequently, its own output is fed as its next step's input. The process continues until the end-of-sentence marker is produced or we reach the maximum number of iterations.

On each step t, the decoder (also a single-layer unidirectional LSTM) receives the last hidden state of the encoder h_t and the word token of the previous word (the previous word of the target playlist title during training, the previous word emitted greedily by the decoder during test time). The decoder then looks up the corresponding embedding and uses it as input along with the encoding of the source tracklist to generate a hidden state s_t, which it then transforms into a logit vector of dimension $|V|$ (projection layer). Finally, the decoder produces a playlist title conditioned on the encoding of its tracklist by finding the maximum logit value of its outputs at each time step.

During training, the loss function is a weighted softmax cross-entropy loss, where the weights are used to mask padding positions outside of the target sequence lengths with values of zero. In particular, at each timestep t,

$$J^{(t)}(\theta) = -\log(P(y_t))$$

where y_t is the target token at that timestep and P is the probability distribution produced by the decoder over all words in the vocabulary; meanwhile, the overall loss for the whole playlist title is given by

$$J(\theta) = \frac{1}{M}\sum_{t=0}^{M} m^{(t)} J^{(t)}(\theta),$$

where $m^{(t)}$ is a masking vector which is 1 for all $t \leq T^*$ and 0 for all $t > T^*$ where T^* is the length of the target sequence and M is the maximum output length.

3.2 Regularized sequence-to-sequence model

This model is identical to the standard sequence-to-sequence network described in Section 3.1, except that we additionally regularize our network by applying Dropout [9] to avoid overfitting. This process amounts to creating an ensemble of smaller networks that share the same weights.

We drop the same network units (inputs, outputs, and recurrent connection) at each time step to avoid an aggregation of the dropout masks that could amplify the noise and drown down the signal of our input, as suggested by Gal et al. [10]. At test time, all neurons are kept to combine all the smaller networks into a single one and the activationas are scaled by the dropout probability to ensure the expected value is the same as the actual output.

3.3 Attentional sequence-to-sequence model

We introduce an attention mechanism to our original sequence-to-sequence network, which relieves the encoder from the burden of having to encode all information in the source sequence into a fixed length vector [7] by allowing the decoder a glimpse into the hidden states of the encoder.

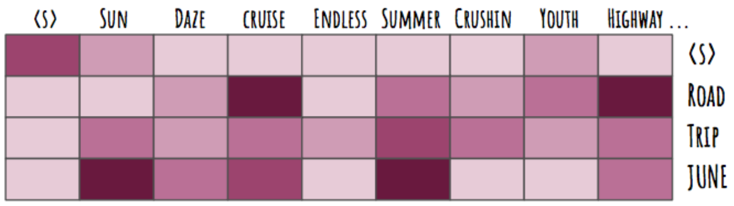

Figure 3: Toy attention visualization. Darker tones denote higher attention score.

The attention distribution is calculated as follows:

$$e_i^{(t)} = v_a^\top \text{tanh}(W_1 h_i + W_2 s_t)$$
$$d^{(t)} = \text{softmax}(e^{(t)}),$$

where v_a, W_1, and W_2 are learnable parameters. This attention distribution is then used to produce the context vector c_t, a linear combination of the source hidden states

$$c_t = \sum_i d_i^{(t)} h_i.$$

Finally, the context vector is combined with the current target hidden state to yield the final attention vector

$$a_t = f(c_t, h_t) = \text{tanh}(W_c[c_t; h_t]),$$

which is fed as an input to the next time step.

We try two variants of this attentional model, one with a unidirectional RNN and the other with a bidirectional RNN for the encoder.

4 Experiments

In this section we: (1) describe the dataset used for our experiments along with some exploratory analysis on coverage, (2) report the choice of final vocabulary used in our model, (3) delineate the model configuration of our experiments, (4) describe the metrics used for performance evaluation, and (5) layout our final results.

4.1 Dataset and Vocabulary

We evaluate our proposed models on the Million Playlist Dataset[1], recently released by Spotify as part of its 2018 RecSys Challenge. It comprises a set of $1,000,000$ playlists that have been created by Spotify users along with a variety of features, including playlist name, description, timestamp when the playlist was last updated, and an array of information about each track in the playlist (track name, artist, album name, duration, and position in the playlist). To simplify our analysis, we work only with playlist names and their corresponding tracklist.

4.1.1 Summary Statistics and Coverage

There are $92,944$ unique titles among our playlists. Tables 1 and 2 show the top playlist titles and tracks with their corresponding count in the full dataset, prior to any processing.

We analyze the word coverage of our dataset to decide the size of our vocabulary. As shown in Table 3, we find that the top $20,000$ most frequent words account for over 98% of all the words in

[1]Official website hosted at https://recsys-challenge.spotify.com

Playlist name	Count (K)
Country	10
Chill	10
Rap	8.5
Workout	8.5
Oldies	8.1
Christmas	8.0
Rock	6.8
Party	6.2
Throwback	5.9
Jams	5.1

Table 1: Top playlist names

Song name and artist	Count (K)
HUMBLE. by Kendrick Lamar	46.6
One Dance by Drake	43.4
Broccoli (feat. Lil Yachty) by DRAM	41.3
Closer by The Chainsmokers	41.1
Congratulations by Post Malone	40.0
Caroline by Amin	35.2
iSpy (feat. Lil Yachty) by KYLE	35.1
Bad and Boujee (feat. Lil Uzi Vert) by Migos	35.0
Location by Khalid	35.0
XO TOUR Llif3 by Lil Uzi Vert	34.9

Table 2: Top tracks

the dataset, including words in the tracklists and playlist names. We hence decide to treat only the top $|V| = 20,000$ of words as unique and convert all other words to an unknown token.

We then filter out playlists with unknown tokens in their title, after which we are left with a total of $885,303$ playlist name-tracklist pairs. We split our dataset into train, validation, and test sets using an $85\% - 15\% - 5\%$ split, which yields in $708,243$ training pairs, $132,795$ validation pairs, and $44,265$ test pairs.

Top k words	**Coverage**	
	# words (M)	%
$1,000$	166.8	73.8
$5,000$	207.6	91.8
$10,000$	217.1	96.0
$20,000$	221.1	98.2
$30,000$	223.7	99.0

Table 3: Word coverage

Top k titles	**Coverage**	
	# playlists (K)	%
1	10.0	1.6
10	77.1	13.0
20	119.6	20.2
50	193.1	32.6
100	255.2	43.1

Table 4: 1-word playlist title coverage

Table 4 shows the playlist coverage of the top 100 playlist names; that is, the percentage of total playlists with one-word names in the top 100 playlist titles. We conclude that there is enough variation in our targets not to treat this problem as a classification task. However, for completeness, we ran a BOW model, a logistic regression, and an LSTM model with an additional soft-max layer to classify the $119,573$ playlists with single-word names in the top 20 titles. These models yielded in accuracies of 0.564, 0.546, and 0.547, respectively. All of these accuracies are significantly higher than the 0.01 accuracy that a majority-vote model (predicting the top title *Country* for all playlists) would have resulted on. We can hence conclude there is enough signal in our input data achieve a good performance on our title generating task.

Finally, the mean number of words our tracklists is 224.9, while the number of words in the playlist titles ranges from 1 to 14, with around 63% of the playlist having single-word titles and over 99% less than 5 words in their title. We will use this information to decide the maximum input and output sequence lengths in our models.

4.2 Experiment Setup

We implemented our model using Tensorflow [11]. For all our experiments, our model has 256-dimensional hidden states and 100-dimensional word embeddings. We tokenize using the PTBTokenizer provided by the Stanford Natural Language Processing Group. We use pretrained GloVe [12] vectors to initialize our word embeddings and treat them as trainable parameters in our model. We train using Adam with learning rate 0.0025. For our standard sequence-to-sequence model, we also try learning rates of 0.001 and 0.005. We use gradient clipping with a maximum gradient norm of 5. In the regularized version of our model, we use drop probability of 0.2. In all our other models we do not use any form of regularization, but we use ROUGE in the validation set to implement early stopping.

During training and at test time we truncate the tracklist length to 200 tokens and limit the playlist name to 6 tokens (including the start- and end-of-sentence tokens for the decoder inputs and targets). We chose these values based on the summary statistics presented in Section 4.1.1.

We train our baseline sequence-to-sequence model and the regularized model for a limit of 25 epochs and our final attentional models for a limit of 30 epochs. Training of our final model took 24 hours on a Tesla M60 GPU with batch size of 64. At test time our playlist titles are produced using Greedy decoding.

4.3 Evaluation Metrics

We evaluate our model reporting the F_1 scores for ROUGE[2]-2, which corresponds to bi-gram overlap between the titles generated by our model and the true labels of our test dataset. We obtain our ROUGE scores using Google's implementation found here [13].

Since this metric could lead to bad scores for creative titles and due to the high subjectivity of this task, we also provide a qualitative analysis of example title names produced by our models.

4.4 Results

In this section we: (1) describe the tuning process that lead to our final learning rate choice, (2) provide a quantitative assessment of our experimental results based on the metric presented in Section 4.3, and (3) lay out a qualitative evaluation of our models through the analysis of the titles generated for the playlists in our test set.

4.4.1 Initial hyper-parameter tuning

We run our standard sequence-to-sequence architecture decribed in Section 3.1 for different values for the learning rate, namely $\eta \in \{0.001, 0.0025, 0.005\}$. Figure 4 shows the corresponding train and validation ROUGE-2 curves.

Figure 4: Train and validation F_1 scores for 2-ROUGE for different learning rates

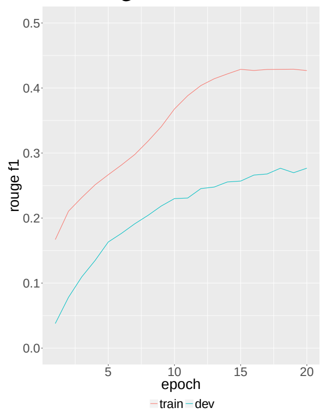

Figure 5: $\eta = 0.001$

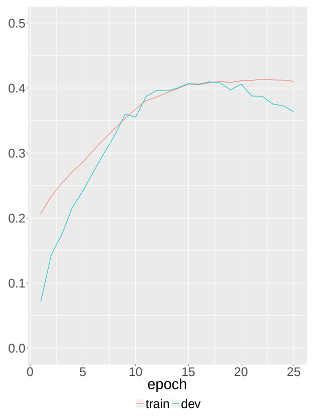

Figure 6: $\eta = 0.0025$

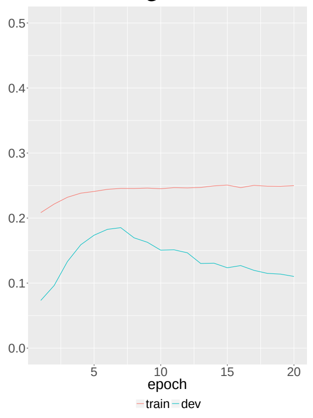

Figure 7: $\eta = 0.005$

We observe that $\eta = 0.005$ is too high, since the training ROUGE plateaus at a low value of approximately 0.25. On the other hand, we see that the training ROUGE increases roughly linearly for the first 15 epochs with $\eta = 0.001$ and then plateaus, which suggests this learning rate is too small. Meanwhile, when $\eta = 0.0025$[3], the shape of the training ROUGE exhibits roughly logarithmic growth.

[2]The acronym ROUGE stands for Recall-Oriented Understudy for Gisting Evaluation.

[3]Note that we ran this model for five extra epochs to allow the train loss to plateau.

Further, the gap between the training and validation ROUGE scores is smallest when $\eta = 0.0025$, which suggests that this model will generalize best out of the three to unseen observations. Based on this analysis, we choose $\eta = 0.0025$ as a learning rate parameter for the rest of our experiments.

4.4.2 Quantitative Results

Our results are shown in Table 5. Our best model scores achieve a 2-ROUGE score of 42.1 on the test set. We observe that adding drop-out does not results in an improvement with respect to our standard model. We note that our network is small relative to our dataset and we are lowering its complexity further by introducing regularizartion. Further, as shown in Figure 6, we were already doing very little overfitting; as a result, regularization hurt our model instead of helping. Hence, we decide not to add dropout in the rest of our models and do early stopping based on 2-ROUGE performance on the validation set instead. Meanwhile, the introduction of an attention mechanism yields in a 4.6 point increase in the test 2-ROUGE score when the encoder is a unidirectional RNN. However, using a bidirectional RNN for the encoder results in a significantly lower ROUGE score, possibly because this model is too complex and has started fitting the error in the training set.

Model	**2-Rouge** F_1
standard seq-to-seq	37.5
standard seq-to-seq + dropout	26.2
standard seq-to-seq + attention	**42.1**
standard seq-to-seq + attention + bidirectional encoder	30.4

Table 5: Rouge F_1 scores on the test set. We note that we have no baselines from previous work to compare our results with, since, to our knowledge, no one has used this dataset for a similar task before. However, for reference, the test ROUGE-2 F_1 scores for the models and baselines for abstactive summarization presented by See et al. [6] range from 11.17 to 17.70. Our best model is the attentional sequence-to-sequence model with an unidirectional RNN for the encoder.

4.4.3 Qualitative results

We start by analyzing the playlists in which our model performs well in terms of ROUGE; that is, test examples where the bi-gram overlap between targets and predictions is high. In particular, we will analyze the examples in which our model predicts exactly the target playlist title.

Figure 8 shows the target playlist titles that our best model predicted exactly on the test set more than 50 times along with the number of times playlists with that title were "classified" correctly.

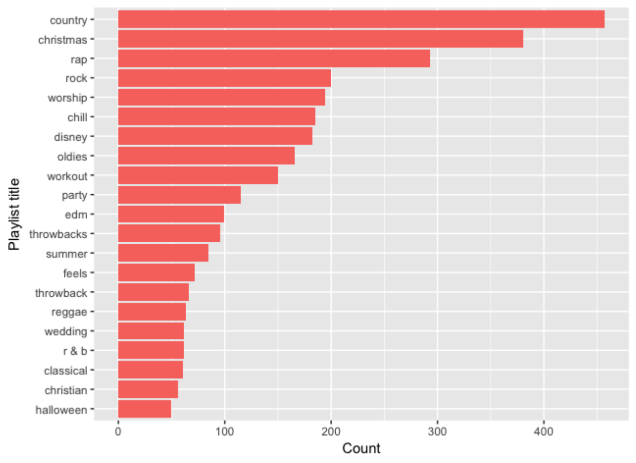

Figure 8: Correct predictions by title

We see that, according to this metric, our model performs particularly well in playlist with titles that are very common in our dataset, such as Country, Christmas, Rap, Rock, Worship, and Chill. As a reference, our model predicts the playlist name exactly right in 11% of our test examples and a substring of the target (or viceversa) in 21.7% of the test examples.

Now we consider playlists examples in which our model commited "errors", if we consider an example with a low ROUGE score an error. In particular, we consider some playlists in which the ROUGE score is zero or very close to zero.

Predicted	**True**
Old Skool	90 's Kid
No Scrubs, No Diggity, Basket Case, Smells Like Teen Spirit, …	
Emo	It's Not a Phase Mom
Sugar We're Going Down, All the Small Things, Welcome to the Black Parade, Misery Business, I Write Sins Not Tragedies, …	
Running	Sweat
Stronger, Sexy and I Know It, Hey Ya, Pump it, Boom Boom Pow …	
Español	Pop Latino
Mi primer millón, Corazón, Te extraño, Bésame, Dulce locura …	
Twerk	Booty
Lemonade, Bedrock, Work, Bad Girlfriend, New Workout Plan, …	
Margaritaville	Summer
All My Friends are Wasted, Fergalicious, Call me Maybe, …	
The Classics of the Days	Dad
Superstition, Son of a Preacher Man, Crossroads, Good Vibrations, …	
Heartache	Break Up Songs
Rolling in the Deep, Just Give Me a Reason, What Hurts the Most, …	

Figure 9: Example titles generated by our best model for playlists in the test set along with a sample of representative songs from their corresponding tracklists.

As shown in Figure 9, even when our model makes mistakes according to our automatic evaluation metric, the playlist titles generated by our model are very similar in meaning to the true title and convey very well the escence of the playlist. In particular, we observe that often our model often predicts a synonym of the original playlist name or a word that captures very well its meaning, such as "oldies" vs. "throwbacks", "christian" vs. "worship", "gym vs. workout", and "yoga music" vs. "meditation".

In general, we observe that our model generates representative titles for most of the examples in the test set; even when our model makes "mistakes" in terms of bi-gram overlap, our playlist title generator predicts names that capture very well the gist of the playlist.

The code for all models, the complete list of predictions on the test set, and the interactive console that generates playlist names given an input tracklist are available online[4].

5 Conclusion

In this project we provided a sequence-to-sequence model for playlist title generation and showed that it produces human-like playlist titles for unseen tracklists. Our main challenge was finding a good metric for evaluating our predictions, given the inherent subjectivity of determining a "good" playlist title. It is still an open question how to automatically evaluate performance on abstractive tasks so partial that not even humans can agree on.

As a next step, we suggest extending our model to a token to character sequence-to-sequence architecture, given the short nature of our outputs. In such model, the encoder would still be in charge of producing a fixed-length encoding of an input sequence of tokens, but the decoder would predict the playlist title one character per output timestep.

[4]`github.com/sofia-samaniego/cs224n-spotify/`

References

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Advances in neural information processing systems*, pp. 3104–3112, 2014.

[2] T. He, J. Chen, L. Ma, Z. Gui, F. Li, W. Shao, and Q. Wang, "Rouge-c: A fully automated evaluation method for multi-document summarization," in *Granular Computing, 2008. GrC 2008. IEEE International Conference on*, pp. 269–274, IEEE, 2008.

[3] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "Text summarization techniques: A brief survey," *arXiv preprint arXiv:1707.02268*, 2017.

[4] A. M. Rush, S. Chopra, and J. Weston, "A neural attention model for abstractive sentence summarization," *arXiv preprint arXiv:1509.00685*, 2015.

[5] R. Nallapati, B. Zhou, C. Gulcehre, B. Xiang, *et al.*, "Abstractive text summarization using sequence-to-sequence rnns and beyond," *arXiv preprint arXiv:1602.06023*, 2016.

[6] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," *arXiv preprint arXiv:1704.04368*, 2017.

[7] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[8] M. Luong, E. Brevdo, and R. Zhao, "Neural machine translation (seq2seq) tutorial," *https://github.com/tensorflow/nmt*, 2017.

[9] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *The Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[10] Y. Gal and Z. Ghahramani, "A theoretically grounded application of dropout in recurrent neural networks," in *Advances in neural information processing systems*, pp. 1019–1027, 2016.

[11] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro, G. S. Corrado, A. Davis, J. Dean, M. Devin, S. Ghemawat, I. Goodfellow, A. Harp, G. Irving, M. Isard, Y. Jia, R. Jozefowicz, L. Kaiser, M. Kudlur, J. Levenberg, D. Mané, R. Monga, S. Moore, D. Murray, C. Olah, M. Schuster, J. Shlens, B. Steiner, I. Sutskever, K. Talwar, P. Tucker, V. Vanhoucke, V. Vasudevan, F. Viégas, O. Vinyals, P. Warden, M. Wattenberg, M. Wicke, Y. Yu, and X. Zheng, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[12] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pp. 1532–1543, 2014.

[13] D. Britz, A. Goldie, T. Luong, and Q. Le, "Massive Exploration of Neural Machine Translation Architectures," *ArXiv e-prints*, Mar. 2017.