MOTIVATIONAL MESSAGE GENERATION USING NLP

Aldo Pioline Antony Charles

1 Motivation

We focused on the task of text generation, with a specific goal of generating motivational and inspiring messages. This is an interesting avenue of work given ongoing research about text generation, and because it adds some level of complexity to the regular text generation, and ensures controlled generation of text messages which is more reliable and fits the users' purpose better. The results of this project can be used for a task like crowdsourced peer-to-peer health promotion messaging, by showing users a sample, auto-generated version of their messages which they can then change to their own preferences and send to their peers to promote physical activity, healthy nutrition and other aspects of healthy living.

2 Dataset

We have used Quote-500K dataset [1] which includes a .csv file containing quotes from multiple sources. Each quote has been tagged with a few topics, such as friendship, love, philosophy, family, etc. and also includes the name of the author (See Figure 1). We discard the author names at this stage, and since we are interested in motivational messages, we filter this data based on the mention of certain tags, including "motivation","positive","inspiration","optimism" and then discard the tags themselves, only keeping the quote. By isolating quotes that include *any* of these tags, we ensure that our data only contains positive and inspiring content. We then clean up the result in a way that we obtain the entire data in a text file, with each line containing a motivational quote, as below:

> *" The magic, grace, growth and wonder of life is that wisdom follows a fall and defeat not from a win. Never underestimate the power of the written word, especially when you put them down in your own voice.*
> *Be strong and free from fear.*
> *Always be true to yourself and follow your dreams!*
>
> *…"*

Quotes-500K does not have any labels other than Author Name and Tags. This dataset was scraped from web sources such as Popular Quotes, Inspirational Quotes at BrainyQuote , Famous Quotes and Authors, Famous Quotations for all Occasions and Curated Quotes - Hand Picked Quotes. The original Quotes-500K dataset includes 500,000 rows (quotes). By keeping only the motivational items, we reduce this number to around 30,000 for each split, train and test. For simplicity, we call this reduced dataset as MotivationalQuotes-500K throughout this report.

Quote	Author	Tags
A friend is someone who knows all about you and still loves you.	Elbert Hubbard	friend, friendship, knowledge, love

Figure 1 - An Example from Quote-500K Dataset

3 Models

Three different models are used in this project, as the base model for our text generation task. Below is an in depth explanation of each model and the considerations we had for our specific purpose.

3.1 Maximum Likelihood Estimator (a.k.a. N-Gram Language Model)

We use a bigram LM as a simple statistical baseline for our task. NLTK package [4] is used to implement the MLE model. NLTK has straightforward commands for performing training and generation using MLE, so we pass our preprocessed and tokenized data to this model and perform generation. In the end we also calculate the perplexity of each sentence and plot the histogram of perplexities for all sentences.

MLE calculates the probability of the next word by performing the following computation:

$$P(w_i | w_{i-1}) = \frac{count(w_{i-1}, w_i)}{\sum_w count(w_{i-1}, w)}$$

To do this, NLTK requires the bigram counts which are provided by passing our tokenized text to the function `padded_everygram_pipeline(2, tokenized_text)` (We also use an NLTK tokenizer to obtain the tokenized text). The vocabulary in this case is the set of all unique tokens (words) in our training corpus. If the model faces any words in the test set that are not part of the training set, it automatically encodes <UNK> in its place. The MLE model uses no numerical representation and takes strings as the main input.
We train this model on our MotivationalQuotes-500K dataset, *train* split. We then generate some example text and also use the *test* split to find perplexity values for each sentence. By using `model.generate(num_words, text_seed="this is a seed text")` we generate as many words as needed. This method uses n-gram probability distribution computed above, to sample words from the vocabulary and find the most likely next word.

3.2 LSTM

Our LSTM baseline has a single hidden LSTM layer with 50 units, and an output layer composed of one neuron for each word in the vocabulary. The model then uses a softmax activation function to ensure the output is normalized to a probability distribution.We use the Keras package to implement the LSTM model [3]. This constructs an embedding layer with embedding size 10, which encodes an input one-hot vector (showing the current word). This embedding vector is then passed into an LSTM layer with 50 cells and the resulting vector is passed into a feed-forward neural net with softmax for final classification of the next word. The softmax layer returns a probability distribution over all words in the vocabulary. The vocabulary of this model is the set of unique training tokens as detected by the tokenizer (we are using a Keras.preprocessing Tokenizer). However, in this case since we are training an embedding layer at the same time, we work with word embeddings instead of strings which makes dealing with unknown

words easier. Any unseen word that is encountered during testing will be assigned to a certain vector that is supposedly 'closer' to certain, semantically related words that exist in the vocabulary. Nonetheless, the embedding layer itself is trained on the same training data and thus will sometimes encounter a token that it does not recognize, in which case the Keras tokenizer discards that token from the sequence. The encoding of an input text is done by Keras's `tokenizer.texts_to_sequences` which uses vocabulary indices to encode strings of text.
For training, we use the MotivationalQuotes-500K *train* split, and utilize a categorical cross entropy loss and an adam optimizer. We train the model for 100 epochs and generate sentences with the trained model. We calculate perplexity by finding the exponential of the cross entropy loss on the generated text. For generation, by using `yhat = model.predict_classes(encoded)` we compute the model output at each step, which returns the class label (index of token in vocabulary) for the next word, which is then looked up in the vocab list to convert to the corresponding word. Unlike MLE which relies on n-grams, our LSTM takes into account the entire history of generation in order to generate the next word, by adding to the input string at each step of generation.

3.3 GPT-2

Generative Pretrained Transformer - 2 (GPT-2) is the second version of OpenAI's transformers which is used for transfer learning in NLP tasks, known to be capable of performing state-of-the-art text generation among many other tasks. This model is implemented using a set of deep feed-forward neural networks, specifically a transformer model with attention in place. Attention mechanisms allow the model to selectively focus on segments of previous text to be the most relevant to the current generation. This model allows for greatly increased parallelization due to the non-sequential nature of feed forward neural nets (able to process 1024 tokens at a time), and has previously shown to work well in generation tasks. The first two layers of GPT-2 perform embedding and positional encoding. The former is trained to obtain word embeddings for input tokens, and the latter is used so that the model could have an understanding of where each token is located (this is necessary because the model is not sequential and takes input in parallel). After that, the encoded input is passed into several *transformer decoders*, each composed of a self-attention module (determining which parts of the text should the model use more or less as cues for a new generation) and a feed-forward network. At each token position, the model will output the next word's hidden state, which can then be used to find the most probable word in the vocabulary that comes next. The model's vocabulary has around 50,000 words and it uses a sub-word level vocabulary called Byte-Pair Encoding. This helps the model understand unknown words by first breaking it into parts as needed until the pieces are found in the vocabulary. We fine tune the GPT-2 model with the MotivationalQuotes-500K *train* split and generate sentences and calculate perplexity for the generated text.

4 Results

All the three models in question performed well in text generation by producing grammatically correct sentences while some generated sentences had their flaws. Evaluating Natural Language

Generation(NLG) results is not a clear venue except for Human Evaluations. Although there exist metrics such as BLEU scores for general Natural Language tasks, NLG does not have a defined metric to test quality of the text produced. We decided to use Perplexity and use the distribution to determine the quality of each model in addition to manually verifying the quality of the generated texts.

MLE uses a straightforward probability distribution as explained previously to predict the next n-gram. A few samples of the generations include: `I am possible when dreams can be dutiful; I am the fire, not apologize;I am stronger through grief and pour himself; second that, shines your dreams, but no safer in such a;`

As we can see, MLE trained on bigrams starts off well by selecting the right bigrams but as the progression goes, the quality of the text deteriorates over time and the later part of the generated text does not match semantically and contextually with the initial phrases. The only advantage MLE has is that it is easier to train and good for basic text generation and we found out that it does not suit the area of inspirational text generation. But surprisingly the mean perplexity (48.37) of MLE was good because perplexity calculates the probability of n grams similar to how MLE does and MLE always tries to maximize the probability and minimize the error similar to perplexity. So using Perplexity alone is not a good approach in focussed text generation problems.

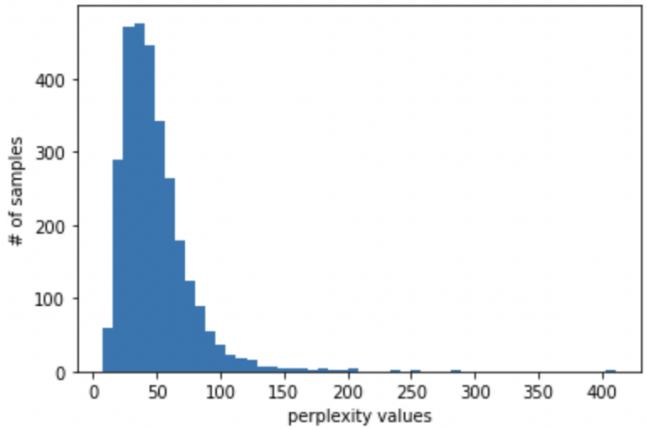

Figure 2: MLE Perplexity distribution

Next, we test a multi-layered LSTM that was trained for ~8 hours over 150k samples and 100 epochs. The generated sentences showcased better coherence than MLE but it was still not sufficient to produce high quality text generations. Some examples are: *winds in life is not the combination of service is what you want to be a real person with thorns; end of the soul you have to endure the answers what you do straight anybody to seek your dreams and; your love to laugh to the lord providing us to celebrate you the truth is the best thing that is;*

The sentences generated have a context embedded in them although the grammatical usage of words are wrong in most cases. The reason might be attributed to the model's training time and

we were able to reach a model accuracy of 40% over 8 hours of training. The perplexity was lower than MLE due to this but as we can see, the sentences had long-term coherent usage of words that needed to be post-processed manually to be made inspirational for real world usage.

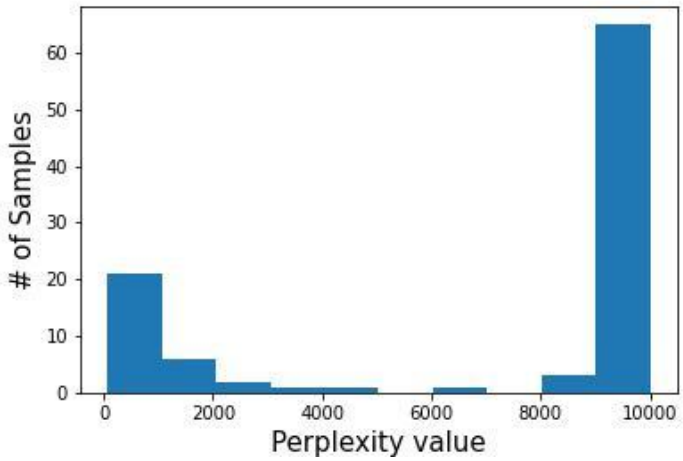

Figure 3: LSTM Perplexity distribution

Finally, we explore GPT-2 from Huggingface's transformers package. We had to tweak the code to suit our needs and after fine tuning the GPT-2 model on our 100k quotes dataset, the results were really good in terms of grammatical usage of words, coherence and originality of texts. Some of the examples are: *Don't forget to appreciate yourself, but to share it with others; Always remember the fact that if your imagination is going to be inspired then that's where it needs to be; To believe, to have faith, and to think, is the only way to be truly and truly happy;*
The texts looked like they were truly written by a human and they require minimal to almost no post-processing when used in real world applications. The perplexity values of most of the generated text were less than 100 which proves the above mentioned traits of a high quality text.

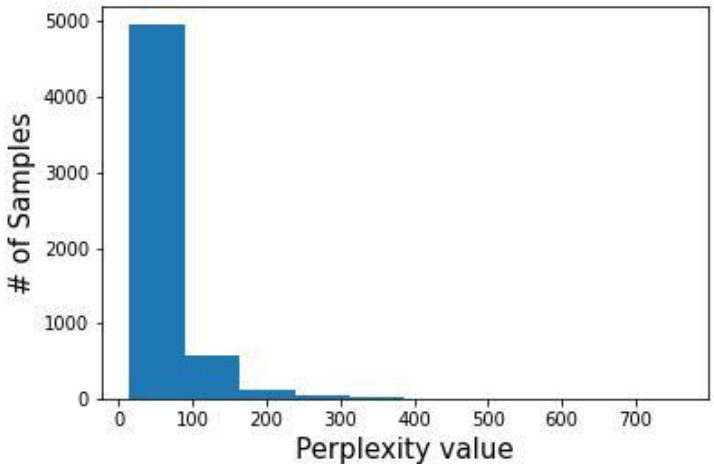

Figure 4: GPT-2 Perplexity distribution

As a bonus, we explored conditioned GPT-2 using PPLM [5] on Bag of words related to motivation and these were the observed results: *Greatness has been made through hard work and determination. But, there is always that small chance of a little bit of luck. And that's what I have for you, today!;Be strong, be kind, be kind to your kids; Never give up! Never get sick.* We used only the pre-trained GPT-2 model but could not get around using the fine-tuned GPT-2 model but it still managed to produce motivational messages like above. This is our future area of research.

5 Conclusion and Future Work

The project gave us a chance to dive deep into how text generation works from a basic probability based model like MLE to models based on deep learning methods like LSTM, GPT-2. With the limited time span, we were able to train fairly good models and there are not many parameters to tweak as we used model architectures that were proven to work well for the use case and GPT-2 performed the best but MLE works well for low computation environments. We were able to accomplish the set goal of generating creative motivational messages through fine tuned GPT-2 and as mentioned in the methods, we explored GPT-2 with PPLM but decided to have the approach as a future work to explore for further refining the generated sentences. We would also want to train LSTM more and see the difference in results and show the produced results to human testers to rate the quality of the generated texts.

6 References

[1] Quotes-500K Dataset: https://github.com/ShivaliGoel/Quotes-500K/blob/master/README.md
[2] Jay Alammar's Blog: https://jalammar.github.io/illustrated-gpt2/
[3] Keras LSTM Documentation https://www.tensorflow.org/api_docs/python/tf/keras/layers/LSTM
[4] NLTK LM Documentation https://www.nltk.org/api/nltk.lm.html

[5] Plug and Play Language Models https://arxiv.org/abs/1912.02164