Automatic Generation of Lyrics with Style of Britpop

Bingying Xia
b6xia@uwaterloo.ca
University of Waterloo
Waterloo, ON, Canada

Abstract

Many of the worldwide hits were created by Britpop Artists, which also led to the development of many other trends of music all over the world. It is interesting to see if algorithms could learn the style of Britpop Music by reading at the lyrics. Character-level recurrent neural networks (RNN), Word-level RNN, and n-gram model are employed for Britpop lyrics generation which are ideally suited to capturing lyrics content and form. With Long-Short-Term-Memory architecture of recurrent neural network and hidden Markov chain, lyrics lines are created by taking into account the previous line and lexical N-gram. Experimental results show that character-level RNN outperforms the other two algorithm using automatic evaluation methods. However, n-gram model achieve better results in manual evaluation part.

Introduction

Many of the world's most famous and iconic bands and artists come from the UK music scene. From Led Zeppelin to Radiohead, the Rolling Stones to Oasis, these are the best music performers of all time. Britpop Music, which produced brighter, catchier alternative rock originated from the 1990s, has had a considerable impact on the music industry across the world. (Bogdanov, Woodstra, and Erlewine 2002; Youngs 2005) Britpop Music encourages innovation, virtuosity, performance, and songwriting by the performers. (Frith and others 2001) The author of this project was influenced by the Britpop Culture long time ago and got inspired by the spirit of freedom and rebellion from the Britpop artists.

In this project, the author is concerned with generating lyrics with Britpop Music style automatically. Algorithms can't create lyrics on their own, but they can analyze lyrics text repositories, extract statistical patterns, maintain them in memory, and generate possible variants with them.

Many research attention has been devoted to lyrics generation and, similarly, poetry generation (see Section 2 for details). Some researchers aim to build an autonomous intelligent system capable of creating meaningful poems which might produce wealth and fortune in entertainment, literature, or even the education industry. (Zhang and Lapata 2014) Although the author don't think the algorithms could replace human beings in art creation area, such as writing lyrics, an assistive environment for lyrics creation could allow musicians and artists to write songs subject to their intention, and enhance their creating experience.

N-gram language generation model and recurrent neural networks are utilized to solve lyrics generation problem in this project. Since Britpop Music is still quite a broad topic, studying it without a particular aim is troublesome. The author chose songs from some representative Britpop Music artists as the study population to train algorithms. The lyrics dataset is collected from Genius (gen 2019b), one of the world's biggest collection of song lyrics and music knowledge. Perplexity and Turing test are the evaluation metrics. Character-level RNN outperforms the other two models in perplexity and average negative log probability. Both two RNN models are able to mimic the form of lyrics. However, n-gram language model performs better in creating meaningful lines though some of the results are copied from origin lyrics.

Here are some key contributions of this project.

- Some hyperparameter settings have been tested in both character-level and word-level RNN and settings with lowest perplexity are selected. It might provide a useful reference to other researchers when they work on lyrics generation using RNN.
- A lyrics dataset about Britpop music has been collected. Researchers who want to work on Britpop culture could easily start their project with this dataset.
- Algorithms achieve better performance in quantitative evaluation methods might be defeated under human judgement. Other than perplexity or average negative log probability, better quantitative evaluation methods should be developed in the future.

Related Work

Lyrics generation is similar to poetry generation. Researchers have been devoting their interests in automatic poetry generation for a long time. The first computational poetry generator can be dated back to 1959 by Theo Lutz (Lutz 1959). A stochastic process was adopted to generate text from only 16 subjects and 16 predicates selected from Kafka's 'The Castle'. Manurung proposed a state goal to the poetry generation problem where a text should satisfies three properties: meaningfulness, grammaticality, and poeticness. (Manurung 2004) Recent work has made significant progress to this goal. Colton et al. described a corpus-based poetry generator which used templates to construct poems according to given constraints on rhyme, meter, stress, sentiment, word frequency, and word similarity. (Colton, Goodwin, and Veale 2012) Genetic algorithms are also utilized to poem generation. (Manurung, Ritchie, and Thompson 2012; Zhou, You, and Ding 2010) It used a sophisticated linguistic formalism to represent its genomic information. However, the experiment shows that it was difficult to produce a text which was both semantically coherent and of high quality metrically.

Human-produced corpora is widely used in computational poetry generation. Agirrezabal et al. extracted the POS-tag sequences from some verse corpora with the help of semantic knowledge base to choose the relevant expression in poems. (Agirrezabal et al. 2013) Netzer et al. developed a system to automatically generate English Haiku poems from a seed word using Word Association Norms generated from corpora. (Netzer et al. 2009). Toivanen et al. employed a corpus-based approach to generate poems with a double-layer structure. The first layer corpus provided semantic content while the second layer one generate a specific grammatical and poetic structure. (Toivanen et al. 2012)

N-gram language model and Markov chain are utilized in language generation with human-produced corpora. Early in 1992, Brown et al. addressed the problem of predicting a word from previous words in a sample of text. (Brown et al. 1992) Barbier et al. extended the basic Markov Chain model to satisfy the local constraints on rhymes and meter. (Barbieri et al. 2012)

Deep learning methods have emerged as a promising method in poetry generation in recent years. Researchers study the poetry generation problem as a sequences modeling problem. Karparthy demonstrated that a recurrent neural network with multi-layer Long-Short Term Memory was capable of generating character-level works from Shakaspeare. (Karpathy 2015) Also, Recurrent neural network with (LSTM) was applied to regularize the generation process to generate Chinese poetry, which resulted in good quality. (Zhang and Lapata 2014; Wang et al. 2016)

Some researchers developed systems to generate song lyrics. Toivanen et al. generated songs with user-specified theme and mood. (Toivanen et al. 2013) They employed Word Association Model to generate lyrics first and then composed the melody according to the syllables, and punctuation of the lyrics. Ramakrishnan et al. presented a lyrics generation system for Tamil language. (Ramakrishnan A and Devi 2010) N-gram based model was used with knowledge-based generation algorithm to generate lyrics that matched the melody. RNN with LSTM was employed to generate lyrics that are similar in style to that of a given rapper. (Potash, Romanov, and Rumshisky 2015) Constrained Markov Processes were used for generation of lyrics in the style of Bob Dylan. (Barbieri et al. 2012)

Methodology

N-Gram

Text generation problem can be solved as creating a probability distribution of word sequences and randomly sampling from it.(Brown et al. 1992)

The probability of a sentence S is:

$$P(S) = P(w_1, w_2, \ldots, w_n) \tag{1}$$

where w_i is the word in the i-th place.

According to the chain rule, we have:

$$P(S) = P(w_1, \ldots, w_n) = \prod_i P(w_i \mid w_1, \ldots, w_{i-1}) \tag{2}$$

If Markov Assumption is applied to simplified the equation 2, then

$$P(w_1, \ldots, w_n) \approx \prod_i P(w_i \mid w_{i-N+1}, \ldots, w_{i-1}) \tag{3}$$

where $N-1$ is the prefix size. The previous $N-1$ words can be used to obtain the N-th one, that's how to obtain a N-gram word sequence through training text.

Maximum likelihood estimate can be used for the estimation.

$$P(w_i \mid w_{i-N+1} \ldots w_{i-1}) = \frac{count(w_{i-N+1}, \ldots, w_i)}{count((w_{i-N+1}, \ldots, w_{i-1})} \tag{4}$$

Recurrent Neural Network and Long-Short-Term-Memory

Recurrent neural network is a machine learning algorithm designed for sequence learning problems. The transitions from previous to current hidden states in RNN is deterministic. (Zaremba, Sutskever, and Vinyals 2014) It can be illustrated as a function:

$$RNN : h_t^{l-1},\ h_{t-1}^l \to h_t^l \tag{5}$$

where $h_t^l \in \mathbb{R}^n$ is a hidden state in layer l in timestep t.

However, Bengio et el. proved that learning long-term dependencies with gradient descent is difficult(Bengio et al. 1994).

Luckily, Long-Short-Term-Memory, a special kind of RNN, is capable of learning long-term dependencies, which is introduced by Hochreiter and Schmidhuber in 1997.(Hochreiter and Schmidhuber 1997) The structure of a normal LSTM is described below(Yao et al. 2014):

- *Forget gate layer*: Decide whether to forget this information
$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f) \tag{6}$$
- *Input gate later*: Decide what new information to update in the cell state
$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i) \tag{7}$$
- *Tanh layer*: Create new candidate values that could be added to the state
$$\widetilde{C_t} = tanh(W_c \cdot [h_{t-1}, x_t] + b_C) \tag{8}$$

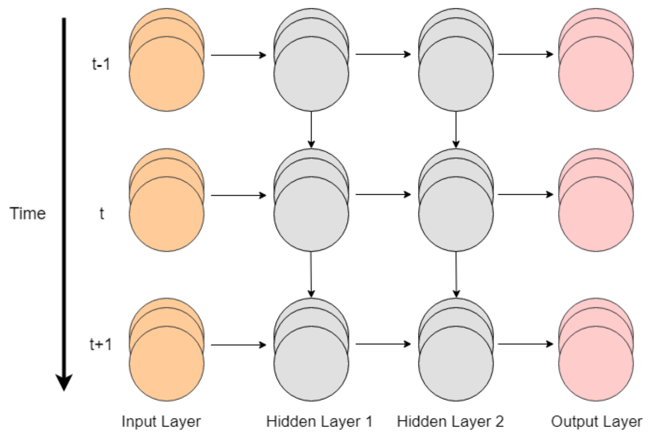

Figure 1: One simple recurrent neural network structure

- *Update C_t*: Update old cell state C_{t-1} into the new cell state C_t
$$C_t = f_t * C_{t-1} + i_t * \widetilde{C_t} \tag{9}$$
- *Sigmoid layer*: Decide what parts of the cell state to output
$$o_t = \sigma(W_o[h_{t-1}, x_t] + b_o) \tag{10}$$
- *Output layer*: Output step
$$h_t = o_t * tanh(C_t) \tag{11}$$

where σ is the logistic sigmoid function, $*$ denotes the element-wise product of the vector.

Language model is one of the most interesting topics in sequence learning questions. Therefore, LSTM is one powerful tool in solving language generation problem. Mikolov et al. claimed that "Neural network based LMs outperform standard backoff n-gram models". (Mikolov et al. 2010).

Experiments

Data

The dataset was collected from Genius (gen 2019b). By using the song names and artist(s)' name, the author used the Genius Api (gen 2019a) to collect the lyrics from certain Britpop singers or bands. Till now there are 782 songs (773 KB) in the dataset, from Radiohead, Oasis, Blur, Suede, and etc. The author collected this dataset because a lyrics generator with Britpop Music style is to be developed in the project. These singers or bands are representatives for Britpop.(top 2019)

A wordcloud is presented in Figure 2. Larger the word is, more important it is in the dataset. It is obvious that `chorus` and `verse` are the most prominence words, which are two major terminologies in song structure. Also, `la la`, `yeah yeah`, and `oh oh` are common modal particles. Therefore, they enjoy eye-catching parts in the wordcloud.

The primary issue of this self-collected dataset is that the size is relatively limited for the neural network model. Also, because the author didn't find any formal definition about Britpop artists, singers and bands selected in this dataset might have a bias.

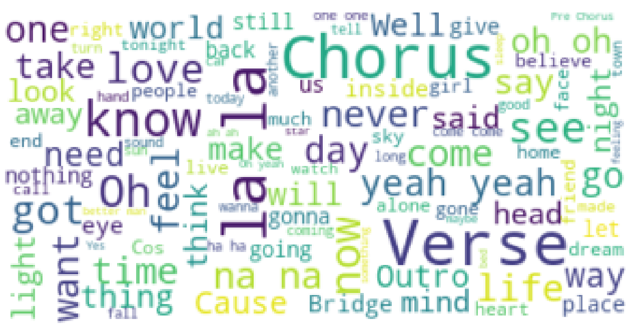

Figure 2: Word Cloud of the lyrics dataset.

Evaluation Metrics

Average Negative Log Probability

The average negative log probability of the target words is defined as:

$$loss = -\frac{1}{N}\sum_{i=1}^{N} \ln p_{target_i} \tag{12}$$

where p_{target_i} is the frequency of the ith word in the generated text.

For RNN models, average negative log probability is employed as the loss function to minimize.

Perplexity The perplexity of a language model on a test set is defined by the inverse probability of the test set, normalized by the number of words.

$$Perp(W) = exp(-\frac{1}{N}\sum_{i=1}^{N} \ln p_{target_i}) = e^{loss} \tag{13}$$

The perplexity can be calculated by exponentiating the average negative log probability. The remarkable fact is that the true model for data has the lowest possible perplexity. The lower the perplexity, the closer to the true model.

For N-gram models, perplexity is utilized to evaluate the model performance.

Turing Test Turing test is also employed to measure the performance of algorithms. 2 example outputs are selected from character-RNN, word-RNN, and N-gram respectively. In total, there are 6 computer-written lyrics. All the 6 examples are listed in Experimental Results. On the other hand, 4 pieces human-created lyrics are quoted from the Genius dataset. These 10 pieces of lyrics are disrupted in order and published in an online survey. 42 volunteers are invited to discriminate whether the lyrics is written by human or algorithms by completing the survey.

RNN Hyperparameters

Hyperparameters is parameters whose values are set before the learning process.(Alemany et al. 2019) The author choose three hyperparameters to tune here. They are the number of neurons, the number of hidden layers, and the max length.

rnn_size	hidden_layer	max_len	loss	perplexity
128	4	5	1.2181	3.3807
		10	1.2038	3.3327
	8	5	1.2092	3.3508
		10	1.1925	3.2953
	12	5	1.2167	3.3760
		10	1.1967	3.3091
256	4	5	1.1946	3.3022
		10	**1.1740**	**3.2349**
	8	5	1.1963	3.3078
		10	15.6109	6.0218+e6
	12	5	15.4741	5.2518+e6
		10	13.8598	1.0452+e6

Table 1: Hyperparameter tuning for word-level RNN with grid search

rnn_size	hidden_layer	max_len	loss	perplexity
128	4	20	0.6823	1.9784
		40	0.6736	1.9613
	12	20	**0.3570**	**1.4290**
		40	0.7117	2.0374
256	4	20	0.3661	1.4421
		40	0.3689	1.4461
	12	20	0.3640	1.4309
		40	0.5465	1.7271

Table 2: Hyperparameter tuning for character-level RNN with grid search

Number of Neurons The number of neurons affects the learning capacity of RNN. Generally, more neurons would be able to learn more complex structure from the dataset at the cost of longer training time. However, more neurons would also lead to overfitting.(Brownlee 2017)

Number of Hidden Layers The number of hidden layers affects the ability to capture the non-linearity of the dataset. Usually, the number of hidden layers is smaller than the number of inputs.(Ligade 2017)

Max Length of Words/Characters Max length of words or characters determines the maximum number of words/characters for the network to use to predict the next word/character, which should be increased to let the network learn longer on character level, or decrease on word level. sequence.(Woolf 2018)

However, hyperparameter tuning for RNN largely depend on the application domian and the context where to apply LSTM neural networks. The activated values of hyperparameters vary from the quantity and quality of the dataset as well. Therefore, the author run a simple grid search for each level of RNN to find the hyperparameter set with best performance. For world-level RNN, the best performance comes with 256 neurons, 4 LSTM hidden layers, and 10 words of max length. For character-level RNN, the best performance appears with 128 neurons, 12 LSTM hidden layers, and 20 characters of max length.

Network Structure and Implementation

Character Level Network Structure For the network architecture on character level, 17 total layers are employed. An input layer, one embedding layer, 12 hidden LSTM layer, one concatenate layer, one Attention layer, and one output layer. The input layer takes in up to 20 characters, converts each character to a 100-D character embedding vector, and feeds those into the following 12 256-cell long-short-term-memory recurrent layers. All the 13 layers are then fed into an Attention layer to weight the most important temporal features and average them together. That output is mapped to probabilities for up to 103 different characters that might be the next character in the sequence, including punctuation and space.

Word Level Network Structure On the word level, 9 total layers are employed. They are an input layer, one embedding layer, 4 hidden LSTM layer, one concatenate layer, one Attention layer, and one output layer. The input layer takes in up to 10 words, converts each word to a 100-D word embedding vector, and feeds those into the following 4 256-cell long-short-term-memory recurrent layers. All the 5 layers are then fed into an Attention layer to weight the most important temporal features and average them as the character level RNN. The output is mapped to probabilities for up to 6919 different words that might be the next word in the lyrics.

Network Implementation `textgenrnn` is utilized for the implementation on the model. `textgenrnn` (Woolf 2017) is a Python 3 module that integrates with deep learning languages such as TensorFlow and Keras. It can create char-rnns and use attention-weighting, decaying learning rate, and skip-embedding to accelerate training and improve network quality. The network is trained on Google Colab, with a GPU of Tesla K80 , having 2496 CUDA cores, compute 3.7, 12GB(11.439GB Usable) GDDR5 VRAM.

Experimental Results

N-gram Here are some sample lyrics generated by the N-gram model.

- Set $N = 2$, the algorithm generated
  ```
  "I will you need the tv and for me
  hear both know where the truth will
  protect you ain't ever makes a screen
  lost myself into the moon shaped pool
  dancing with himself, don't believe"
  ```
  These words don't make much sense because words are generated only with dependency on the previous one.
- Set $N = 3$, the algorithm generated
  ```
  "I go through this neighbourhood I
  don't need to know, you are all this
  way? Have you ever find the words he
  tried to live your life and catching
  my reflection it's always near chasing
  you home you wouldn't say it anyway"
  ```
  These sentences sounds more reasonable.

- Set $N = 4$, the algorithm generated `"I took her to a supermarket. I don't know where to begin walk in the park. You meet an old soldier and talk of the past "` These words might be more meaningful. However, as the N increases, the N-gram algorithm is more likely to copy the original lines from training dataset. For example, `I took her to a supermarket` is actually quoted from *Common People*, a song by **Pulp**.

	Perplexity
$N = 2$	4.872
$N = 3$	4.979
$N = 4$	4.782

Table 3: Perplexity of different N in N-gram model

Because N-gram model reaches the lowest perplexity at $N = 4$, the author selects two examples of the lyrics generated by 4-gram model. The perplexity fluctuates as n increases although larger n leads to fewer choices of the next word. It is interesting to see that N-gram model tend to repeat same lines in its generated results. However, it isn't trapped in an infinite loop.

Sample 1 of 4-gram:

```
[verse 1]
step off the train
he lights a cigarette,
leans back to see the sun
'cause in the end
'cause then you're on your own
forget the lovers you've know
and your friends on your own
forget the lovers you've know
and your friends on your own
'cause you're on your own

[verse 2]
tell me if it's true
that i need holding
```

Sample 2 of 4-gram:

```
[chorus]
the suburban girls
and they're making noise
and they're making eyes at suburban boys
at suburban boys
at suburban boys
at suburban boys
at suburban boys

[verse 3]
watching all the women
shaking slimming walking
down the hall of fame
slowly down the hall
faster than a cannonball
where were you
while we were getting high
we were getting high
we were getting high
```

Word-level RNN Here are two examples of the lyrics generated by word-level RNN. The algorithm knows to add blank space after punctuation, but it fails to get rid of the space before punctuation. Also, word level RNN don't know how to capitalize the beginning word of each sentence. Moreover, there are some naive grammar mistakes in the generated lyrics. For example, `kiss me where the sun don ' t shine`, where `don ' t` should be `doesn't`. In addition, word-level RNN surfers the same problem as the N-gram do that it tends to clip the original lines from the training dataset. For example, `london loves the mystery of a speeding` **`heart`** is twisted from `London loves the mystery of a speeding` **`car`**, *London Love* by **Blur**.

Sample 1 of word-level RNN:

```
[ verse 1 ]
and i ' m just a killer for your love

we ' re getting better man
took the dreams we both know
no , i don ' t mind being on my own
we ride tonight
baby , you ' re driving me crazy
with the whole world staring at you
waltzing on an autobahn
running through my vein
i ' ll be your whole
london loves the mystery
of a speeding heart
we ' re getting better man
is that all i give you ?
but i ' m not sure if it ' ll ever ,
ever , ever work out right
the sun is coming up and it ' s going down
```

Sample 2 of word-level RNN:

```
[ chorus ]
kiss me where the sun don ' t shine
i ' m a hare in the cat ' s eyes
i ' m gonna live alone in bed
droplets of accidents
giving away time to casio
i ' m not so tall
i ' m gonna die alone in bed
```

Character-level RNN Here are another two examples of the lyrics generated by character-level RNN. The algorithm knows to capitalize the first character of the first word of each sentence. The format of the result is close to the general understanding. It is good that since now RNN learns to make prediction on characters, it doesn't tend to clip or copy the original lyrics anymore. The spelling of words are well-learned except that it still fails to use the right auxiliary verb after third pronouns. For example, in `And the wind blow my brain`, `blow` should be `blows`.

Sample 1 of character-level RNN

```
[Chorus]
The sky goes on and on for you and me

I want to crawl all over her
And pour all the love that you keep inside
She's so high

Then I'm gonna know what sorrow means

When you're alone
I got no distance left to run
```

Sample 2 of character-level RNN

```
I'll take my time

I have no choice but to follow you
Pretending that the way's alone
Love is all, love is tall,
love is older than you
And if you could walk away,
where would you go
She knows exactly what she's worth to me
The more I miss the good times, baby,
let it roll again
And I want to be
Where do we go from here?
So why does she feel this way?

And it's not a day too soon
And the wind blow my brain
```

Turing Test Feedback

The results of Turing test study[1] are shown in Figure 3. One bar stands for the feedback of one question. The dark blue/red part accounts for the number of volunteers who vote that lyrics in this question is created by human. Vice versa, the light color part indicates the number of volunteers voting for computer. Every panel represents the results of 2 examples for each method. 4 human-created examples are presented as the benchmark.

Discussion

With respect to perplexity and average negative log probability, character-level RNN displays significant advantage over word-level RNN and N-gram model. Though different parameter settings are tested, it is hard to summarize a general pattern between the change of loss and the change of hyperparameters.

Interestingly, from the feedback of Turing test, the result of N-gram model is most close to the human-written benchmark, followed by the word-level RNN, and character-level RNN earns least votes from volunteers. The reason might be that the grammar and spelling mistakes reveal the computer-generated lyrics. Moreover, N-gram model and word-level RNN both tend to copy or clip the original lines from the dataset. Though semantic associations between different clipped parts might be weak, the results from them are more close to the human-created work.

[1]The test can be reached at `https://forms.gle/vduYXG7xwFJB5nBu8`

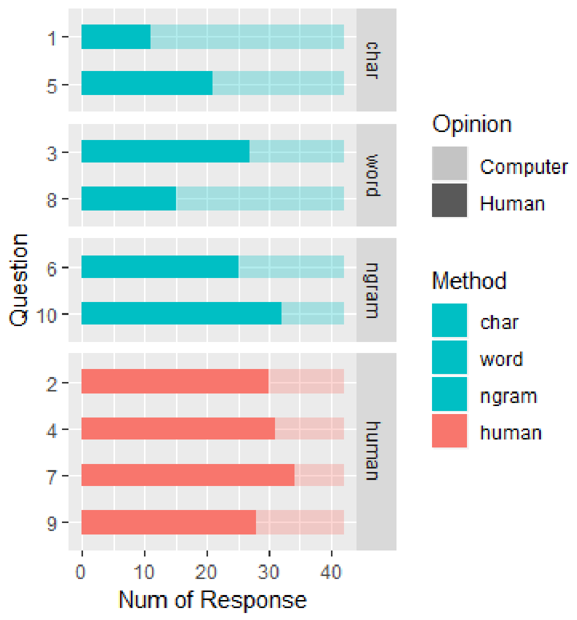

Figure 3: Turing test responses from 42 volunteers

The six lyrics samples above don't totally make sense, but many of the human produced Britpop lyrics are sort of confusing because there might be some insanity inherited in Britpop culture(Simpson 2008). Moreover, these algorithms seems to be good at learning the basic structure of lyrics. The average sentence length is close to common sense. They also tend to learn to respect the relative length of each part song structure, e.g. verse, chorus. For example, there are on average 8 words in each line and about 9 lines of lyrics in each chorus or verse.

However, some limitations are still in this project. One major problem is that the size of the dataset used here is relatively small. Better performance might be achieved with a lager dataset. Another problem is that algorithms are good at 'surface realization'("how to say") but fail to perform 'content selection'("what to say") (Zhang and Lapata 2014). Algorithms are able to learn spell words and mimic general syntactic structures, but they don't know the meaning between words and sentences. Some generated lyrics don't follow a consistent theme. Though word-level embedding could capture semantic information about words, it is hard to maintain control over the semantics of lyrics generation. Basic spelling or grammar mistakes are also key clues to discriminate the computer-created lyrics from human-mades.

Conclusion

In this project the author presents a Britpop lyrics generator based on three text-generating algorithms. They are character-level recurrent neural networks, word-level recurrent neural networks, and N-gram model. In the past researchers usually focused on poetry generation and they had reached many promising achievements. However, not much emphasis has been cast on lyrics generation. In contrast, the author applies three text-generating algorithms popular in poem generation area to lyrics generation problem. Experimental results show that character-level RNN yields high perplexity compared to the other two algorithms. Turing test reveals that N-gram model with $N = 4$ outperforms two RNN algorithms. Character-level RNN lags behind N-gram model under human evaluation. It is worth bearing mind that lyrics writing is a challenging task for human, let alone machines.

Potential future work are many and varied. The most convenient one to accomplish is to collect more lyrics in the original dataset and conduct all the experiments again based on the new dataset. Also, the author would like to apply spelling and grammatical constraints on the algorithm by adding related penalty to the loss function. Finally, more sophisticated network structure might be developed to generate lyrics with consistent theme.

Acknowledgements

I would like to express sincere thanks to Ziqian Zhuang, Yitong Gao, Haoyu Jiang, Xu Liu, Yicheng Zheng, and Yichen Shu for helping distribute the Turing Test survey in the Internet. I am also grateful to the 42 volunteers for participating in the Turing test.

Thanks to Oasis, Radiohead, and Blur. It was a great comfort to listen to their music while I was working on this project.

References

[Agirrezabal et al. 2013] Agirrezabal, M.; Arrieta, B.; Astigarraga, A.; and Hulden, M. 2013. Pos-tag based poetry generation with wordnet. In *Proceedings of the 14th European Workshop on Natural Language Generation*, 162–166.

[Alemany et al. 2019] Alemany, S.; Beltran, J.; Perez, A.; and Ganzfried, S. 2019. Predicting hurricane trajectories using a recurrent neural network. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 33, 468–475.

[Barbieri et al. 2012] Barbieri, G.; Pachet, F.; Roy, P.; and Degli Esposti, M. 2012. Markov constraints for generating lyrics with style. In *Ecai*, volume 242, 115–120.

[Bengio et al. 1994] Bengio, Y.; Simard, P.; Frasconi, P.; et al. 1994. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks* 5(2):157–166.

[Bogdanov, Woodstra, and Erlewine 2002] Bogdanov, V.; Woodstra, C.; and Erlewine, S. T. 2002. *All music guide to rock: the definitive guide to rock, pop, and soul*. Hal Leonard Corporation.

[Brown et al. 1992] Brown, P. F.; Desouza, P. V.; Mercer, R. L.; Pietra, V. J. D.; and Lai, J. C. 1992. Class-based n-gram models of natural language. *Computational linguistics* 18(4):467–479.

[Brownlee 2017] Brownlee, J. 2017. How to tune lstm hyperparameters with keras for time series forecasting. [Online; accessed ¡today¿].

[Colton, Goodwin, and Veale 2012] Colton, S.; Goodwin, J.; and Veale, T. 2012. Full-face poetry generation. In *ICCC*, 95–102.

[Frith and others 2001] Frith, S., et al. 2001. *The Cambridge companion to pop and rock*. Cambridge University Press.

[gen 2019a] 2019a. Genius developer api documantation. `https://docs.genius.com/`. Accessed: 2019-07-15.

[gen 2019b] 2019b. Genius.com. `https://genius.com/`. Accessed: 2019-07-15.

[Hochreiter and Schmidhuber 1997] Hochreiter, S., and Schmidhuber, J. 1997. Long short-term memory. *Neural computation* 9(8):1735–1780.

[Karpathy 2015] Karpathy, A. 2015. The unreasonable effectiveness of recurrent neural networks. *Andrej Karpathy blog* 21.

[Ligade 2017] Ligade, M. 2017. How to select number of hidden layers and number of memory cells in an lstm? [Online; accessed ¡today¿].

[Lutz 1959] Lutz, T. 1959. Stochastische texte. *augenblick* 4(1):3–9.

[Manurung, Ritchie, and Thompson 2012] Manurung, R.; Ritchie, G.; and Thompson, H. 2012. Using genetic algorithms to create meaningful poetic text. *Journal of Experimental & Theoretical Artificial Intelligence* 24(1):43–64.

[Manurung 2004] Manurung, H. 2004. An evolutionary algorithm approach to poetry generation.

[Mikolov et al. 2010] Mikolov, T.; Karafiát, M.; Burget, L.; Černockỳ, J.; and Khudanpur, S. 2010. Recurrent neural network based language model. In *Eleventh annual conference of the international speech communication association*.

[Netzer et al. 2009] Netzer, Y.; Gabay, D.; Goldberg, Y.; and Elhadad, M. 2009. Gaiku: Generating haiku with word associations norms. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, 32–39. Association for Computational Linguistics.

[Potash, Romanov, and Rumshisky 2015] Potash, P.; Romanov, A.; and Rumshisky, A. 2015. Ghostwriter: Using an lstm for automatic rap lyric generation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1919–1924.

[Ramakrishnan A and Devi 2010] Ramakrishnan A, A., and Devi, S. L. 2010. An alternate approach towards meaningful lyric generation in tamil. In *Proceedings of the NAACL HLT 2010 Second Workshop on Computational Approaches to Linguistic Creativity*, 31–39. Association for Computational Linguistics.

[Simpson 2008] Simpson, D. 2008. *The Fallen: Life in and Out of Britain's Most Insane Group*. Canongate Books.

[Toivanen et al. 2012] Toivanen, J.; Toivonen, H.; Valitutti, A.; Gross, O.; et al. 2012. Corpus-based generation of content and form in poetry. In *Proceedings of the third international conference on computational creativity*. University College Dublin.

[Toivanen et al. 2013] Toivanen, J.; Toivonen, H.; Valitutti, A.; et al. 2013. Automatical composition of lyrical songs. In *The Fourth International Conference on Computational Creativity*.

[top 2019] 2019. Top 10 british rock bands from the 90s. `https://www.top10hq.com/top-10-british-rock-bnds-from-the-90s/`. Accessed: 2019-07-15.

[Wang et al. 2016] Wang, Q.; Luo, T.; Wang, D.; and Xing, C. 2016. Chinese song iambics generation with neural attention-based model. *arXiv preprint arXiv:1604.06274*.

[Woolf 2017] Woolf, M. 2017–. textgenrnn. [Online; accessed ¡today¿].

[Woolf 2018] Woolf, M. 2018. How to quickly train a text-generating neural network for free. [Online; accessed ¡today¿].

[Yao et al. 2014] Yao, K.; Peng, B.; Zhang, Y.; Yu, D.; Zweig, G.; and Shi, Y. 2014. Spoken language understanding using long short-term memory neural networks. In *2014 IEEE Spoken Language Technology Workshop (SLT)*, 189–194. IEEE.

[Youngs 2005] Youngs, I. 2005. Looking back at the birth of britpop.

[Zaremba, Sutskever, and Vinyals 2014] Zaremba, W.; Sutskever, I.; and Vinyals, O. 2014. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329*.

[Zhang and Lapata 2014] Zhang, X., and Lapata, M. 2014. Chinese poetry generation with recurrent neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 670–680.

[Zhou, You, and Ding 2010] Zhou, C.-L.; You, W.; and Ding, X. 2010. Genetic algorithm and its implementation of automatic generation of chinese songci. *Journal of Software* 21(3):427–437.