

Automatic Generation of Lyrics in Bob Dylan’s Style

Dongzhuo Li¹, Chao Liang², and Tianze Liu³

¹lidongzh@stanford.edu, ²chao2@stanford.edu, ³tianze@stanford.edu

1 Introduction

Bob Dylan was recently awarded the Nobel Prize “for having created new poetic expressions within the great American song tradition”. It is interesting to see if machine could learn his poetic style by looking at his lyrics. In this project, we use N-grams and Recurrent Neural Network (RNN) with Long Short Term Memory (LSTM) to model Dylan’s lyrics, and eventually use the algorithms to generate samples of lyrics in Bob Dylan’s style.

2 Data Preparation

The data we use includes lyrics of 465 songs downloaded from Bob Dylan’s official website (728KB in total). In order to make it easier for machines to process the data, we pre-processed the data by lowercasing all words and isolating punctuation from words.

3 Methods

3.1 N-grams

The lyrics generation problem can be formulated as creating a probability distribution of word sequences and randomly sampling from it. To model this distribution, we can learn from existing lyrics the probability distributions of the next word conditioning on the previous word sequences. In other words, we want to obtain $P(w_n, w_{n-1}, \dots, w_1)$, where w_i stands for the i -th word in the lyrics document. According to the chain rule of probability, we have:

$$P(w_n, w_{n-1}, \dots, w_1) = \prod_{k=1}^n P(w_k | w_{k-1}, \dots, w_1), \quad (1)$$

where in principal we can derive each conditional distribution $P(w_k | w_{k-1}, \dots, w_1)$ from the training data.

However, it is impractical to obtain each of these distributions down to the first word. A more tractable approach is to only use the previous $N - 1$ words, that is:

$$P(w_n | w_{n-1}, \dots, w_1) \approx P(w_n | w_{n-1}, \dots, w_{n-N+1}). \quad (2)$$

Based on this assumption, we can implement the algorithm by first sliding a window of length N through the training text to obtain a series of short word sequences (N-grams), and then count the time of appearances of each possible last word after each $N - 1$ words in the collection of N-grams. In this way, we acquire an approximation of $P(w_n | w_{n-1}, \dots, w_{n-N+1})$. To generate new lyrics, we can just continuously pick new words based on the conditional distributions.

3.2 Recurrent Neural Network (RNN)

Since it is daunting to provide a rigorous introduction of RNN and LSTM in several pages, we instead state here the main characteristics of RNN and LSTM and give references to readers who want a more rigorous treatment of this subject. Michael Nielsen (2015) and Goodfellow et al. (2016) are two highly recommended books for fundamentals of neural networks.

Compared to traditional neural networks (convolutional networks) that only accept a fixed-sized vector as input and produce a fixed-sized output, RNN accepts a sequence of vectors and outputs a sequence of vectors because RNN allows loops in the architecture, which are simply multiple copies (number of unrolling steps) of the same network each passing a message to a successor. State variables are used to keep track of the context. During the training process, the internal parameters of the neurons (weights, biases, parameters for calculating state variables) are tuned to maximize the probability of the target words (or to minimize the perplexity defined in equation 3). A semi-rigorous introduction to RNN can be found on Andrej Karpathy’s blog (2016).

In particular, LSTM is a special kind of RNN that is capable of learning long-term dependencies, first introduced by Hochreiter and Schmidhuber (1997). Instead of having only a single neural network layers (usually a tanh layer), three gates are used to regulate (adding or removing) the state variable of LSTM, which holds the memory of the LSTM unit. More detailed introduction of LSTM can be found in the exceptional blog of Christopher Olah (2015).

In our project, we use the open source CharRNN code by Chen Liang (2016) on github, which is implemented using Tensorflow. The key parameters are number layers, hidden size (number of LSTM units per layer), batch size, number of steps (number of unrolling steps), number of epochs, learning rate, decay rate (the decay rate of learning rate), dropout (the rate of dropout).

4 Model training and evaluation

4.1 Perplexity: Metrics for Performance Evaluation

We use perplexity as the metrics for performances of N-grams and RNNs, which is defined as (Jurafsky D. and Martin J., 2008)

$$\text{perplexity} = \exp \left(-\frac{1}{N} \sum_{i=1}^N \ln (p_{\text{target}}) \right), \quad (3)$$

where N is the length of the sequence and p_{target} is the probability of the i -th word/character output by N-grams or softmax regression following RNN.

4.2 Training RNN

CharRNN are trained using mini-batch gradient descent method with learning rate decaying with number of epochs, $\alpha(i) = \alpha_0 * (r_d)^t$, where $\alpha_0 = 1$ is the initial learning rate, $r_d = 0.85$ is the decay rate, and $t = 50$ is the number of epochs. 80 % of lyrics are used for training and 10% are used for validation and another 10% are used for testing. Figure 1 shows the training progress during gradient descent steps. As the epoch number increases, RNN starts to output correct words (no miss spelling) in correct grammar (such as capitalizing the first letter of the sentence, correctly using past tense, etc.).

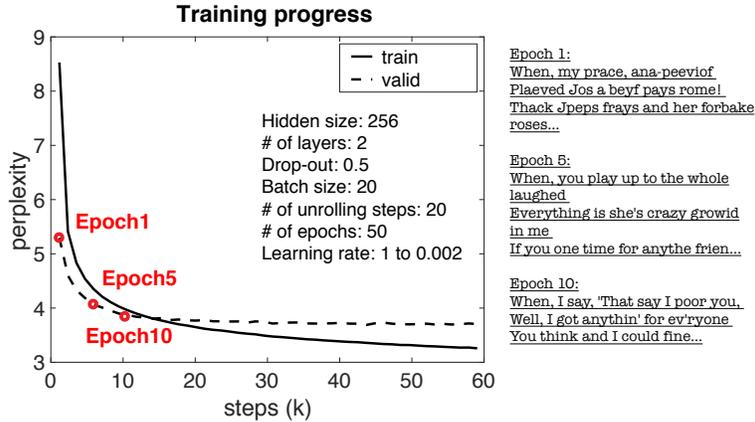


Figure 1: The training process, decrease of both training and validation perplexity as with the increase of gradient descent steps. The right side is showing the sample lyrics after epoch 1,5 and 10, which clearly

4.3 Parameter Studies

In this part, we present studies of three key parameters of RNN: hidden size, batch size and drop out rate. Future works can include the study of more parameters.

1. Hidden size

Figure 2a shows that without drop-out, larger hidden layer size reduces training perplexity but increases validation perplexity, a sign of over-fitting. This behavior of RNN is expected because RNN with larger hidden size has more parameters than RNN with smaller hidden size, which makes the model more likely to over-fit data.

2. Batch size

Figure 2b shows that without drop-out, larger batch size has faster training speed and causes over-fitting earlier in the training process.

3. Dropout rate

Drop-out means randomly disabling a certain fraction of neurons in each iteration in order to avoid over-fitting (Srivastava et al., 2014). Figure 3 shows that a non-zero drop-out rate is very effective in preventing over-fitting.

Table 1: Parameter studies. The base-case set of parameters are indicated by red. Other parameters are the same as those shown in Figure 1. Note that the hidden size is set to 256 for the sensitivity study of drop-out rate.

Batch Size	10	20	50
Hidden Size	64	128	256
Drop-out Rate	0.0	0.25	0.5

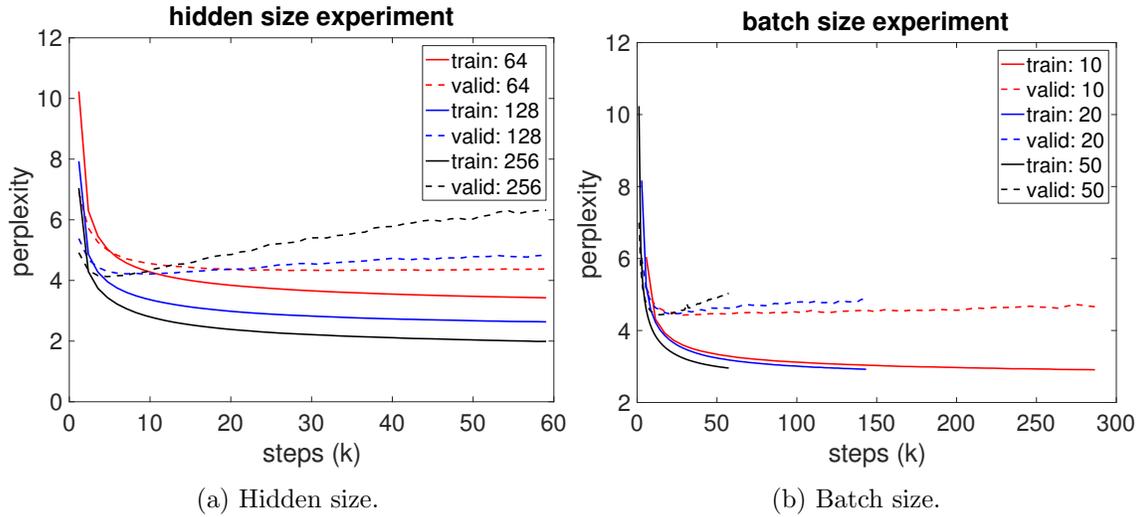


Figure 2: Parameter study on hidden size and batch size.

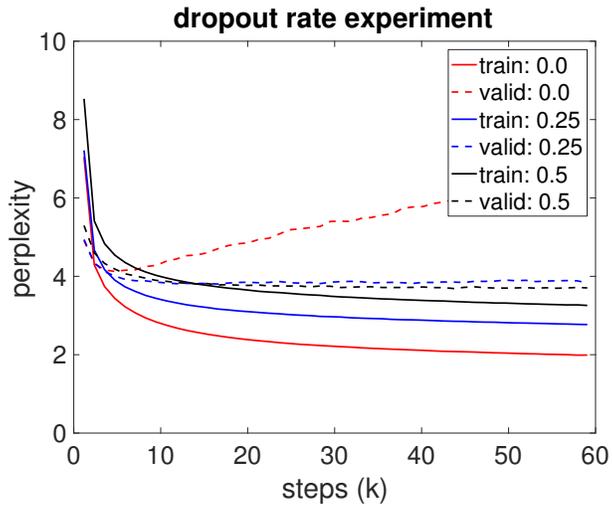


Figure 3: Parameter study on drop-out rate

4.4 Sample Results

1. N-grams

Here are some sample lyrics generated by the N-grams model. First, setting $N = 1$ we have “you weary, shine. road. Hell’s dead fate me stand That a. winds With name?””. These words do not make much sense, because they are generated without any dependence on other ones.

Then, we let $N = 2$ and get “Señor, señor, I saw the webs that ever after makin’ me back what did before. I heard my true love. Say that I’ll see you, yesterday I had enough, what else seems to hate. So no more, no clothes.” These words sound more reasonable.

If we set $N = 3$, the results will become even better. For example, “One by one, until there were none. Two by two, they stepped into the night. Drinkin’ white rum in a pie. Let the bird sing, let the bird fly.” But one also needs to note that as N increases, the algorithm is more likely to copy original lines from the training text, such as the last sentence in this example. In our future work, we will try to select the best N based on it’s performance on testing data.

2. Character-level RNN

Here are two examples of the lyrics generated by Character-level RNN. The spelling of words, grammar and the format of the training text are well learned. For example, the model knows to put a comma after “Well”, and to use “seen” after “be” instead of “see”. Also, the first letter in each sentence is capitalized. “Well, I want to hear my money from the river. Yes, and the next from the ground. With the wind blown of the day” “I am going one. I’m in the moon. Something is the wind. I can’t be seen”

5 Discussion

5.1 N-grams

N-grams have low training perplexity: 32.3 when $N = 2$, and 2 when $N = 3$. The training perplexity decreases as N increases because larger N leads to fewer choices of the next word, hence higher probability to select the right word. On the other hand, the test perplexity is high (589902 approximately). This may be due to the irrelevance between lyrics of different songs and the small size of dataset.

In addition, the generated lyrics make more sense as N increases, but with higher probability of duplication of original lyrics from the training set.

5.2 CharRNN

We find in the experiments that larger hidden size has the risk of causing overfitting, since more parameters are introduced into the model. This can be regularized by using dropout.

Character-level RNN seems good at capturing the grammar(syntax) of the sentences, but may be weak in generating text that makes sense in the context.

One more thing to note is that the size of the dataset we use is relatively small (745KB). To have better performance, we may need to use larger dataset (≥ 1 MB, for example).

References

- Andrej Karpathy. <http://karpathy.github.io/2015/05/21/rnn-effectiveness/>, last visited, 10/20/2016.
- Andrej Karpathy. <https://github.com/crazydonkey200/TensorFlow-char-rnn>. last visited, 10/20/2016.
- Bob Dylan, <http://bobdylan.com>, last visited, 10/20/2016.
- Chen Liang, <https://github.com/crazydonkey200/tensorow-char-rnn>, last visited, 12/12/2016.
- Goodfellow I., Bengio Y., and Courville A. 2016, Deep Learning, Book in preparation for MIT Press, <http://www.deeplearningbook.org>.
- Michael A. Nielsen, Neural Networks and Deep Learning, Determination Press, 2015.
- Jurafsky D. and Martin J., Speech and Language Processing, Prentice Hall, 2008
- Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. Neural Comput. 9, 8 (November 1997), 1735-1780. DOI=<http://dx.doi.org/10.1162/neco.1997.9.8.1735>
- Srivastava, N., Hinton, G. E., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. Journal of Machine Learning Research, 15(1), 1929-1958.