

# DopeLearning: A Computational Approach to Rap Lyrics Generation\*

Eric Malmi  
Aalto University and HIIT  
Espoo, Finland  
eric.malmi@aalto.fi

Pyry Takala  
Aalto University  
Espoo, Finland  
pyry.takala@aalto.fi

Hannu Toivonen  
University of Helsinki and HIIT  
Helsinki, Finland  
hannu.toivonen@cs.helsinki.fi

Tapani Raiko  
Aalto University  
Espoo, Finland  
tapani.raiko@aalto.fi

Aristides Gionis  
Aalto University and HIIT  
Espoo, Finland  
aristides.gionis@aalto.fi

## ABSTRACT

Writing rap lyrics requires both creativity to construct a meaningful, interesting story and lyrical skills to produce complex rhyme patterns, which form the cornerstone of good flow. We present a rap lyrics generation method that captures both of these aspects. First, we develop a prediction model to identify the next line of existing lyrics from a set of candidate next lines. This model is based on two machine-learning techniques: the RankSVM algorithm and a deep neural network model with a novel structure. Results show that the prediction model can identify the true next line among 299 randomly selected lines with an accuracy of 17%, i.e., over 50 times more likely than by random. Second, we employ the prediction model to combine lines from existing songs, producing lyrics with rhyme and a meaning. An evaluation of the produced lyrics shows that in terms of quantitative rhyme density, the method outperforms the best human rappers by 21%. The rap lyrics generator has been deployed as an online tool called DeepBeat, and the performance of the tool has been assessed by analyzing its usage logs. This analysis shows that machine-learned rankings correlate with user preferences.

## 1. INTRODUCTION

Emerging from a hobby of African American youth in the 1970s, rap music has quickly evolved into a mainstream music genre with several artists frequenting Billboard top rankings. Our objective is to study the problem of computational creation of rap lyrics. Our interest in this problem is motivated by two different perspectives. First, we are interested in analyzing the formal structure of rap lyrics and in developing a model that can lead to generating artistic

\*When used as an adjective, *dope* means *cool*, *nice*, or *awesome*.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than the author(s) must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from [permissions@acm.org](mailto:permissions@acm.org).

KDD '16, August 13 - 17, 2016, San Francisco, CA, USA

© 2016 Copyright held by the owner/author(s). Publication rights licensed to ACM. ISBN 978-1-4503-4232-2/16/08...\$15.00

DOI: <http://dx.doi.org/10.1145/2939672.2939679>

work. Second, with the number of smart devices increasing that we use on a daily basis, it is expected that the demand will increase for systems that interact with humans in non-mechanical and pleasant ways.

Rap is distinguished from other music genres by the formal structure present in rap lyrics, which makes the lyrics rhyme well and hence provides better flow to the music. Literature professor Adam Bradley compares rap with popular music and traditional poetry, stating that while popular lyrics lack much of the formal structure of literary verse, rap crafts “intricate structures of sound and rhyme, creating some of the most scrupulously formal poetry composed today” [5].

We approach the problem of lyrics creation from an information-retrieval (IR) perspective. We assume that we have access to a large repository of rap-song lyrics. In this paper, we use a dataset containing over half a million lines from lyrics of 104 different rap artists. We then view the lyrics-generation problem as the task of identifying a *relevant next line*. We consider that a rap song has been partially constructed and treat the first  $m$  lines of the song as a *query*. The IR task is to identify the most relevant next line from a collection of candidate lines, with respect to the query. Following this approach, new lyrics are constructed line by line, combining lyrics from different artists in order to introduce novelty. A key advantage of this approach is that we can evaluate the performance of the generator by measuring how well it predicts existing songs. While conceptually one could approach the lyrics-generation problem by a word-by-word construction, so as to increase novelty, such an approach would require significantly more complex models and we leave it for future work.

Our work lies in the intersection between the areas of *computational creativity* and *information retrieval*. In our approach, we assume that users have a certain concept in their mind, formulated as a sequence of rap lines, and their information need is to find the missing lines, composing a song. Such an information need does not have a factual answer; nevertheless, users will be able to assess the relevance of the response provided by the system. The relevance of the response depends on factors that include rhyming, vocabulary, unexpectedness, semantic coherence, and humor. Tony Veale [26] illustrates other linguistically creative uses of information retrieval, e.g., for metaphor generation. He argues that phrases extracted from large corpora can be used

as “readymade” or “found” objects, like *objets trouvés* in arts, that can take on fresh meanings when used in a new context.

From the computational perspective, a major challenge in generating rap lyrics is to produce semantically coherent lines instead of merely generating complex rhymes. As Paul Edwards [9] puts it: “If an artist takes his or her time to craft phrases that rhyme in intricate ways but still gets across the message of the song, that is usually seen as the mark of a highly skilled MC.”<sup>1</sup> As a result of record results in computer vision, deep neural networks [1] have become a popular tool for feature learning. To avoid hand-crafting a number of semantic and grammatical features, we introduce a deep neural network model that maps sentences into a high-dimensional vector space. This type of vector-space representations have attracted much attention in recent years and have exhibited great empirical performance in tasks requiring semantic analysis of natural language [19, 20].

While some of the features we extract from the analyzed lyrics are tailored for rap lyrics, a similar approach could be applied to generate lyrics for other music genres. Furthermore, the proposed framework could form the basis for several other text-synthesis problems, such as generation of text or conversation responses. Practical extended applications include automation of tasks, such as customer service, sales, or even news reporting.

Our contributions can be summarized as follows:

- (i) We propose an information-retrieval approach to rap lyrics generation. A similar approach could be applied to other tasks requiring text synthesis.
- (ii) We introduce several useful features for predicting the next line of a rap song and hence for generating new lyrics. In particular, we have developed a deep neural network model for capturing the semantic similarity of lines. This feature carries the most predictive power of all the features we have studied.
- (iii) We present *rhyme density*, a measure for the technical quality of rap lyrics. This measure is validated with a human subject, a native-speaking rap artist.
- (iv) We have built an online demo of the rap lyrics generator openly available at [deepbeat.org](http://deepbeat.org). The performance of the algorithm has been assessed based on the usage logs of the demo.

The rest of the paper is organized as follows. We start with a brief discussion of relevant work. Next, we discuss the domain of rap lyrics, introduce the dataset that we have used, and describe how rhymes can be analyzed. We proceed by describing the task of NextLine and our approach to solving it, including the used features, the neural language model, and the results of our experiments. We apply the resulting model to the task of lyrics generation, showing also examples of generated lyrics. The final sections discuss further these tasks, our model, and conclusions of the work.

## 2. RELATED WORK

While the study of human-generated lyrics is of interest to academics in fields such as linguistics and music, artificial rhyme generation is also relevant for various subfields of computer science. Relevant literature can be found under domains of computational creativity, information extrac-

<sup>1</sup>MC, short for master of ceremonies or microphone controller, is essentially a word for a rap artist.

tion and natural language processing. Additionally, relevant methods can be found under the domain of machine learning, for instance, in the emerging field of deep learning.

Hirjee and Brown [12, 13] develop a probabilistic method, inspired by local alignment protein homology detection algorithms, for detecting rap rhymes. Their algorithm obtains a high rhyme detection performance, but it requires a training dataset with labeled rhyme pairs. We introduce a simpler, rule-based approach in Section 3.3, which seemed sufficient for our purposes. Hirjee and Brown obtain a phonetic transcription for the lyrics by applying the CMU Pronouncing Dictionary [16], some hand-crafted rules to handle slang words, and text-to-phoneme rules to handle out-of-vocabulary words, whereas we use an open-source speech synthesizer, eSpeak, to produce the transcription. The computational generation of rap lyrics has been previously studied in [29, 28]. These works adopt a machine-translation approach, whereas we view it as an information-retrieval problem. Furthermore, we have deployed our lyrics generator as an openly accessible web tool to assess its performance in the wild.

Automated creation of rap lyrics can also be viewed as a problem within the research field of Computational Creativity, i.e., study of computational systems which exhibit behaviors deemed to be creative [7]. According to Boden [4], creativity is the ability to come up with ideas or artifacts that are new, surprising, and valuable, and there are three different types of creativity. Our work falls into the class of “combinatorial creativity” where creative results are produced as novel combinations of familiar ideas. Combinatorial approaches have been used to create poetry before but were predominantly based on the idea of copying the grammar from existing poetry and then substituting content words with other ones [25].

In the context of web search, it has been shown that document ranking accuracy can be significantly improved by combining multiple features via machine-learning algorithms instead of using a single static ranking, such as PageRank [21]. This *learning-to-rank* approach is very popular nowadays and many methods have been developed for it [17]. In this paper, we use the RankSVM algorithm [14] for combining different relevance features for the next-line prediction problem, which is the basis of our rap-lyrics generator.

Neural networks have been recently applied to various related tasks. For instance, recurrent neural networks (RNNs) have shown promise in predicting text sequences [11, 23]. Other applications include tasks such as information extraction, information retrieval and indexing [8]. Question answering has also been approached by using deep learning to map questions and answers to a latent semantic space, and then either generating a response [27] or selecting one [30]. Our neural network approach has some similarity to response selection as we also learn a mapping to a hidden space. On the other hand, our network architecture uses a feed-forward net—a suitable choice in our context where sentences are often relatively short and equal in length. Also, generation of lyrics is not typically considered as a response-selection task but we interpret it as one.

## 3. ANATOMY OF RAP LYRICS

In this section, we first describe a typical structure of rap lyrics, as well as different rhyme types that are often used. This information will be the basis for extracting useful fea-

tures for the next-line prediction problem, which we discuss in the next section. Then we introduce a method for automatically detecting rhymes, and we define a measure for rhyme density. We also present experimental results to assess the validity of the rhyme-density measure.

### 3.1 Rhyming

Various different rhyme types, such as *perfect rhyme*, *alliteration*, and *consonance*, are employed in rap lyrics, but the most common rhyme type nowadays, due to its versatility, is the *assonance* rhyme [2, 9]. In a perfect rhyme, the words share exactly the same end sound, as in “slang – gang,” whereas in an assonance rhyme only the vowel sounds are shared. For example, words “crazy” and “baby” have different consonant sounds, but the vowel sounds are the same as can be seen from their phonetic representations “kɹeɪzi” and “beɪbi.”

An assonance rhyme does not have to cover only the end of the rhyming words but it can span multiple words as in the example below (rhyming part is highlighted). This type of assonance rhyme is called *multisyllabic rhyme*.

“This is a job — I get paid to **slang some raps**,

What you made last year was less than my **income tax**” [10]

It is stated in [10] that “[Multisyllabic rhymes] are hallmarks of all the dopest flows, and all the best rappers use them,”

### 3.2 Song structure

A typical rap song follows a pattern of alternating verses and choruses. These in turn consist of lines, which break down to individual words and finally to syllables. A line equals one musical bar, which typically consists of four beats, setting limits to how many syllables can be fit into a single line. Verses, which constitute the main body of a song, are often composed of 16 lines. [9]

Consecutive lines can be joined through rhyme, which is typically placed at the end of the lines but can appear anywhere within the lines. The same end rhyme can be maintained for a couple of lines or even throughout an entire verse. In the verses our algorithm generates, the end rhyme is kept fixed for four consecutive lines unless otherwise specified by the user (see Appendix A for an example).

### 3.3 Automatic rhyme detection

Our aim is to automatically detect multisyllabic assonance rhymes from lyrics given as text. For this, we first obtain a phonetic transcription of the lyrics by applying the text-to-phonemes functionality of an open source speech synthesizer eSpeak.<sup>2</sup> The synthesizer assumes a typical American-English pronunciation. From the phonetic transcription, we can detect rhymes by finding matching vowel phoneme sequences, ignoring consonant phonemes and spaces.

#### 3.3.1 Rhyme density measure

In order to quantify the technical quality of lyrics from a rhyming perspective, we introduce a measure for the *rhyme density* of the lyrics. A simplified description<sup>3</sup> for the computation of this measure is provided below:

1. Compute the phonetic transcription of the lyrics and remove all but vowel phonemes.

<sup>2</sup><http://espeak.sourceforge.net/>

<sup>3</sup>For the details, see: <https://github.com/ekQ/raplysaattori>

**Table 1: A selection of popular rappers and their rhyme densities, i.e., their average rhyme lengths per word.**

Rank	Artist	Rhyme density
1.	Inspectah Deck	1.187
2.	Rakim	1.180
3.	Redrama	1.168
30.	The Notorious B.I.G.	1.059
31.	Lil Wayne	1.056
32.	Nicki Minaj	1.056
33.	2Pac	1.054
39.	Eminem	1.047
40.	Nas	1.043
50.	Jay-Z	1.026
63.	Wu-Tang Clan	1.002
77.	Snoop Dogg	0.967
78.	Dr. Dre	0.966
94.	The Lonely Island	0.870

2. Scan the lyrics word by word.
3. For each word, find the longest matching vowel sequence (=multisyllabic assonance rhyme) in the proximity of the word.
4. Compute the rhyme density by averaging the lengths of the longest matching vowel sequences of all words.

The rhyme density of an artist is computed as the average rhyme density of his or her (or its) songs. Intuitively speaking, rhyme density means the average length of the longest rhyme per word.

#### 3.3.2 Data

We compiled a list of 104 popular English-speaking rap artists and scraped all their songs available on a popular lyrics website. In total, we have 583 669 lines from 10 980 songs.

To make the rhyme densities of different artists comparable, we normalize the lyrics by removing all duplicate lines within a single song, as in some cases the lyrics contain the chorus repeated many times, whereas in other cases they might just have “Chorus 4X,” depending on the user who has provided the lyrics. Some songs, like intro tracks, often contain more regular speech rather than rapping, and hence, we have removed all songs whose title has one of the following words: “intro,” “outro,” “skit,” or “interlude.”

#### 3.3.3 Evaluating human rappers’ rhyming skills

We initially computed the rhyme density for 94 rappers, ranked the artists based on this measure, and published the results online [18]. An excerpt of the results is shown in Table 1.

Some of the results are not too surprising; for instance Rakim, who is ranked second, is known for “his pioneering use of internal rhymes and multisyllabic rhymes.”<sup>4</sup> On the other hand, a limitation of the results is that some artists, like Eminem, who use a lot of multisyllabic rhymes but construct them often by bending words (pronouncing words un-

<sup>4</sup>The Wikipedia article on Rakim <http://en.wikipedia.org/wiki/Rakim> (Accessed: 2016-02-11)

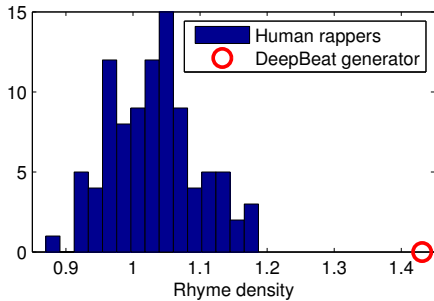


Figure 1: Rhyme density distribution of 105 rappers.

Table 2: List of rap songs ranked by the algorithm and by the artist himself according to how technical he perceives them. Correlation is 0.42.

Rank by artist	Rank by algorithm	Rhyme density
1.	1.	1.542
2.	4.	1.214
3.-4.	9.	0.930
3.-4.	3.	1.492
5.	2.	1.501
6.-7.	10.	0.909
6.-7.	7.	1.047
8.-9.	6.	1.149
8.-9.	5.	1.185
10.	8.	1.009
11.	11.	0.904

usually to make them rhyme), are not as high on the list as one might expect.

In Figure 1, we show the distribution of rhyme densities along with the rhyme density obtained by our lyrics generator algorithm DeepBeat (see Section 5.3 for details).

### 3.4 Validating the rhyme density measure

After we had published the results shown in Table 1 online, a rap artist called *Ahmen* contacted us, asking us to compute the rhyme density for the lyrics of his debut album. Before revealing the rhyme densities of the 11 individual songs he sent us, we asked the rapper to rank his own lyrics “starting from the most technical according to where you think you have used the most and the longest rhymes.” The rankings produced by the artist and by the algorithm are shown in Table 2.

We can compute the correlation between the artist produced and algorithm produced rankings by applying the Kendall tau rank correlation coefficient. Assuming that all ties indicated by the artist are decided unfavorably for the algorithm, the correlation between the two rankings would still be 0.42, and the null hypothesis of the rankings being independent can be rejected ( $p < 0.05$ ). This suggests that the rhyme density measure adequately captures the technical quality of rap lyrics.

## 4. NEXT LINE PREDICTION

We approach the problem of rap lyrics generation as an information-retrieval task. In short, our approach is as fol-

lows: we consider a large repository of rap lyrics, which we treat as training data, and which we use to learn a model between consecutive lines in rap lyrics. Then, given a set of seed lines we can use our model to identify the best next line among a set of candidate next lines taken from the lyrics repository. The method can then be used to construct a song line-by-line, appending relevant lines from different songs.

In order to evaluate this information-retrieval task, we define the problem of “next-line prediction.”

### 4.1 Problem definition

As mentioned above, in the core of our algorithm for generating rap lyrics is the task of finding the next line of a given song. This next line prediction problem is defined as follows.

PROBLEM 1. (*NextLine*) Consider the lyrics of a rap song  $S$ , which is a sequence of  $n$  lines  $(s_1, \dots, s_n)$ . Assume that the first  $m$  lines of  $S$ , denoted by  $B = (s_1, \dots, s_m)$ , are known and are considered as “the query.” Consider also that we are given a set of  $k$  candidate next lines  $C = \{\ell_1, \dots, \ell_k\}$ , and the promise that  $s_{m+1} \in C$ . The goal is to identify  $s_{m+1}$  in the candidate set  $C$ , i.e., pick a line  $\ell_i \in C$  such that  $\ell_i = s_{m+1}$ .

Our approach to solving the NextLine problem is to compute a relevance score between the query  $B$  and each candidate line  $\ell \in C$ , and return the line that maximizes this relevance score. The performance of the method can be evaluated using standard information retrieval measures, such as mean reciprocal rank. As the relevance score we use a linear model over a set of similarity features between the query song-prefix  $B$  and the candidate next lines  $\ell \in C$ . The weights of the linear model are learned using the RankSVM algorithm, described in Section 4.3.2.

In the next section, we describe a set of features that we use for measuring the similarity between the previous lines  $B$  and a candidate next line  $\ell$ .

### 4.2 Feature extraction

The similarity features we use for the next-line prediction problem can be divided into three groups, capturing (i) *rhyming*, (ii) *structural similarity*, and (iii) *semantic similarity*.

(i) We extract three different rhyme features based on the phonetic transcription of the lines, as discussed in Section 3.3.

**EndRhyme** is the number of matching vowel phonemes at the end of lines  $\ell$  and  $s_m$ , i.e., the last line of  $B$ . Spaces and consonant phonemes are ignored, so for instance, the following two phrases would have three end rhyme vowels in common.

Line	Phonetic transcription
pay for	peɪ fɔːr
stay warm	steɪ wɔːrɪm

**EndRhyme-1** is the number of matching vowel phonemes at the end of lines  $\ell$  and  $s_{m-1}$ , i.e., the line before the last in  $B$ . This feature captures alternating rhyme schemes of the form “*abab*.”

**OtherRhyme** is the average number of matching vowel phonemes per word. For each word in  $\ell$ , we find the longest matching vowel sequence in  $s_m$  and average the lengths of these sequences. This captures other than end rhymes.

(ii) With respect to the structural similarity between lines, we extract one feature measuring the difference of the lengths of the lines.

**LineLength.** Typically, consecutive lines are roughly the same length since they need to be spoken out within one musical bar of a fixed length. The length similarity of two lines  $\ell$  and  $s$  is computed as

$$1 - \frac{|\text{len}(\ell) - \text{len}(s)|}{\max(\text{len}(\ell), \text{len}(s))},$$

where  $\text{len}(\cdot)$  is the function that returns the number of characters in a line.<sup>5</sup> We compute the length similarity between a candidate line  $\ell$  and the last line  $s_m$  of the song prefix  $B$ .

(iii) Finally, for measuring semantic similarity between lines, we employ four different features.

**BOW.** First, we tokenize the lines and represent each last line  $s_m$  of  $B$  as a bag of words  $S_m$ . We apply the same procedure and obtain a bag of words  $L$  for a candidate line  $\ell$ . We then measure semantic similarity between two lines by computing the Jaccard similarity between the corresponding bags of words

$$\frac{|S_m \cap L|}{|S_m \cup L|}.$$

**BOW5.** Instead of extracting a bag-of-words representation from only the last line  $s_m$  of  $B$ , we use the  $k$  previous lines.

$$\frac{\left| \left( \bigcup_{j=m-k}^m S_j \right) \cap L \right|}{\left| \left( \bigcup_{j=m-k}^m S_j \right) \cup L \right|}.$$

In this way we can incorporate a longer context that could be relevant with the next line that we want to identify. We have experimented with various values of  $k$ , and we found out that using the  $k = 5$  previous lines gives the best results.

**LSA.** Bag-of-word models are not able to cope with synonymy nor polysemy. To enhance our model with such capabilities of use a simple latent semantic analysis (LSA) approach. As a preprocessing step, we remove stop words and words that appear less than three times. Then we use our training data to form a line-term matrix and we compute a rank-100 approximation of this matrix. Each line is represented by a term vector and is projected on the low-rank matrix space. The LSA similarity between two lines is computed as the cosine similarity of their projected vectors.

**NN5.** Our last semantic feature is based on a neural language model. It is described in more detail in the next section.

### 4.3 Methods

In this section we present two building blocks of our method: the neural language model used to incorporate semantic similarity with the NN5 feature and the RankSVM method used to combine the various features.

#### 4.3.1 Neural language model

Our knowledge of all possible features can never be comprehensive, and designing a solid extractor for features such as semantic or grammatical similarity would be extremely time-consuming. Thus, attempting to learn additional features appears a promising approach. Since neural networks

<sup>5</sup>We also tested the number of syllables in a line but the results were similar.

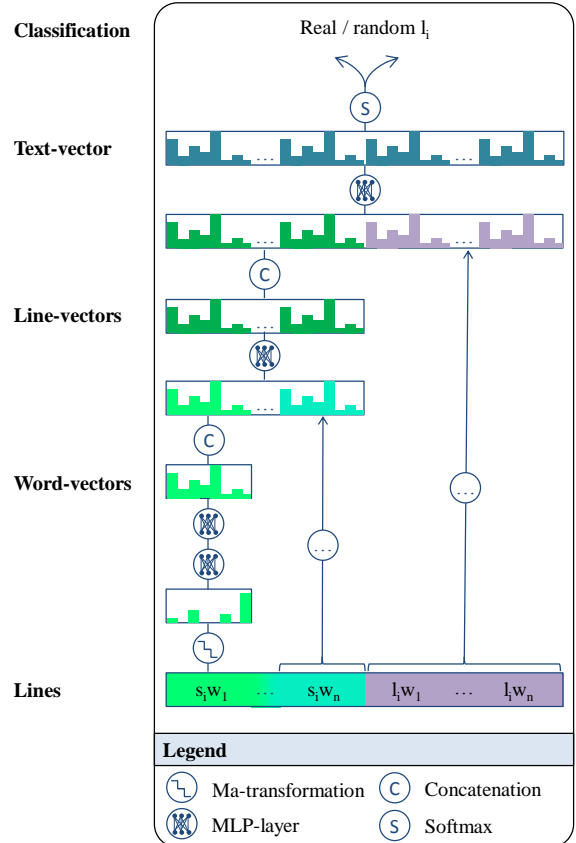


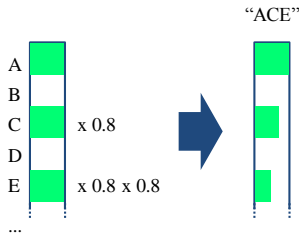
Figure 2: Network architecture

can have a vast number of available parameters and can learn complex nonlinear mappings, we experiment with whether they could be used to automatically extract relevant features. To this end, we design a neural network that learns to use raw text sequences to predict the relevance of a candidate next line given previous lines.

In brief, our neural network starts by finding distributed (vector) representations for words. These are combined to distributed representations of lines, and further combined to vector representations of multiple lines, where the last line may be a real line, or a randomly sampled line. Based on this vector representation of text, our network learns to predict whether it believes that the last line is a suitable next line candidate or not.

**Network architecture.** At the core of our predictor, we use multi-layered, fully-connected neural networks, trained with backpropagation. While the network structure was originally inspired by the work of Collobert et al. [6], our model differs substantially from theirs. In particular, we have included new input transformations and our input consists of multiple sentences.

The structure of our model is illustrated in Figure 2. We start by preprocessing all text to a format more easily handled by a neural network, removing most non-ascii characters and one-word lines, and stemming and lower-casing all words. Our choice of neural network architecture requires



**Figure 3: Simplified example of exponential moving-average transformation**

fixed line lengths, so we also remove words exceeding 13 words, and pad shorter lines. We build samples in the format: “candidate next line ( $\ell_i$ )”; “previous lines ( $s_1, \dots, s_m$ ).” For lines with too few previous lines, we add paddings.

The first layers of the network are word-specific: they perform the same transformation for each word in a text, regardless of its position. For each word, we start with an exponential moving average transformation that turns a character sequence into a vector. A simplified example of this is shown in Figure 3.<sup>6</sup> We use this transformation as it is, compared to one-hot encoding, more robust to different spellings, coping well for instance with slang and spelling errors.

The transformation works as follows: we create a zero-vector, with the length of possible characters. Starting with the first character in a word, we choose its corresponding location in the vector. We increment this value with a value proportional to the character’s position in the word, counting from the beginning of the word. We get a word-representation  $\mathbf{w} = (w_a w_b \dots w_z \dots)^T$ , where

$$w_a = \sum \frac{(1 - \alpha)^{c_a}}{Z},$$

where  $c$  is the index of the character at hand,  $\alpha$  is a decay hyperparameter, and  $Z$  is a normalizer proportional to word length. For further details of this transformation, see [24].

To avoid always giving larger weight to the beginning of a word, we also concatenate to the vector this transformation backwards, and a vector of character counts in the word. Following this transformation, we feed each word vector to a fully-connected neural network.

Our word-specific vectors are next concatenated in the word order to a single line-vector, and fed to a line-level network. Next, the line-vectors are concatenated, so that the candidate next-line is placed first, and preceding lines are then presented in their order of occurrence. This vector is fed to the final layer, and the output is fed to a softmax-function that outputs a binary prediction indicating whether it believes one line follows the next or not. In our ensemble model, we use the activation before the softmax, corresponding to the confidence the network has in a line being the next in the lyrics.

**Training.** For training the neural network, we retrieve a list of previous lines for each of the lyrics lines. These lyrics are

<sup>6</sup>While we achieve our best results using this transformation, a traditional ‘one-hot’ vector approach yields nearly as good results. A model that learns the transformation, such as a recurrent neural network could also be tested in future work.

fed to the neural model in small batches, and a corresponding gradient descent update is performed on the weights of the network. To give our network negative examples, we follow an approach similar to [6], and generate fake line examples by choosing for every second example the candidate line uniformly at random from all lyrics.

A set of technical choices is necessary for constructing and training our network. We use two word-specific neural layers (500 neurons each), one line-specific layer (256 neurons), and one final layer (256 neurons). All layers have rectified linear units as the activation function. Our minibatch size is 10, we use the adaptive learning rate Adadelta [31], and we regularize the network with 10% dropout [22]. We train the network for 20 epochs on a GPU machine, taking advantage of the Theano library [3]. Analyzing hyperparameters, we find that we can improve results especially by going from one previous line to a moderately large context (5 previous lines), and by using a larger input window, i.e. giving the network more words from each line.

**Manual evaluation.** To get some understanding of what our neural network has learnt, we manually analyze 25 random sentences where a random candidate line was falsely classified as being the real next line. For 13 of the lines, we find it difficult to distinguish whether the real or the random line should follow. This often relates to the rapper changing the topic, especially when moving from one verse to another. For the remaining 12 lines, we notice that the neural network has not succeeded in identifying five instances with a rhyme, three instances with repeating words, two instances with consistent sentence styles (lines having always e.g., one sentence per line), one instance with consistent grammar (a sentence continues to next line), and one instance where the pronunciation of words is very similar the previous and the next line. In order to detect the rhymes, the network would need to learn a phonetic representation of the words, but for this problem we have developed other features presented in Section 4.2.

### 4.3.2 Ranking candidate lines

We take existing rap lyrics as the ground truth for the next line prediction problem. The lyrics are transformed into preferences

$$s_{m+1} \succ_B \ell_i,$$

which are read as “ $s_{m+1}$  (the true next line) is preferred over  $\ell_i$  (a randomly sampled line from the set of all available lines) in the context of  $B$  (the preceding lines)”. When sampling  $\ell_i$  we ensure that  $\ell_i \neq s_{m+1}$ . Then we extract features  $\Phi(B, \ell)$  listed in Section 4.2 to describe the match between the preceding lines and a candidate next line.

The RankSVM algorithm [14] takes this type of data as input and tries to learn a linear model which gives relevance scores for the candidate lines. These relevance scores should respect the preference relations given as the training data for the algorithm. The relevance scores are given by

$$r(B, \ell) = \mathbf{w}^T \Phi(B, \ell), \quad (1)$$

where  $\mathbf{w}$  is a weight vector of the same length as the number of features.

The advantage of having a simple linear model is that it can be trained very efficiently. We employ the SVM<sup>rank</sup> software for the learning [15]. Furthermore, we can interpret the weights as importance values for the features. Therefore,

**Table 3: Next line prediction results for  $k = 300$  candidate lines. MRR stands for mean reciprocal rank and Rec@N for recall when retrieving top  $N$  lines.**

Feature(s)	Mean rank	MRR	Rec@1	Rec@5	Rec@30	Rec@150
Random	150.5	0.021	0.003	0.017	0.010	0.500
LineLength	117.6	0.030	0.002	0.029	0.177	0.657
EndRhyme	103.2	<b>0.140</b>	<b>0.077</b>	<b>0.181</b>	<b>0.344</b>	0.480
EndRhyme-1	126.0	0.075	0.037	0.092	0.205	0.347
OtherRhyme	123.3	0.047	0.016	0.055	0.190	0.604
BOW	112.4	0.116	0.074	0.138	0.280	0.516
BOW5	99.1	0.110	0.065	0.129	0.314	0.708
LSA	111.3	0.089	0.051	0.107	0.262	0.662
NN5	<b>84.7</b>	0.067	0.020	0.083	0.319	<b>0.793</b>
FastFeats	73.5	0.224	0.160	0.272	0.476	0.802
FastFeatsNN5	61.2	<b>0.244</b>	<b>0.172</b>	<b>0.306</b>	0.524	0.853
<b>All features</b>	<b>60.8</b>	0.243	0.169	0.304	<b>0.527</b>	<b>0.855</b>

when the model is later applied to lyrics generation, the user could manually adjust the weights, if he or she, for instance, prefers to have lyrics where the end rhyme plays a larger/smaller role.

#### 4.4 Empirical next line prediction evaluation

Our experimental setup is the following. We split the lyrics by artist randomly into training (50%), validation (25%), and test (25%). The RankSVM model and the neural network model are learned using the training set, while varying a trade-off parameter  $C$  among  $\{1, 10^1, 10^2, 10^3, 10^4, 10^5\}$  to find the value which maximizes the performance on the validation set. The final performance is evaluated on the unseen test set.

We use a candidate set containing the true next line and 299 randomly chosen lines. The performance is measured using mean rank, mean reciprocal rank (reciprocal value of the harmonic mean of ranks), and recall at  $N$  where the value of  $N$  is varied. The results are computed based on a random sample of 10 000 queries.

Table 3 shows the test performance for different feature sets. Each feature alone yields a mean rank of  $< 150$  and thus carries some predictive power. The best individual feature with respect to the mean rank is the output of the neural network model. However, if we look at recall at 1 (the probability to rank the true next line first), the best performance, 7.7%, is obtained by EndRhyme which works very well in some cases but is in overall inferior to NN5 since in some cases consecutive lines do not rhyme at all. Combining all features, we achieve a mean rank of 60.8 and can pick the true next line with 16.9% accuracy. The probability to pick the true next line at random is only 0.3%. The features are combined by taking a linear combination of their values according to Equation (1).

To enable the generation of rap lyrics in real time, we also tested employing only the features which are fast to evaluate, i.e.,  $\text{FastFeats} = \text{LineLength} + \text{EndRhyme} + \text{EndRhyme-1} + \text{BOW} + \text{BOW5}$ , and  $\text{FastFeatsNN5} = \text{FastFeats} + \text{NN5}$ . With the latter feature set, the performance is almost identical to using the full feature set and even FastFeats works relatively well.

---

#### Algorithm 1: Lyrics generation algorithm DeepBeat.

---

**Input:** Seed line  $\ell_1$ , length of the lyrics  $n$ .  
**Output:** Lyrics  $L = (\ell_1, \ell_2, \dots, \ell_n)$ .  
 $L[1] \leftarrow \ell_1$ ; // Initialize a list of lines.  
**for**  $i \leftarrow 2$  **to**  $n$  **do**  
     $C \leftarrow \text{retrieve\_candidates}(L[i-1])$ ; // Sec. 5.2.1  
     $\hat{c} \leftarrow \text{NaN}$ ;  
    **foreach**  $c \in C$  **do**  
        /\* Check relevance and feasibility of the candidate. \*/  
        **if**  $\text{rel}(c, L) > \text{rel}(\hat{c}, L) \ \& \ \text{rhyme\_ok}(c, L)$  **then**  
             $\hat{c} \leftarrow c$ ;  
     $L[i] \leftarrow \hat{c}$ ;  
**return**  $L$ ;

---

## 5. LYRICS GENERATION

### 5.1 Overview

The lyrics generation is based on the idea of selecting the most relevant line given the previous lines, which is repeated until the whole lyrics have been generated. Our algorithm DeepBeat is summarized in Algorithm 1.

The relevance scores for candidate lines are computed using the RankSVM algorithm described in Section 4. Instead of selecting the candidate with strictly the highest relevance score, we filter some candidates according to  $\text{rhyme\_ok}()$  function. It checks that consecutive lines are not from the same song and that they do not end with the same words. Although rhyming the same words can be observed in some contemporary rap lyrics, it has not always been considered a valid technique [9]. While it is straightforward to produce long “rhymes” by repeating the same phrases, it can lead to uninteresting lyrics.

### 5.2 Online demo

An online demo for the lyrics generation algorithm is available at [deepbeat.org](http://deepbeat.org). This web tool was built in order to (1) make the generator available to the public, (2) provide the users easy ways of customizing the generated lyrics, and (3) collect limited usage logs in order to evaluate and improve the algorithm. The website was launched in November 2015 and as of June 2016 it has been visited by more than 42 000 users.

After the initial launch of the project, we started collaborating with some musicians to record the first songs written by DeepBeat,<sup>7</sup> which showed us the importance of giving the user sufficient customization capabilities instead of merely outputting complete lyrics. With this in mind, we designed the online demo so that users can: (1) define keywords that must appear in the generated lyrics; (2) ask the algorithm to give suggestions for the next line and pick the best suggestion manually; and (3) write some of the lines by themselves. A user can, for example, write the first line by herself and let the algorithm generate the remaining lines. An interesting mode of usage we noticed some users adopting is to write every other line by yourself and generate every other.

<sup>7</sup>The first music video is available at: <https://youtu.be/Js0HYmH31ko>  
More about the collaboration can be read at: <https://howwegettonext.com/deepbeat-what-happens-when-a-robot-writes-rhymes-for-rappers-77d07c406ff5>

When the user generates lyrics line-by-line, asking for suggestions from DeepBeat, we log the selected lines. In Section 5.3.2, we show how to evaluate the algorithm using the log data. Conveniently, we can also use the logs to refine the learned models employing the RankSVM approach as done in [14].

### 5.2.1 Performance optimization

**Candidate set retrieval.** Results in Table 3 show that EndRhyme alone is a good predictor. Therefore, we define the set of candidate next lines as the 300 best rhyming lines instead of 300 random lines. This candidate set has to be retrieved quickly without evaluating each of the  $n_l = 583\,669$  lines in our database. Conveniently, this can be formulated as the problem of finding  $k$  strings with the longest common prefix with respect to a query string, as follows

1. Compute a phonetic transcription for each line.
2. Remove all but vowel phonemes.
3. Reverse phoneme strings.

We solve the longest common prefix problem by first, sorting the phoneme string as a preprocessing step, second, employing binary search<sup>8</sup> to find the line with the longest common prefix ( $\ell_{\text{long}}$ ), and third, taking the  $k - 1$  lines with the longest common prefix around  $\ell_{\text{long}}$ . The computational complexity of this approach is  $\mathcal{O}(\log n_l + k)$ . For the online demo, we have set  $k = 300$ .

**Feature selection.** By default, `deepbeat.org` employs the `FastFeats` feature set by which the generation of an 8-line verse takes about 0.3 seconds. The user can additionally enable the `NN5` feature in which case the algorithm will retrieve the top-30 lines based on `FastFeats` and then rerank the top lines based on `FastFeatsNN5`. We only evaluate 30 lines using `NN5` since `NN5` is much heavier to compute than the other features (it increases the generation time to about 35 seconds per 8-line verse) and since recall at 30 is only 3.8 percentage points lower for `FastFeats` compared to `FastFeatsNN5`. The `NN5` feature could be used more heavily by acquiring a server which enables GPU computation or via parallelization (different candidate lines can be evaluated independently).

## 5.3 Empirical evaluation of generated lyrics

The lyrics generated by DeepBeat are evaluated by the rhyme density measure, introduced in Section 3.3.1, and by measuring the correlation between relevance scores assigned by DeepBeat and human preferences recorded via `deepbeat.org`.

### 5.3.1 Rhyme density of DeepBeat

We ran a single job to generate a hundred 16-bar verses with random seed lines. A randomly selected example verse from this set is shown in Appendix A. The rhyme density for the hundred verses is 1.431, which is, quite remarkably, 21% higher than the rhyme density of the top ranked human rapper, Inspectah Deck.

One particularly long multisyllabic rhyme created by the algorithm is given below (the rhyming part is highlighted)

“Drink and drown **in my own iniquity**  
Never smile style **is wild only grin strictly**”

In this example, the first line is from the song *Rap Game* by *D-12* and the latter from *I Don't Give a F\*\*k* by *AZ*. The rhyme consists of nine consecutive matching vowel phonemes.

### 5.3.2 Online experiment

In order to evaluate the algorithm, we performed an online experiment using the demo. The idea was to employ an approach which is used for optimizing search engines [14] where the clicked search result  $r_i$  is logged and the following pairwise preferences are extracted:  $r_i \succ r_j$ ,  $j = 1, \dots, i - 1$  (the lines below the selected line are ignored since we cannot assume that the user has evaluated those). The objective is to learn a ranking model which assigns relevance scores that agree with the extracted preferences.

At `deepbeat.org`, when a user clicks “Suggest Rhyming Line” button, 20 suggested candidate next lines are shown to the user. We wanted to see how often the line selected by the user was assigned a higher score than the lines above the selection. Our hypothesis was that the larger the absolute difference between the algorithm-assigned relevance scores of two lines was, the more likely the user would pick the line preferred by the algorithm.

In the initial data we collected through the website, we noticed that users are more likely to select a line the higher it appears on the list of suggestions. Furthermore, the users tend to prefer the fifth line since it often appears in the same location of the screen as the “Suggest Rhyming Line” button, so if a user is just playing around with the tool without putting much thought to the content of the suggestions, the user is likely to select the fifth suggestion. It is very challenging to get rid of this type of biases entirely when conducting an uncontrolled experiment in the wild but to mitigate the biases, we shuffled the order of the suggested lines and removed the first three selections of each user since we assumed that in the beginning users are more likely to play with the tool without thinking too much. Moreover, we wanted to create more variability among the suggestions, as the top 30 might be almost equally good, so we defined the set of suggested lines as the lines ranked: 1–14., 298–300., and three randomly picked lines from range 15–297.

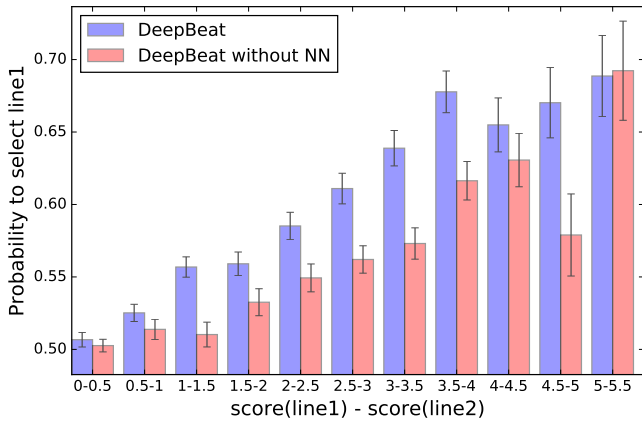
To avoid degrading the usability of the tool too much, we only applied the aforementioned manipulations when `NN5` was not enabled by the user. However, we also stored the text of the selected line and the previous lines to enable the computation of the relevance scores with `FastFeatsNN5` as a post-processing step. In total, this experiment resulted in 34 757 pairwise preferences from 1 549 users.<sup>9</sup>

The results of the experiment are shown in Figure 4. They confirm our hypothesis that the higher the difference between the relevance scores of two lines the more likely the users select the line which is evaluated more suitable by the algorithm. Furthermore, including the `NN5` feature improves the evaluations, which can be seen by studying the difference of the two data series in the figure. We may conclude that the learned RankSVM model generalizes and is able to successfully learn human preferences from existing rap lyrics, and that the developed deep neural network is an important component of the model.

<sup>8</sup>The candidate set could be found even faster by building a trie data structure of the phoneme strings. However, in our experiments, the binary search approach was fast enough and more memory-efficient.

<sup>9</sup>An anonymized version of the dataset including the scores assigned by DeepBeat is available at: <https://github.com/ekQ/dopelearning>





**Figure 4: Probability of a `deepbeat.org` user to select a line with a higher score from a pair of lines given the (binned) score difference of the lines. User preferences correlate with the scores assigned by DeepBeat.**

## 6. DISCUSSION

According to Boden [4], creativity is the ability to come up with ideas or artifacts that are (1) new, (2) surprising, and (3) valuable. Here, produced lyrics as a whole are novel by construction as the lines of the lyrics are picked from different original lyrics, even if individual lines are not novel. Lyrics produced with our method are likely to be at least as surprising as the original lyrics.

The (aesthetic) value of rap lyrics is more difficult to estimate objectively. Obvious factors contributing to the value are poetic properties of the lyrics, especially its rhythm and rhyme, quality of the language, and the meaning or message of the lyrics. Rhyme and rhythm can be controlled with relative ease, even outperforming human rappers, as we have demonstrated in our experiments. The quality of individual lines is exactly as good as the dataset used.

The meaning or semantics is the hardest part for computational generation. We have applied standard bag-of-words and LSA methods and additionally introduced a deep neural network model in order to capture the semantics of the lyrics. The importance of these features has been proven by the experimental results for the next line prediction problem and an online experiment. The features together contribute towards the semantic coherence of the produced lyrics even if full control over the meaning is still missing.

The task of predicting the next line can be challenging even for a human, and our model performs relatively well on this task. We have used our models for lyrics prediction, but we see that components that understand semantics could be very relevant also to other text processing tasks, for instance conversation prediction.

This work opens several lines for future work. It would be interesting to study automatic creation of story lines by analyzing existing rap songs and of novel lines by modifying existing lines or creating them from scratch. Even more, it would be exciting to have a fully automatic rap bot which would generate lyrics and rap them synthetically based on some input it receives from the outside world. Alternatively, the findings of this paper could be transferred to other text processing tasks, such as conversation predic-

tion which could carry significant business potential when applied to tasks like customer service automation.

## 7. CONCLUSIONS

We developed DeepBeat, an algorithm for rap lyrics generation. Lyrics generation was formulated as an information retrieval task where the objective is to find the most relevant next line given the previous lines which are considered as the query. The algorithm extracts three types of features of the lyrics—rhyme, structural, and semantic features—and combines them by employing the RankSVM algorithm. For the semantic features, we developed a deep neural network model, which was the single best predictor for the relevance of a line.

We quantitatively evaluated the algorithm with three measures. First, we evaluated prediction performance by measuring how well the algorithm predicts the next line of an existing rap song. The true next line was identified among 299 randomly selected lines with an accuracy of 17%, i.e., over 50 times more likely than by random, and it was ranked in the top 30 with 53% accuracy. Second, we introduced a rhyme density measure and showed that DeepBeat outperforms the top human rappers by 21% in terms of length and frequency of the rhymes in the produced lyrics. The validity of the rhyme density measure was assessed by conducting a human experiment which showed that the measure correlates with a rapper’s own notion of technically skilled lyrics. Third, the rap lyrics generator was deployed as a web tool (`deepbeat.org`) and the analysis of its usage logs showed that machine evaluations of candidate next lines correlate with user preferences.

## Acknowledgments

We would like to thank Stephen Fenech for developing the front end for `deepbeat.org` and Jelena Luketina, Miquel Perelló Nieto, and Vikram Kamath for useful comments on the manuscript.

## 8. REFERENCES

- [1] Y. Bengio, I. J. Goodfellow, and A. Courville. Deep learning. Book in preparation for MIT Press, 2015.
- [2] D. Berger. Rap genius university: Rhyme types. <http://genius.com/posts/24-Rap-genius-university-rhyme-types>. Accessed: 2015-05-07.
- [3] J. Bergstra, O. Breuleux, F. Bastien, P. Lamblin, R. Pascanu, G. Desjardins, J. Turian, D. Warde-Farley, and Y. Bengio. Theano: a CPU and GPU math expression compiler. *Proceedings of the Python for Scientific Computing Conference (SciPy)*, 2010.
- [4] M. A. Boden. *The Creative Mind: Myths and Mechanisms*. Psychology Press, 2004.
- [5] A. Bradley. *Book of rhymes: The poetics of hip hop*. Basic Books, 2009.
- [6] R. Collobert, J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa. Natural language processing (almost) from scratch. *Journal of Machine Learning Research*, 12:2493–2537, 2011.
- [7] S. Colton and G. A. Wiggins. Computational creativity: The final frontier? In *Proceedings of the*

*European Conference on Artificial Intelligence*, pages 21–26, 2012.

- [8] L. Deng. An overview of deep-structured learning for information processing. In *Proceedings of Asian-Pacific Signal & Information Processing Annual Summit and Conference*, pages 1–14, October 2011.
- [9] P. Edwards. *How to rap: the art and science of the hip-hop MC*. Chicago Review Press, 2009.
- [10] Flocabulary. How to write rap lyrics and improve your rap skills. <https://www.flocabulary.com/multies/>. Accessed: 2015-05-07.
- [11] A. Graves. Generating sequences with recurrent neural networks. In *Arxiv preprint*, arXiv:1308.0850, 2013.
- [12] H. Hirjee and D. G. Brown. Automatic detection of internal and imperfect rhymes in rap lyrics. In *Proceedings of the Tenth International Society for Music Information Retrieval Conference*, pages 711–716, 2009.
- [13] H. Hirjee and D. G. Brown. Using automated rhyme detection to characterize rhyming style in rap music. 2010.
- [14] T. Joachims. Optimizing search engines using clickthrough data. In *Proceedings of the 8th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 133–142, 2002.
- [15] T. Joachims. Training linear SVMs in linear time. In *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 217–226, 2006.
- [16] K. Lenzo. The CMU Pronouncing Dictionary. <http://www.speech.cs.cmu.edu/cgi-bin/cmudict>. Accessed: 2015-05-07.
- [17] T.-Y. Liu. Learning to rank for information retrieval. *Foundations and Trends in Information Retrieval*, 3(3):225–331, 2009.
- [18] E. Malmi. Algorithm that counts rap rhymes and scouts mad lines. <http://mining4meaning.com/2015/02/13/raplyzer/>, 2015. Blog post.
- [19] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean. Distributed representations of words and phrases and their compositionality. In *Advances in Neural Information Processing Systems*, pages 3111–3119, 2013.
- [20] J. Pennington, R. Socher, and C. D. Manning. Glove: Global vectors for word representation. *Proceedings of the Empirical Methods in Natural Language Processing*, 12, 2014.
- [21] M. Richardson, A. Prakash, and E. Brill. Beyond pagerank: machine learning for static ranking. In *Proceedings of the 15th international conference on World Wide Web*, pages 707–715. ACM, 2006.
- [22] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15:1929–1958, 2014.
- [23] I. Sutskever, J. Martens, and G. Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning*, pages 1017–1024. ACM, 2011.
- [24] P. Takala. Word embeddings for morphologically rich

languages. In *Proceedings of European Symposium on Artificial Neural Networks, Computational Intelligence and Machine Learning*, 2016.

- [25] J. M. Toivanen, H. Toivonen, A. Valitutti, and O. Gross. Corpus-based generation of content and form in poetry. In *Proceedings of the Third International Conference on Computational Creativity*, pages 175–179, 2012.
- [26] T. Veale. Creative language retrieval: A robust hybrid of information retrieval and linguistic creativity. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies–Volume 1*, pages 278–287. Association for Computational Linguistics, 2011.
- [27] O. Vinyals and Q. Le. A neural conversational model. In *Proceedings of ICML Deep Learning Workshop*, 2015.
- [28] D. Wu and K. Addanki. Learning to rap battle with bilingual recursive neural networks. In *Proceedings of the 24th International Conference on Artificial Intelligence*, pages 2524–2530, 2015.
- [29] D. Wu, V. S. K. Addanki, M. S. Saers, and M. Beloucif. Learning to freestyle: Hip hop challenge-response induction via transduction rule segmentation. In *Proceedings of the Empirical Methods in Natural Language Processing Conference*, pages 102–112, 2013.
- [30] L. Yu, K. M. Hermann, P. Blunsom, and S. Pulman. Deep Learning for Answer Sentence Selection. In *Proceedings of NIPS Deep Learning Workshop*, 2014.
- [31] M. D. Zeiler. Adadelata: an adaptive learning rate method. Technical report, 2012.

## APPENDIX

### A. SAMPLE VERSES

Table 4 shows an example of a generated verse. It was randomly selected from a set of a hundred verses, excluding the verses with profane language. The other verses are available at: <https://github.com/ekQ/dopelearning>

**Table 4: A randomly selected verse from the set of 100 generated lyrics.**

Everybody got one	(2 Chainz - Extremely Blessed)
And all the pretty mommies want some	(Mos Def - Undeniable)
And what i told you all was	(Lil Wayne - Welcome Back)
But you need to stay such do not touch	(Common - Heidi Hoe)
They really do not want you to vote	(KRS One - The Mind)
what do you condone	(Cam'ron - Bubble Music)
Music make you lose control	(Missy Elliot - Lose Control)
What you need is right here ahh oh	(Wiz Khalifa - Right Here)
This is for you and me	(Missy Elliot - Hit Em Wit Da Hee)
I had to dedicate this song to you Mami	(Fat Joe - Bendicion Mami)
Now I see how you can be	(Lil Wayne - How To Hate)
I see u smiling i kno u hattig	(Wiz Khalifa - Damn Thing)
Best I Eva Had x4	(Nicki Minaj - Best I Ever Had)
That I had to pay for	(Ice Cube - X Bitches)
Do I have the right to take yours	(Common - Retrospect For Life)
Trying to stay warm	(Everlast - 2 Pieces Of Drama)