Generation of Hip-Hop Lyrics with Hierarchical Modeling and Conditional Templates

Enrique Manjavacas[1], **Folgert Karsdorp**[2], and **Mike Kestemont**[1]

[1]Computational Linguistics and Psycholinguistics Research Center, University of Antwerp, Belgium
[2]Meertens Institute, Royal Netherlands Academy of Arts and Sciences, Amsterdam

Abstract

This paper addresses Hip-Hop lyric generation with conditional Neural Language Models. We develop a simple yet effective mechanism to extract and apply conditional templates from text snippets, and show—on the basis of a large-scale crowd-sourced manual evaluation—that these templates significantly improve the quality and realism of the generated snippets. Importantly, the proposed approach enables end-to-end training, targeting formal properties of text such as rhythm and rhyme, which are central characteristics of rap texts. Additionally, we explore how generating text at different scales (e.g. character-level or word-level) affects the quality of the output. We find that a hybrid form—a hierarchical model that aims to integrate Language Modeling at both word and character-level scales—yields significant improvements in text quality, yet surprisingly, cannot exploit conditional templates to their fullest extent. Our findings highlight that text generation models based on Recurrent Neural Networks (RNN) are sensitive to the modeling scale and call for further research on the observed differences in effectiveness of the conditioning mechanism at different scales.

1 Introduction

Neural Networks approaches to text generation have recently proliferated partly due to substantial progress made in Language Modeling. Being essentially a generative model, a Language Model (LM) is fit by definition to drive natural language generation systems. LMs based on Neural architectures, such as RNNs, ConvNets or self-attentive models such as Transformers, provide better fits to the underlying data distributions of the training material (currently holding the state-of-the-art on common benchmarks) and are also assumed to produce more realistic text than their count-based counterparts (Karpathy, 2016).

The end-to-end nature of such models and the ability to leverage out-of-domain data through pretraining have led to a broadening of domains in which text generation systems are being developed and applied. In particular, interest has emerged or increased in the generation of artistic text such as poetry (Zhang and Lapata, 2014; Yan, 2016), literature (Manjavacas et al., 2017), song lyrics (Watanabe et al., 2018) or cooking recipes (Kiddon et al., 2016), etc. In the present paper, we focus on generating Hip-Hop lyrics, a genre known for its relatively liberal formal properties (e.g. rhythm and rhyme) and topic specificity.

While full algorithmic modeling of the high-level creative process of song composition remains a challenge, we seek to improve the quality of the generated text by focusing on formal text properties. Typically, generating text with formal structure is done by applying constraints over the LM output distribution. By contrast, in this paper, we follow an end-to-end approach to generate text snippets that directly match the required structure. Our proposal makes use of templates based on sentence-level conditions (Ficler and Goldberg, 2017) that allow us to enforce rhyme and verse structure as it naturally occurs in training data. Our focus on Hip-Hop allows us to crowd-source an extensive collection of authenticity judgments through an online pseudo-Turing serious game. The evaluation shows the efficiency of the approach by bringing human guessing performance to chance-level.

Finally, while architectural improvements in Neural LMs target both character and word-level modeling, the application of LMs to artistic text generation has mostly focused on the word level. This situation is more likely the result of words being a central component of the creative process —e.g. topic, style and concepts are best modeled at the word level (Ghazvininejad et al., 2016;

Yan, 2016)—, than a lack of generation capabilities of character-level LMs (Karpathy et al., 2015). Moreover, despite a few exceptions (Jagfeld et al., 2018), comparisons of LM-based generation systems at different scales are not common. To fill this gap, we explore the effects of modeling scale on text generation quality — including character-level, syllable-level and a hierarchical LM (HLM) — as well as the interplay between modeling scale and the proposed conditional template approach.

More specifically, we make the following contributions. (i) We introduce a simple approach to template-based generation, suitable for genres with a loose formal structure such as poetry or Hip-Hop. Crucially, this approach does not require search or constrained decoding to generate formally correct output. (ii) We present a comparison of unconstrained language generation with LMs at different scales and provide empirical evidence that hierarchical modeling produces more realistic output than both character-level and word-level modeling. (iii) We find that the success of the conditioning mechanism is dependent on the LM scale and the type of condition. In particular, we find that the gains from hierarchical modeling do not compound with the benefits obtained from the conditioning mechanism, which calls for further research on the matter.

2 Related Work

Much research has been devoted to poetry generation systems, and the field has reached a considerable degree of maturity (see (Gonçalo Oliveira, 2017)). A variety of approaches based on LMs have been proposed, including both Markov Models (Barbieri et al., 2012) and RNNs (Zhang and Lapata, 2014; Ghazvininejad et al., 2016; Yan, 2016; Hopkins and Kiela, 2017; Lau et al., 2018). In the literature on Hip-Hop lyric generation, besides an RNN-based LM (Potash et al., 2015), researchers have explored retrieval-based approaches where a Support Vector Machine is trained to select the continuing sentence based on formal properties of the text and global semantic coherence (Malmi et al., 2016). Moreover, various strategies have been proposed to generate text that matches specific verse structures. For example, Zhang and Lapata (2014) follow a generate-and-select approach that discards non-rhyming lines, and Ghazvininejad et al. (2016) use a finite-state-acceptor to decode lines that meet the desired output structure. Hopkins and Kiela (2017) compose a WFST with an RNN to enforce meter in the output text. Finally, Lau et al. (2018) use a rhyme detector jointly trained with a LM and discard non-rhyming line-ending words based on the models' confidence scores.

Songs	Artists	Words	Vocabulary
64,542	28,099	37,236,248	380,013

Table 1: Counts the total number of songs, artists and words collected from the *Original Hip-Hop (Rap) Lyrics Archive* (OHHLA).

Statistic	μ	σ
Words/Song	576.93	223.77
Songs/Artist	2.3	7.65
Words/Artist	1325.2	4223.2

Table 2: Statistics on the average number of (1) words per song, (2) songs per artist, and (3) words per artist.

3 Dataset

The training data for this study was derived from the *Original Hip-Hop (Rap) Lyrics Archive* (OHHLA)[1], an online archive documenting Hip-Hop through since 1992 and offering a large collecting of Hip-Hop lyrics. A total of 64,542 songs were collected. The database contains almost exclusively English songs, although code-switching is common. The final corpus is the result of the following pre-processing steps. First, each text was tokenized using the Ucto tokenizer (Van Gompel et al., 2012). Second, all words were segmented into syllables using an in-house LSTM-based syllabifier trained on the CMU Pronouncing Dictionary (Lenzo, 2007). The syllabifier's segmentation accuracy is well over 99% on both a held-out development and test set, supporting confidence in its application to the lyric data. Finally, we applied the G2P toolkit[2] to extract phonological representations of words and corresponding stress patterns that will be exploited during training. The syllabified corpus consists of 43,531,133 syllables comprising 89,337 syllable types. A summary of overall corpus statistics is shown in Table 1 and Table 2.

[1]http://www.ohhla.com

[2]https://github.com/cmusphinx/g2p-seq2seq

4 LM-based Text Generation

We generate text by sampling from a LM implemented on top of LSTM Networks (Hochreiter and Schmidhuber, 1997) trained to predict the next symbol y in the sequence given the history using the following definition:

$$P(y_1 \dots y_n) = \prod_{t=1}^{n} P(y_t | y_{\dots<t}) \tag{1}$$

Regardless of the details of specific architectures, sampling is done in the following way. Let x_k be the activation for the k^{th} vocabulary symbol at the penultimate layer (i.e. before the output softmax layer). At any given step, we sample from the multinomial distribution defined by:

$$P(y_k) = \frac{x_k/T}{\Sigma_{l \in V} x_l/T} \tag{2}$$

where V refers to the vocabulary and T is a temperature parameter controlling the models confidence. We leave other sampling approaches such as Top-K Sampling (Fan et al., 2018) and Nucleus Sampling (Holtzman et al., 2019) for future work.

4.1 Hierarchical Language Model

LMs are typically trained at the character or word-level. Character-level modeling has the advantages of (i) reducing the vocabulary size and (ii) increasing the number of training examples available during model fitting, but it incurs the cost of enlarging the number of steps to account for a given dependency between any two given input words. Arguably, a hybrid approach that models language at the character level but also incorporates word-level information flow should provide a way out of such a trade-off (Karpathy et al., 2015).

Therefore, in the present paper, we compare text generation at three levels: character-level, syllable-level and a hierarchical LM (HLM) that integrates both levels. Note that we consider syllable-level instead of word-level based on two-fold reasoning: (i) similar to sub-word models — such as those induced through Byte-Pair-Encoding (Sennrich et al., 2016) or SentencePiece (Kudo and Richardson, 2018) —, syllable-level segmented input helps limiting the exploding vocabulary size of noisy corpora. (ii) Syllables play a more central role than words in a particularly rhythmic genre like Hip-Hop in which, moreover, a tendency towards monosyllabic words reduces the vocabulary differences for word-level modeling.

As mentioned above, the key idea behind HLM is to allow different layers to specialize in modeling the information flow at different scales. In order to achieve that, the HLM uses the chain-rule of probability to decompose the probability of a sentence into the product of the probabilities of words (exactly as in the word-level LM) but, furthermore, it decomposes the probability of each word into the product of the probabilities of its characters:[3]

$$P(w_{t+1} | w_{1\dots t}) = \prod_{i=1}^{|w_{t+1}|} P(c_{t+1}^{i} | c_{t+1}^{1 \dots i-1}; w_{1 \dots t}) \tag{3}$$

We implement the HLM with LSTM layers at different scales. A first bidirectional $\mathrm{LSTM}_{\mathrm{inp}}$ takes the input sequence of character embeddings of the current word w_t and produces word-level features concatenating the final activations of the forward and backward pass. Secondly, $\mathrm{LSTM}_{\mathrm{word}}$ takes word-level feature vector $\mathbf{w_t}$[4] and the recurrent state to generate sentence-level features $\mathbf{s_t} = \mathrm{LSTM}_{\mathrm{word}}(\mathbf{w_t}, \mathbf{s_{t-1}})$. Finally, $\mathrm{LSTM}_{\mathrm{out}}$ computes the vector of scores x for the next character c_{t+1}^{i+1} of the target word w_{t+1} using the previously decoded character embedding $\mathbf{c_{t+1}^{i}}$, $\mathbf{s_t}$ and the recurrent state $\mathbf{h_{t+1}^{i}}$:

$$\mathbf{x_{t+1}^{i+1}} = W \cdot \mathrm{LSTM}_{\mathrm{out}}([\mathbf{c_{t+1}^{i}}; \mathbf{s_t}], \mathbf{h_{t+1}^{i}}) \tag{4}$$

where W is a matrix that maps the LSTM output to the vocabulary space. HLM is a specific case of the Hierarchical Multi-scale LSTM by Chung et al. (2016) with the differences that HLM uses a fixed segmentation at syllable boundaries instead of implicitly learning a segmentation model, and that HLM only considers bottom-up information passing across layers. Interestingly, despite the simplification HLM achieves similar results on the Penn Treebank benchmark corpus (see Table 3).

4.2 Conditional Templates

Recent research has shown the effectiveness of a conditioning mechanism for controlled text generation (Ficler and Goldberg, 2017), which uses specific sentence-level information during training

[3] Note that while we discuss the HLM in terms of words, in practice, our implementation uses syllables following the argumentation at the beginning of the current section.

[4] We use bold to denote the feature vector (e.g., $\mathbf{w_t}$) corresponding to a particular input (e.g., word w_t).

Model	Parameters	BPC
HM-LSTM	NA	1.23
GHRNN	10.5M	1.225
2-layer LSTM	7.6M	1.275
HLM	7.5M	1.233

Table 3: A comparison of the proposed architecture with respect to a non-hierarchical deep character-level LSTM (2-later LSTM) and related hierarchical architectures on the Penn Treebank benchmark. Results correspond to bits-per-character (BPC). HM-LSTM corresponds to the Hierarchical Multiscale LSTM by Chung et al. (2016). GHRNN corresponds to the Gated Hierarchical RNN by Choi et al. (2018).

(e.g. tense, mood, sentiment or formal/informal style) to bias the generation towards text that reflects such conditions. Formally, such conditional information is encoded using condition embeddings and is fed into the LMs through vector concatenation. More formally, let c be a given condition with N_c assignments. For each c we allocate an embedding matrix $C_c \in \mathbb{R}^{N_c \times d}$. During training, each model input embedding is concatenated with a vector of condition embeddings $\mathbf{c} = [\mathbf{c_1}; \ldots; \mathbf{c_m}]$ representing the conditional information corresponding to the input sentence.

We deploy such conditioning mechanism in the form of *conditional templates* to the task of generating Hip-Hop lyrics. The idea behind conditional templates is to leverage the training material to bias the generation towards more realistic output. Consider the task of generating a verse consisting of m lines of Hip-Hop. In such a case, we sample a verse from the training corpus consisting of m lines and apply the corresponding conditions to a conditionally trained LM. The next question is what sentence-level information can be easily extracted and used to improve the quality and realism of the output. In the case of Hip-Hop, we focus on two formal characteristics that most typically represent Hip-Hop lyrics: *rhythm* and *rhyme* (Condit-Schultz, 2017).

Rhythm in Hip-Hop is characterized by a strict alignment between beat and stress with high correspondence between syntactic units and measures and a relatively stable ratio of number of syllables per beat (Adams, 2009). In order to approximate this stylistic feature[5], we condition our LMs on a measure of verse length. In particular, we count the number of syllables of each line in the verse and bucket them according to the following ranges: < 10, $(10-15)$, $(15-20)$ and > 20.

Rhyme Hip-Hop employs liberal rhyme patterns in terms of placement — e.g., off-beat, syncopated rhyme, etc. — and often relies on imperfect matches (e.g. slant rhyme Adams, 2009). To mimic such rhyming style, we condition LMs on phonological endings, which we define, in alignment with a loose notion of rhyme, as the syllabic nucleus of the last stressed syllable followed by the syllabic nuclei of any following syllables. For example, the rhyme-based condition corresponding to the line 'unite around the corner' is `AO1-ERO` — i.e. the ARPABET representations corresponding to the stressed syllabic nuclei of 'cor-' and '-ner'.[6] Such a representation is then shared with other rhyming words such as 'daughter' or 'offer'. A successfully trained conditional LM can thus generate rhymes when the same phonological condition is passed to the networks for two consecutive lines. Similarly, templates from the corpus contain rhyming schemes and patterns (such as AABB, ABAB, etc.) that the conditional models can exploit for a more realistic effect. Table 4 shows example generations from conditional models at all three considered scales.

4.3 Model Training Details

We implement all models in PyTorch (Paszke et al., 2017), with the following parametrizations. Input and condition embedding layers have dimensionality of 100. Non-hierarchical models have 2 hidden LSTM layers with 640 units per layer. By definition the HLM has 2 LSTM layers in addition to the bidirectional LSTM layer that computes character-level word embeddings. For replication purposes, our implementation is available online.[7]

[5] The approximation lies in the fact that we ignore stress patterns in the template source. Initially, we experimented with conditioning on line-level stress patterns extracted through a cluster analysis but found the approach inconclusive. The difficulty stems from the fact that Hip-Hop artists commonly shift the word stress in order to align it to the underlying beat, and such misalignment cannot be recovered based on only text.

[6] We focus on generating rhyming in verse-final position, which represents the most abundant type. We extract a total of 430 such phonological endings in our corpus, from which only 270 involve an actual rhyme in the training corpus. Interestingly, however, we observed that the conditional models generalize so as to generate rhymes on phonological endings that have not been seen during training.

[7] Code can found in the following url `https://www.github.com/emanjavacas/hierarchical-lm`.

"I Like It Like That" (by Hot Chelle Rae)			**Character-level Model**
I like it like that! Hey windows down	AW1	10-15	Now baby get the fuck out, check it out
Chillin with the radio on	AA1	<10	I be on top
I like it like that! Damn, the sun's so hot	AA1	10-15	I'll make some money what the fuck is goin' on

"Nothing to worry about" (by Peter Bjorn)			**Syllable-level Model**
C'mon everybody let's all get down, let's all get down, let's all get down	AW1	>20	We gon' shut'em down, if you wanna get down, don't fuck around
I've got nothin' to worry about	AW1	10-15	On the real, it's how it's goin' down
C'mon everybody let's all get down, let's all get down, let's all get down	AW1	>20	I'm trying to get this money right, you got to eat right now

"Lil like bic" (by Rae Sremmurd)			**Hierarchical Model**
Who said they got that stanky loud? I wanna smell it	IH1	10-15	Don't act like you ain't ready for this
You say you run your fuckin' town, I let you tell it	IH1	10-15	I never created this shit
Who really run the underground? I wanna meet you	UW1	10-15	You don't understand, it's all about you
I'm really tryna bite the style, you know we see you	UW1	10-15	Try to maintain, you know the rules

Table 4: Generated samples following the conditional templating approach. Left: the original snippet from which the template was extracted. Middle: Condition values extracted from the source text (phonological ending using the 2-letter ARPABET phoneset and the bucketed length in number of syllables). Right: generated text.

Scale	Conditional	Parameters	Result
Character		12.6M	1.65
Character	✓	12.9M	1.55
Syllable		29.8M	46.12
Syllable	✓	29.9M	33.43
HLM		14.M	1.38
HLM	✓	14.9M	1.27

Table 5: Model details. In agreement with the literature, the results correspond to perplexity for syllable-level models and bits per character (BPC) for character-level models.

We trained all models with a cross-entropy objective targeted at predicting the next symbol in the sequence. Parameter optimization was done using the Adam optimizer (Kingma and Ba, 2015) with default hyperparameters. Models are regularized using dropout (Srivastava et al., 2014) on the input embeddings, variational dropout (Gal and Ghahramani, 2016) on hidden recurrent layers, and default L2 penalty on model parameters. Finally, we stop training based on an early-stopping criterion computed after each epoch on held-out data. Table 5 shows total number model parameters and development perplexity per configuration.

5 Evaluation

Our first evaluation concerns the quality of the Hip-Hop snippets generated by each of the six architectures (three modeling scales, each with a conditioned variant). We focus on the effectiveness of the conditional template approach and hierarchical modeling. Evaluating artistic text generation poses additional challenges, mostly due to the absence of reference text against which a model output can be compared. While some authors rely on questionnaires addressing poetic properties of interest (e.g., "poeticness", "grammaticality", "meaningfulness") for evaluation (Das and Gambäck, 2014), we instead turn to a Turing-like setup that allowed us to crowdsource a large-scale pool of user authenticity judgments. In order to encourage user participation, we implemented a serious game where participants were shown Hip-Hop samples of lengths of 3 to 4 lines and were tasked to guess whether the displayed text was generated or real in 15 seconds.[8] Participants were motivated by being shown feedback immediately after each answer. Furthermore, the game entered a "sudden-death" phase after the first ten guesses, in which a wrong answer would finish the game. Finally, a leader-board was kept visible, showing the scores of the ten best performing participants. The resulting dataset underlying the present evaluation comprises 3,620 guesses by 670 participants.

In order to leverage the collected evaluations, we model guessing behavior using a Logistic Regression model (implemented in `brms`, Bürkner and others, 2017), taking into account user-specific variability through the inclusion of varying intercepts (i.e. for each participant, we use a unique intercept parameter). Our evaluation strategy contrasts with similar approaches in the literature — (e.g. Netzer et al., 2009) — which typically only provide raw empirical, single point estimates. Regularized estimates obtained from using a varying intercepts model provide more accurate estimates for individual user intercepts, enabling predictions about future behavior that are less prone to both over- and underfitting (cf. McElreath, 2015). Additionally, the interaction between generation scale and conditioning are modeled as fixed effects.

As shown in Figure 1, hierarchical modeling outperforms both character and syllable-level models in the unconditioned setup, with the median guessing accuracy dropping to 54.6%. Moreover, conditional templates push guessing accuracy further down for all models, with HLM and syllable-level achieving a median accuracy of 51.9% and 49.4%, respectively. Interestingly, the effect of the conditional templates differs across models. The smallest effect is observed for the hierarchical model (decrease of 2.6 points), followed by the character-level model (decrease of 6.7 points), while the effect on syllable-level model corresponds to a decrease of 13.4 points. The relatively high impact of conditioning on syllable-level generation contrasts sharply with the much smaller improvement on both character-level and HLM.

On first sight, this result seems to suggest that conditioning is more effective at higher modeling levels, perhaps hinting at optimization incompat-

[8] Generated text was sampled at random from one of the six models.

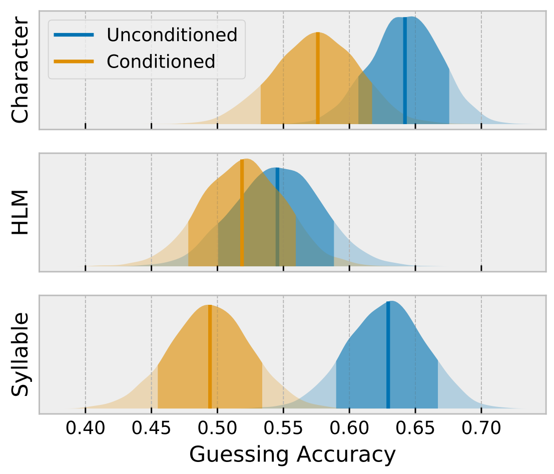

Figure 1: Results of crowd-sourced evaluation. Displayed results correspond to the full posterior predictive distributions of a logistic regression model, with (the interaction between) conditioning and generation scale as fixed effects, and participants as varying intercepts. For explanation purposes, median and 80% credible intervals are highlighted.

ibilities between sentence-level conditioning and character-level training objectives. Though, in order to better understand this result, we pursue the following two questions. First, why are conditional templates much more effective at a higher-level scale (i.e., syllable-level)? Second, what textual properties characterize text generated by different models, and can explain the better performance of the hierarchical model in particular?

5.1 Modeling scale and conditioning

To better understand the divergent effectiveness of conditioning at different scales, we investigate to what extent different models succeed at generating text that matches the conditions required by the template. Note that the benefits of a conditioned model might not be restricted to a model's ability to fulfill the target template conditions—for example, rhyme and rhythm information results in a better fit to the data as shown in Table 5. However, successfully replicating formal structures seen in the training data ensures a level of realism by definition and thus can be interpreted as an, at least partial, explanation for the observed differences in performance.

In order to quantify the ability of models to successfully generate the requested templates, we generate a dataset of lines exploring the space of possible templates. For each of the 2150 combinations (430 rhyming conditions by 5 length buckets), we sample 1000 lines. We then syllabify each of the lines using the same pre-processing pipeline described in Section 3. Subsequently, rhyme generation accuracy (`Acc`) can be quantified by the proportion of generated lines with the expected phonological ending. Moreover, we also quantify rhyme diversity (`H`) — i.e. the entropy of the distribution of successfully generated rhyme words. Finally, in order to quantify the ability to meet target verse length, we compute the average difference in syllables between the generated verse length and corpus-level average length per bucket (`Diff`).

The results in Table 6 show that syllable-level is, in fact, most accurate and diverse at generating rhyme by a large extent. The HLM achieves higher accuracy than the character-level but similar diversity. Overall, rhyme diversity is notably lower in generated text than in real text ($\mathtt{H} = 1.669$), a result that is in agreement with the expectations. In terms of rhythm, we observe a different picture: the character-level model generates lines much closer to the observed data than both HLM and syllable-level. From the last two, the HLM improves over syllable-level but both tend to produce shorter lines.

These results seem to suggest that character-level RNNs excel at modeling surface-level information responsible for estimating the current number of processed symbols, and can thus very accurately replicate the verse lengths observed in the training data. Moreover, it seems that the syllable-level model can derive a more substantial improvement from the conditional templates because rhyming patterns have an arguably more prominent impact on the perceived realism — however, we will leave an analysis of perceived realism for future work. Finally, it appears that in terms of conditioning, our hierarchical model does not succeed in exploiting the best of both worlds.

5.2 Modeling scale and text quality

We now turn to characterize the effect of modeling text at different scales as well as, more specifically, what textual properties single out hierarchically generated text. In order to approach this question, we utilize the unconditioned output from the main experiment and conduct a feature analysis across the following linguistic levels:

Prosody We quantify *rhyme* as the proportion of rhyming lines in the snippet, *assonance* as the proportion of the most frequent stressed syllable nucleus over total number of syllables, and *alliteration* as the proportion of consecutive words with equal consonant onset.

Condition		Char	HLM	Syll
Rhyme	`Acc`	28.37	36.29	61.75
	`H`	0.861	0.836	1.077
Rhythm	`Diff`	0.09	-1.59	-2.59

Table 6: Effects of conditioning on model scale. The rhyme diversity (`H`) as observed in the training corpus for a sample of comparable size equals 1.669.

Morphology We approximate the morphological complexity of the text with the average *word-length* in syllables.

Lexical level *lexical diversity* is measured using entropy based on overall word distributions. Moreover, we also quantify the proportion of consecutive *word repetitions*, which represent a common artifact in LM-based text generation (Holtzman et al., 2019).

Syntax We approximate syntactic complexity based on the average *mean tree depth* from the corresponding dependency parse trees of the snippet lines. Parse trees are extracted using the dependency parser provided by AllenNLP[9] based on Dozat and Manning (2016) and trained on the PTB3.0 corpus.

Based on such features, we fit a Random Forest to classify the model underlying the corresponding text snippet. We resort to the machine learning library `scikit-learn` (Pedregosa et al., 2011) for the implementation and extract feature importance scores following the feature permutation approach detailed in Parr et al. (2018). Furthermore, in order to extract feature-class associations (i.e., which class each feature is mostly predictive of) odds-ratios are computed based on a linear model taking character-level as reference class. The resulting Random Forest achieves 91.7 out-of-bag accuracy, which provides certainty that the feature-set has sufficiently large coverage.

Figure 2 ranks features by importance scores. As we can see, word-length is by far the strongest predictor. The feature is most strongly associated with HLM and slightly less with syllable-level modeling. Following word-length, we encounter syntax (mean tree-depth) and lexical diversity, which again are mostly associated with HLM — with odds-ratios in favour of HLM amounting to

[9] `https://www.allennlp.org/models`

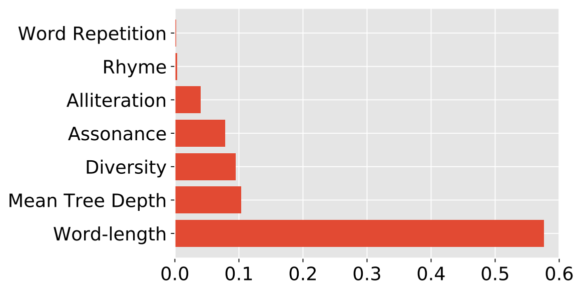

Figure 2: Feature importance analysis based on a Random Forest classifier trained to predict modeling scale.

3.8 and 2.4, respectively. Furthermore, prosodic features — in particular assonance — play a role in distinguishing character-level output from the other models. Finally, word repetition and rhyme density show near-zero importance scores.

Based on the present feature analysis, it can be inferred that one of the main advantages of hierarchical modeling relates to increased lexical diversity, which is further boosted by the ability to generate longer and more morphologically complex words. On the other hand, character-level modeling seems to be better characterized by surface-level prosodic features (in particular assonance). This analysis would connect with the interpretation advanced in Section 5.1, in which character-level modeling was shown to provide an accurate replication of a related surface-level textual property: rhythm as captured by verse length.

6 Discussion & Conclusion

Based on a large-scale evaluation involving hundreds of participants and authenticity judgments, we have shown that the modeling scale influences the quality of generated Hip-Hop lyrics. A feature analysis shows that hierarchically generated text displays morphologically and syntactically more complex output as well as higher lexical diversity. All such properties may help explain the better scores achieved by the hierarchical model in the absence of conditioning.

Furthermore, the proposed end-to-end approach to enforce formal structure in texts has similarly proved efficient. It reduces the human guessing accuracy of all models and is particularly efficient in the case of syllable-level modeling. Moreover, our analysis of the interplay between modeling scale and conditioning showed that syllable-level modeling displays much greater ability to exploit rhyme templates than the other lower-scale modeling variants. This advantage can help to explain the more pronounced effect of conditional templates on syllable-level modeling when considering that rhyme patterns contribute arguably more strongly to the realism of a generated snippet.

And yet, character-level modeling scored much better than the other models at generating the requested verse lengths and was shown to be positively associated with prosodic features such as assonance by the feature analysis. Both results thus seem to suggest that character-level modeling has an edge at capturing surface-level information. The overall lower performance of the character-level model implies, however, that such an advantage does not translate in improved realism as perceived by the participants.

Finally, the evaluation shows that despite the already mentioned advantages of hierarchical modeling, the effects of conditional templates did not compound in this case. This result is somewhat discouraging, since the primary motivation of hierarchical modeling is to overcome deficiencies of both character and word-level modeling. The analysis in Section 5.1 shows that our implementation of the conditioned HLM scores in between the other two models. Future research might be able to overcome this drawback by carefully designing adaptive mechanisms that let the model decide to which layer in the hierarchy a particular type of sentence-level conditional embedding should be fed.

7 Future Work

Our study suggests several directions for future work. The most urgent issue, briefly touched upon above, concerns an investigation of improved hierarchical architectures that can exploit conditioning information better than either character-level and word-level models in isolation. Moreover, the positive results obtained for the hierarchical model in isolation encourage scaling up the modeling hierarchy, investigating the inclusion of higher scales such as stanza and document level. Furthermore, while we have only considered templates covering formal aspects of the text, the same approach can be extended to include content features such as keyword or stanza-level topic information. Finally, in this study, we have restricted ourselves to relatively short snippets of text, but future work should move on and consider an evaluation on more substantial text portions.

References

Kyle Adams. 2009. On the metrical techniques of flow in rap music. *Music Theory Online*, 15(5).

Gabriele Barbieri, Franois Pachet, Pierre Roy, and Mirko Degli Esposti. 2012. Markov Constraints for Generating Lyrics with Style. In *Ecai*, volume 242, pages 115–120.

Paul-Christian Bürkner and others. 2017. brms: An R package for Bayesian multilevel models using Stan. *Journal of Statistical Software*, 80(1):1–28.

Iksoo Choi, Jinhwan Park, and Wonyong Sung. 2018. Character-level Language Modeling with Gated Hierarchical Recurrent Neural Networks. *Proc. Interspeech 2018*, pages 411–415.

Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. 2016. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*.

Nathaniel Condit-Schultz. 2017. MCFlow: A Digital Corpus of Rap Transcriptions. *Empirical Musicology Review*.

Amitava Das and Bjrn Gambäck. 2014. Poetic Machine: Computational Creativity for Automatic Poetry Generation in Bengali. In *ICCC*, pages 230–238.

Timothy Dozat and Christopher D Manning. 2016. Deep biaffine attention for neural dependency parsing. *arXiv preprint arXiv:1611.01734*.

Angela Fan, Mike Lewis, and Yann Dauphin. 2018. Hierarchical Neural Story Generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 889–898, Melbourne, Australia. Association for Computational Linguistics.

Jessica Ficler and Yoav Goldberg. 2017. Controlling Linguistic Style Aspects in Neural Language Generation. In *Proceedings of the Workshop on Stylistic Variation*, pages 94–104, Copenhagen, Denmark. Association for Computational Linguistics.

Yarin Gal and Zoubin Ghahramani. 2016. A theoretically grounded application of dropout in recurrent neural networks. In *Advances in neural information processing systems*, pages 1019–1027.

Marjan Ghazvininejad, Xing Shi, Yejin Choi, and Kevin Knight. 2016. Generating Topical Poetry. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 1183–1191, Austin, Texas. Association for Computational Linguistics.

Hugo Gonçalo Oliveira. 2017. A Survey on Intelligent Poetry Generation: Languages, Features, Techniques, Reutilisation and Evaluation. In *Proceedings of the 10th International Conference on Natural Language Generation*, pages 11–20. Association for Computational Linguistics.

Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

Ari Holtzman, Jan Buys, Maxwell Forbes, and Yejin Choi. 2019. The curious case of neural text degeneration. *arXiv preprint arXiv:1904.09751*.

Jack Hopkins and Douwe Kiela. 2017. Automatically Generating Rhythmic Verse with Neural Networks. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 168–178. Association for Computational Linguistics.

Glorianna Jagfeld, Sabrina Jenne, and Ngoc Thang Vu. 2018. Sequence-to-Sequence Models for Data-to-Text Natural Language Generation: Word- vs. Character-based Processing and Output Diversity. In *Proceedings of the 11th International Conference on Natural Language Generation*, pages 221–232, Tilburg University, The Netherlands. Association for Computational Linguistics.

Andrej Karpathy. 2016. The Unreasonable Effectiveness of Recurrent Neural Networks.

Andrej Karpathy, Justin Johnson, and Li Fei-Fei. 2015. Visualizing and understanding recurrent networks. *arXiv preprint arXiv:1506.02078*.

Chlo Kiddon, Luke Zettlemoyer, and Yejin Choi. 2016. Globally Coherent Text Generation with Neural Checklist Models. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 329–339, Austin, Texas. Association for Computational Linguistics.

Diederik P. Kingma and Jimmy Lei Ba. 2015. Adam: a Method for Stochastic Optimization. *International Conference on Learning Representations 2015*, pages 1–15.

Taku Kudo and John Richardson. 2018. SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing. *arXiv preprint arXiv:1808.06226*.

Jey Han Lau, Trevor Cohn, Timothy Baldwin, Julian Brooke, and Adam Hammond. 2018. Deep-speare: A joint neural model of poetic language, meter and rhyme. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1948–1958. Association for Computational Linguistics.

Kevin Lenzo. 2007. The CMU pronouncing dictionary.

Eric Malmi, Pyry Takala, Hannu Toivonen, Tapani Raiko, and Aristides Gionis. 2016. DopeLearning: A computational approach to rap lyrics generation. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 195–204. ACM.

Enrique Manjavacas, Folgert Karsdorp, Ben Burtenshaw, and Mike Kestemont. 2017. Synthetic Literature: Writing Science Fiction in a Co-Creative Process. In *Proceedings of the Workshop on Computational Creativity in Natural Language Generation (CC-NLG 2017)*, pages 29–37, Santiago de Compostela, Spain. Association for Computational Linguistics.

Richard McElreath. 2015. *Statistical rethinking: A Bayesian course with examples in R and Stan*. Chapman and Hall/CRC.

Yael Netzer, David Gabay, Yoav Goldberg, and Michael Elhadad. 2009. Gaiku: Generating haiku with word associations norms. In *Proceedings of the Workshop on Computational Approaches to Linguistic Creativity*, pages 32–39. Association for Computational Linguistics.

Terence Parr, Kerem Turgutlu, Christopher Csiszar, and Jeremy Howard. 2018. Beware Default Random Forest Importances.

Adam Paszke, Sam Gross, Soumith Chintala, and Gregory Chanan. 2017. Pytorch: Tensors and dynamic neural networks in python with strong gpu acceleration. *PyTorch: Tensors and dynamic neural networks in Python with strong GPU acceleration*.

Fabian Pedregosa, Gal Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, and others. 2011. Scikit-learn: Machine learning in Python. *Journal of machine learning research*, 12(Oct):2825–2830.

Peter Potash, Alexey Romanov, and Anna Rumshisky. 2015. GhostWriter: Using an LSTM for Automatic Rap Lyric Generation. *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, (September):1919–1924.

Rico Sennrich, Barry Haddow, and Alexandra Birch. 2016. Neural Machine Translation of Rare Words with Subword Units. In *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1715–1725, Berlin, Germany. Association for Computational Linguistics.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research*, 15:1929–1958.

Maarten Van Gompel, Ko Van Der Sloot, and Antal Van den Bosch. 2012. Ucto: Unicode Tokeniser Reference Guide. Technical report.

Kento Watanabe, Yuichiroh Matsubayashi, Satoru Fukayama, Masataka Goto, Kentaro Inui, and Tomoyasu Nakano. 2018. A Melody-Conditioned Lyrics Language Model. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 163–172, New Orleans, Louisiana. Association for Computational Linguistics.

Rui Yan. 2016. i, Poet: Automatic Poetry Composition through Recurrent Neural Networks with Iterative Polishing Schema. In *IJCAI*, pages 2238–2244.

Xingxing Zhang and Mirella Lapata. 2014. Chinese Poetry Generation with Recurrent Neural Networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 670–680. Association for Computational Linguistics.