End-to-end style-conditioned poetry generation: What does it take to learn from examples alone?

Jörg Wöckener[1] and **Thomas Haider**[2] and **Tristan Miller**[3]
Thanh Tung Linh Nguyen[1] and **The-Khang Nguyen**[1] and **Minh Vu Pham**[1]
Jonas Belouadi[1] and **Steffen Eger**[1]

[1] Technische Universität Darmstadt [2] Universität Göttingen
[3] Austrian Research Institute for Artificial Intelligence

Abstract

In this work, we design an end-to-end model for poetry generation based on conditioned recurrent neural network (RNN) language models whose goal is to learn stylistic features (poem length, sentiment, alliteration, and rhyming) from examples alone. We show this model successfully learns the 'meaning' of length and sentiment, as we can control it to generate longer or shorter as well as more positive or more negative poems. However, the model does not grasp sound phenomena like alliteration and rhyming, but instead exploits low-level statistical cues. Possible reasons include the size of the training data, the relatively low frequency and difficulty of these sublexical phenomena as well as model biases. We show that more recent GPT-2 models also have problems learning sublexical phenomena such as rhyming from examples alone.

1 Introduction

Poetry is a very old form of human language use which is deeply embedded in the heritage of many cultures (Beissinger, 2012; Fabb and Halle, 2010). It is characterized by refined, creative, and sublime expressions and linguistic structure. Thus, it comes as little surprise that automatic poetry generation has long captivated artificial intelligence (AI) researchers (Gervás, 2001; Gonçalo Oliveira et al., 2017), for any general solution to this task will have assuredly brought us a step closer to understanding human creativity.[1]

[1]Poetry generation systems may have manifold applications. For example, an AI tool that could assist songwriters in composing lyrics to a given tune, or even generate them entirely automatically, would have commercial potential. Besides such entertainment-oriented applications, producing tools for poetry generation will necessarily involve the development of poetry analysis tools – since generation and analysis are complementary aspects of poetry (Jhamtani et al., 2019) – which may prove useful in their own right. And the production and application of these analysis tools could result in insights into the structural and aesthetic features that distinguish human-penned poetry from its artificially produced counterpart.

```
 -    +   | -      + | -       +    | -    +    |
 I   felt   a     funeral      in     my  brain ,

 -    +   | -      + | -       +    |
And  mourners ,    to  and    fro    ,

 -    +   | -      + | -       +    | -    +    |
Kept treading ,   treading ,  till    it seemed

 -    +   | -      + | -       +    |
That sense was    breaking   through .
```

Figure 1: Example of alliteration (red), a (near) rhyme (blue), and metre (black/grey). This is the first stanza of 'I felt a Funeral, in my Brain' by Emily Dickinson. The rhyme scheme is 'abcb', the overall lexical sentiment of the stanza is negative (many words are associated with negative sentiment), and the time period is the 19th century.

A crucial factor when producing poetry (either by humans or automatically) is *style*. While definitions of style have varied considerably over time, space, and fields of study (Herrmann et al., 2015), there is widespread agreement that the study of literary *form*, not (just) *content*, is what allows us to determine the criteria for aesthetic appreciation and impact. That is, it is formal or stylistic features that often distinguish sophisticated, effectual, or simply 'good' poetry from poetry that is pedestrian, mundane, or just plain 'bad' (Jakobson, 1960; Shlovsky, 1965; Ganjigunte Ashok et al., 2013; Kao and Jurafsky, 2015; Menninghaus et al., 2017).

In this work, we consider the problem of *style-conditioned poetry generation*, where the user may specify stylistic factors that the generated text has to fulfil. Our stylistic factors include rhyme, alliteration, sentiment, text length, and time period. Examples of some of our stylistic features (particularly rhyme and alliteration) can be found in Figure 1. (We include metre in the example, but do not consider it in the experiments.)

In generating poetry conditioned upon stylistic factors, we explore a theme in the automatic generation of poetry that we believe will dominate the field in the upcoming years: *end-to-end generation* of formal poetry *from examples alone*, without the need for human involvement in designing the model or filtering its output (e.g., via hard-coded stylistic constraints on rhyme, alliteration, etc.). We believe advances in this task serve as an indicator of the maturity of natural language processing (NLP) systems as they progress towards true AI. Indeed, the recent success of GPT-3 (Brown et al., 2020) with its focus on few-shot learning (i.e., learning from a few illustrative examples, even without model updating) is inspiration for our work to design poetry generation models that learn relevant stylistic features without any other form of human intervention.

We find that, while our models can capture sentiment, time epoch, and text length, they have trouble learning intricate phonetic stylistic features from examples alone, instead learning low-level statistical cues that are weakly correlated with these target features. This holds both for a classical RNN language model and more recent transformer models (GPT-2). We take this as a negative result and as a challenge for future poetry generation systems, viz., to be able to produce typical relevant stylistic factors of poetry without hard-coding them into the model architecture.

2 Related work

Poetry analysis. Computational analysis of poetry has been concerned largely with formal features such as metre (Greene et al., 2010; Agirrezabal et al., 2016; Estes and Hench, 2016), rhyme (Reddy and Knight, 2011; Haider and Kuhn, 2018), and enjambment (Ruiz et al., 2017; Baumann et al., 2018). More recently, higher-level phenomena, including semantic coherence (Herbelot, 2015), metaphor (Reinig and Rehbein, 2019; Kesarwani et al., 2017), and diachronic analysis of tropes (Haider and Eger, 2019) have come into focus. Haider et al. (2020) annotate poetry for fine-grained aesthetic emotion categories elicited in readers.

Poetry generation. In the era of statistical NLP, text generation has evolved along the path of *n*-gram language models (Kneser and Ney, 1995), whose inherent modelling limitations allow them to consider only a finite amount of history when generating next output symbols, to recurrent neural network (RNN) language models (Sutskever et al., 2011), which can take an infinite history into account but suffer from vanishing and exploding gradients, to transformer models (Vaswani et al., 2017), which have constant path lengths between inputs and outputs. Transformers are feed-forward networks that use self-attention between consecutive layers. Their shorter paths between inputs and outputs make it easier to learn long-range dependencies. They can also be parallelized easier. As a consequence, they can be pre-trained on massive amounts of data, where they acquire general text generation abilities (Radford et al., 2019).

Approaches to poetry generation can be categorized into pre- and post-deep learning era approaches. Pre-deep learning approaches were highly hand-engineered and 'top-down', using specific grammar formalisms and linguistic resources to generate poetry; they typically built stylistic aspects such as rhyme and rhythm into their model architectures (Manurung et al., 2000; Gervás, 2001; Gonçalo Oliveira, 2013; Colton et al., 2012).

Neural approaches to poetry generation learn from collections of real poetry data. Most of them use RNN language models (Zhang and Lapata, 2014; Lau et al., 2018). This framework has also been extended to related tasks such as generating poetry from images (Liu et al., 2018), translating poetry (Ghazvininejad et al., 2018), and generating song lyrics given an input melody (Watanabe et al., 2018). A drawback shared by most poetry generation systems, including even very recent ones (Van de Cruys, 2020; Agarwal and Kann, 2020), is that they require hard- and hand-coded rules to generate poetry with specific properties, such as rhyming and alliteration, or to filter output not having these properties. For example, to ensure rhyming, Lau et al. (2018) exploit the fact that their data (sonnets) has a particular structure from which they can infer that certain word pairs must rhyme, which they incorporate in the modelling. Hämäläinen and Alnajjar (2019) define rules for style features. Agarwal and Kann (2020) tell the model when a rhyming word is required and then modify the prediction process. This means that the models themselves lack the ability to discover elementary properties of poetry themselves, but instead rely on the modeller's intention and ingenuity to do so.

Recently, Jhamtani et al. (2019) introduced an intriguing approach to poetry generation that learns rhyming constraints using an adversarial model: an RNN language model (the *generator*) generates poetry and a discriminator (the *adversary*) decides whether the line endings of the generated text are plausible for poetry. The generator and the discriminator compete against each other: the generator tries to generate poetry that the discriminator misclassifies and the discriminator aims at distinguishing generated poetry from human-authored poetry. The authors show that this competition leads to models learning rhyming with higher probability. While this model comes close to our envisioned ideal of a poetry generation system learning from examples alone, it still has the drawback that the modellers need to make the model aware of what to pay attention to (i.e., character information of ending words). This is unsatisfactory given the large space of possible features to consider in generated poetry.

Formally similar approaches to ours are those of Ficler and Goldberg (2017) and Manjavacas et al. (2019) (without emphasizing the 'examples-only' aspect, however). Ficler and Goldberg (2017) conditioned RNN language models on stylistic features such as length and whether the text appears to be professionally written, and on sentiment and theme. They showed this approach to be successful for the given conditions in the movie reviews domain, though they did not deal with the much more challenging case of poetry. Manjavacas et al. (2019) generate hip-hop lyrics conditional upon rhythm and rhyme. To enforce rhyming, they feed phonetic rhyme representations to the model, and thus also encode relevant knowledge top-down into the model.

2.1 Poetry evaluation

Evaluation of generated poetry is usually done either automatically or manually. Zhang and Lapata (2014) compare generated lines to manually produced 'gold-standard' lines using BLEU (Papineni et al., 2002), a metric borrowed from machine translation, for the task of continuing an initial line of poetry. They also report perplexity, a widely used metric for language model evaluation. For manual evaluation, they ask human annotators to rate the generated poems for fluency, coherence, meaning, and poeticness. Hopkins and Kiela (2017) and Lau et al. (2018) use a sort of Turing test where they ask human participants to distinguish between human- and computer-generated poems. Lau et al. (2018) additionally have an expert rate generated poetry with respect to metre, rhyme, readability, and emotion. Ghazvininejad et al. (2017) ask crowd-workers to give star ratings for the general quality of the generated poems, and then investigate whether users give higher scores to style-adjusted poems compared to the default poems.

In our work, we choose a simple evaluation scheme that measures whether the generated poetry satisfies user constraints. We do not measure other qualities of the generated poetry as we are mostly interested in whether end-to-end systems can successfully generate poetry from examples alone, without human intervention.

3 Models

Our goal is to design an end-to-end poetry generation model that learns to associate a desired style feature vector with poetic content and form purely inductively, using examples rather than hand-coded knowledge about what rhyming, alliteration, metre, and sentiment are. To do so, we implement a *conditioned language model* that samples the next word w_{t+1} to be generated conditioned on the history of past words $w_0, \ldots, w_t$ and a style constraint c. That is, it samples w_{t+1} from the conditional distribution $P(w_{t+1} \mid w_0, w_1, \ldots, w_t, c)$. We implement the model in this formula by a unidirectional RNN language model. At every time step, the RNN predicts the next word to be generated by sampling from a softmax distribution over the whole vocabulary, taking the current word vector $\mathbf{w}_t$ as well as the current history vector $\mathbf{h}_t$, which summarizes all past inputs, into account. The model starts generation from a special input token, 'SOS', and generates until it encounters an end-of-sentence token 'EOS'. To take the context c into account, we encode it as a feature vector $\mathbf{c}$ and concatenate it to every input representation. While there are other possible ways of including $\mathbf{c}$, our way constantly 'reminds' the model of the style context, which we hope better enforces that style in the output. The model is illustrated in Figure 2. Our RNN language model also uses character-level information to build input representations, which may help it to generalize to rare words or sublexical phenomena, such as rhyming. Our implementation is taken from that of Reimers and Gurevych (2017).[2]

For the conditions c, we consider the following:

Rhyming: How much rhyming is present in a base unit (e.g., a stanza or full poem) of poetry? We measure the degree of rhyming by using a

[2]`https://github.com/UKPLab/emnlp2017-bilstm-cnn-crf`

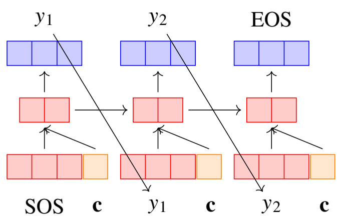

Figure 2: A conditioned RNN language model.

supervised rhyme model based on character-level Siamese bidirectional long short-term memory (Bi-LSTM) networks (Haider and Kuhn, 2018), available for English and German. We count both internal and end rhymes as contributing to the rhyme level of a base unit of poetry.

Alliteration: Alliteration is a device, often associated with humour (Mihalcea and Strapparava, 2006), where consecutive words begin with the same sound or sounds. As a character does not uniquely specify a sound in most writing systems (cf. *cat* and *chase*), it is not sufficient to compare the first characters of the words. Instead, we use the CMU Sphinx Sequence-to-Sequence grapheme-to-phoneme toolkit[3] for converting words into a phonemic representation and then check whether subsequent words start with the same phoneme. We measure the degree of alliteration by counting how many alliterating word sequences occur within a base unit of poetry.

Sentiment: To determine the sentiment of a base unit of poetry, we use a lexicon lookup approach. For English, we use the opinion lexicon developed by Hu and Liu (2004) and for German, we use the German polarity clues word lists by Waltinger (2010). The English lexicon consists of 2,007 positive and 4,782 negative words and the German lexicon consists of 17,627 positive and 19,962 negative words. In each base unit of poetry, we count the number of words that appear in the positive and negative lists of the lexicon, respectively, and then calculate the difference between these two numbers. The final result is normalized by dividing by the number of total matches for this unit, resulting in the formula $c_\sigma = \frac{p-n}{p+n}$, where p is the number of positive words found and n is the number of negative words found. For instance, if four words of the unit were found in the positive list and six words in the negative list, the final rating is $-2/10 = -0.2$.

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

3.1 Training and testing mode

In **training mode**, we extract sentiment, rhyme, and alliteration levels of poems (or stanzas) and pair them with the poems themselves. The model then predicts the text of poems (next words) based on the previous text and the context vector $\mathbf{c}$. The model is end to end and has no built-in understanding of rhyme, sentiment, nor alliteration, except for what it gleans from its input examples. In **testing mode**, the user may select a context specification in terms of rhyme, alliteration, and sentiment, which is fed into the RNN language model. The model starts generating from the SOS token and terminates when it has produced the EOS token.

4 Experiments

4.1 Experimental setup

For training data, we use two small corpora, one English and one German. The English corpus, the Chicago Rhyming Poetry Corpus,[4] has 15,672 training and 1,979 development quatrains, and the German corpus is a small curated subset of Textgrid[5] with 67,054 training and 6,697 development quatrains. We generate quatrain stanzas instead of full poems for speed reasons. As our network architecture, we use a single hidden layer RNN with 64 hidden units, a dropout of 0.2 and input word embeddings of 300 (trained on poetry data using word2vec (Mikolov et al., 2013)) with an additional character-level word representation of dimensionality 25. As optimizer, we use Adam (Kingma and Ba, 2014) with a learning rate of 0.001; we set the batch size to 25.

In general, we obtained very similar results across our two English and German datasets. We report results only for one language per experimental condition.

4.2 Results

Text length. As a sanity check, we test whether we can generate stanzas of a desired length. We select lines that have between 4 and 14 tokens from the English Chicago corpus. We then calculate a normalized condition c_λ which divides the number of tokens in a line by the maximum number available in the subcorpus, which is 14. The values for c_λ thus range from $4/14$ to 1 in the training data. For evaluation, we let the condition c_λ range from 0 to 1 in steps of $1/10$; we also include the value 2. Note

[4]https://github.com/sravanareddy/rhymedata
[5]https://textgrid.de

that all values below 4/14 ≈ 0.2857 as well as 2 were not seen during training. Figure 3 (top) shows average line length per condition c_λ, averaged over 100 generated lines for any value of c_λ. We see that the value of c_λ and the length of the generated output lines correlate strongly. The average line length for $c_\lambda = 0.3$ is 4, which matches the length of training lines with that value. The model can even extrapolate to unseen values, at least to a certain degree – with a value of 2, the line length exceeds the limit of 100 tokens, indicating overgeneralization, while values smaller than 4/14 are smoothly extrapolated.

Sentiment. We infer sentiment from the corpus with the help of word lists: each quatrain receives a rating c_σ as previously defined which we then divide by the maximum sentiment rating of the corpus, normalizing the values to a range of −1 (for very negative sentiment) to 1 (for very positive sentiment). After training, we generate 100 quatrains for each c_σ ranging from −1 to +1 in steps of 1/10. Additionally, we add the extreme values of −2 and +2, which were never seen during training. Figure 3 (right) shows the average sentiment rating value in the generated test data as a function of the c_σ condition. The average rating of the sentiment in the test data increases linearly with c_σ. The model can even extrapolate to the unseen extreme values. Table 1 shows examples of quatrains generated with positive and negative c_σ. With a positive c_σ, the model tends to sample stanzas containing positively connotated words like '*himmels*' (heaven), '*liebe*' (love), and '*kuesst*' (kisses), while a negative c_σ increases the selection of negatively connotated words like '*schwache*' (weak), '*angst*' (fear), and '*heuchler*' (hypocrite). Apparently, the model has understood the semantics of positive and negative sentiment on the sole basis of examples.

Alliteration. In order to control the alliteration level in the generated quatrains, we count how many alliterating sequences there are in a quatrain, as described in Section 3. To normalize these values, we divide them by the maximum number of alliterating sequences in the quatrains of the corpus and add the normalized values as conditional information c_α. We then generate 100 quatrains for each c_α ranging from 0 to 1 in steps of 1/10. Figure 4 (left) shows that the alliteration level in the test data indeed increases as c_α increases. However, Figure 4 (right) shows that increasing c_α for the test poems has

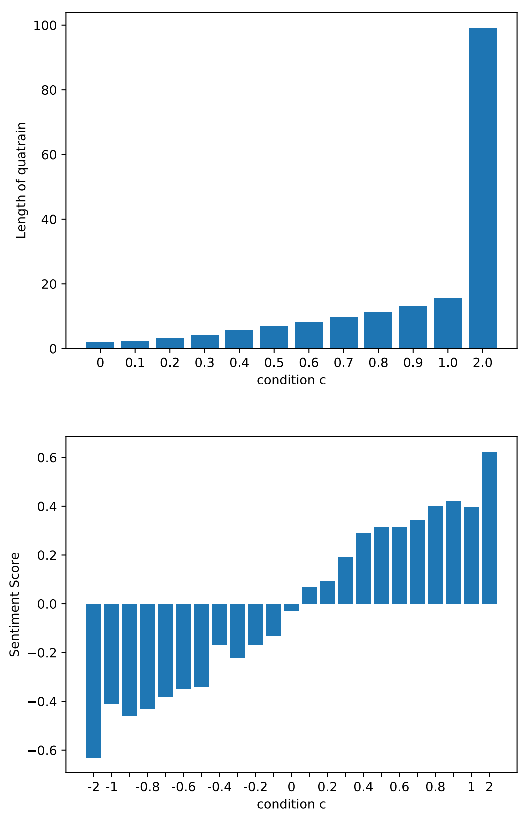

Figure 3: Conditioning on length (c_λ, top) and polar sentiment (c_σ, bottom). The input condition (a real number) is shown on the x axis and the average value measured in the generated output is on the y axis.

the effect that the model generates longer stanzas; when we further normalize the degree of alliteration measured in the output by the number of tokens in the output, then the correlation between the input condition and the measured normalized alliteration degree in the output disappears. Thus, the model apparently does not understand the phenomenon of alliteration, but instead increases the number of words generated: the longer a poem, the more likely it is to (coincidentally) contain instances of alliteration. We also experimented with the input condition $c_{\alpha_{\text{norm}}}$, which measures the degree of alliteration *per token* in a poem, to see if the models would understand the phenomenon of alliteration when they cannot rely on length. Figure 5 (left) shows that this is not the case. There is no clearly visible effect of the input condition on the measured normalized alliteration degree in the generated test data.

Rhyme. We similarly experiment with internal rhymes within a verse and end rhymes within a quatrain. We consider all possible pairs of distinct

ein zeuge o der schoenen du fuer die schuldlos buch bluehend dein schoenstes und noch sieht ein licht jedwedes blatt zum joche ein tor	schwerer streit und weinend flehn vom letzten wort mit farb ergeben hirn und bluete in die der natur ist freude ist des ihm thut
in liebe die streng umgekehret eins sei mein ewig zugethan erfuelle des himmels wieder dass das mit kuesst den lauf	komm doch er sey der heuchler sei nicht dann im spott du solltest eh mir geschrieben die schwache angst

Table 1: Examples of automatically generated sentiment-conditioned quatrains using the sentiment conditions $c_\sigma = +1$ (left) and $c_\sigma = -1$ (right).

words in a verse and feed them into the rhyme classifier described in Section 3, which returns a cosine distance for each pair. If the distance is smaller than a threshold of 0.2, the pair is classified as rhyming. The number of rhyming word pairs in a quatrain is normalized by division by the maximum number of rhyming word pairs found in the quatrains of the corpus. The results are values between 0 (for no rhyming word pair) and 1 (for quatrains with the most rhyming word pairs). Since also the degree of rhyming is positively correlated with the number of tokens in a stanza, we further divide these values by the number of tokens of the respective quatrain, then scale them again to a range between 0 and 1. We denote this rhyme level as $c_{\rho_{\text{norm}}}$. As before, we let $c_{\rho_{\text{norm}}}$ range from 0 to 1 in steps of $1/10$ and generate 100 quatrains for each value. Figure 5 (right) shows the fraction of rhyming words among all generated words in the output as a function of $c_{\rho_{\text{norm}}}$. Similar to the results of conditioning on alliterations, there is no positive correlation of the measured output rhyme level and the input rhyme degree condition when stanza length is accounted for.

4.3 Discussion

In our experiments, we found that end-to-end learning of word-level properties such as sentiment and poem length works – the models successfully learn to generate longer or shorter, and more positive or negative poems, when we condition them on these variables. This means that they have understood the semantics of these conditions on the sole basis of examples – i.e., without the need to explicitly encode the conditions into the model architecture. In contrast, conditioning on sublexical phenomena such as rhyming and alliteration was not successful. Instead, we found that the models learned dataset artifacts and biases – in particular, they learned to generate longer stanzas, which coincidentally have higher degrees of rhyming and alliteration. When controlling for this factor, we could not find evidence that the models learn rhyming or alliteration from examples alone. There are several possible explanations for this finding:

Dataset size. All our datasets were relatively small in size. It is unclear how model results depend on dataset size in terms of the conditioning variables considered here. In terms of the fluency, coherence, etc. of the generated texts, there is clear evidence that size improves model output – see our discussion of GPT-2 in Section 4.4.

Frequency of phenomena. On a per-token level, very few words rhyme or alliterate: for example, less than 3% of tokens in our datasets are part of an alliteration. This makes it harder to detect the meaning of these phenomena from examples alone. Also, very few stanzas have alliteration or rhyming levels that exceed 0.5, meaning that the model is in 'extrapolation mode' for most of them. A possible remedy for this would be to artificially augment the training data with stanzas having large degrees of alliteration or rhyming.

Word-level vs. sublexical phenomena. We note that sentiment is mostly expressed on the word level and rhyme and alliteration are expressed sublexically (i.e., on the sub-word-level). As RNNs (and other text generation models) produce lexical tokens (one after the other), they may be more naturally suited for the former types of phenomena.

Size of character-level representation. For computational reasons, we kept our RNN language models small, with few hidden units (64) and a small dimensionality of character-level representations for words (25). It is possible that the model simply did not have enough capacity to store the

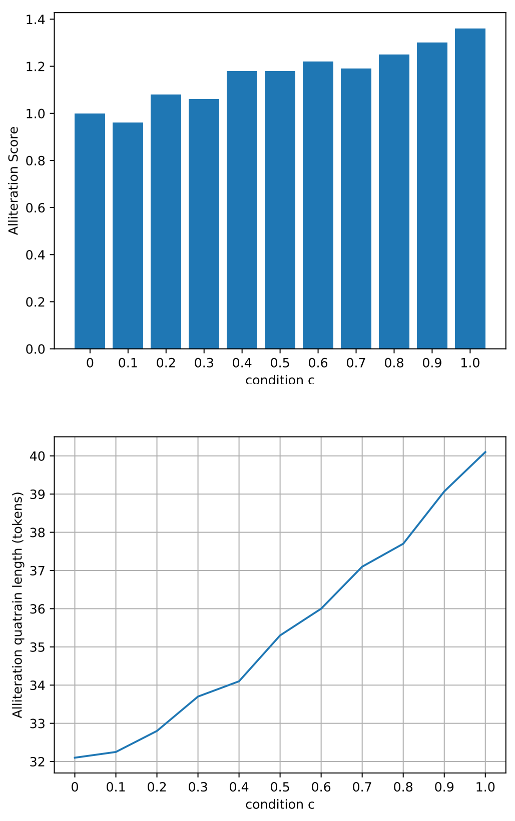

Figure 4: Top: Conditioning on degree of alliteration (c_α) on the English Chicago corpus data. The input condition (a real number) is shown on the x axis and the average value measured in the generated output is reported on the y axis. Bottom: Increasing c_α generally leads to an increase in the length of the generated poems.

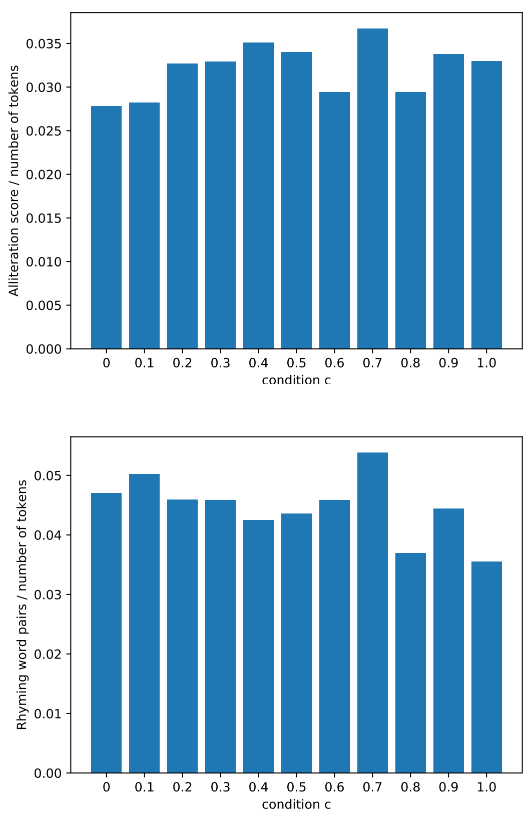

Figure 5: Conditioning on normalized alliteration degree ($c_{\alpha_{\text{norm}}}$, top) and normalized rhyme degree ($c_{\rho_{\text{norm}}}$, bottom) on the English Chicago corpus data. The input condition (a real number) is shown on the x axis and the average value measured in the generated output is reported on the y axis.

relevant information about the sublexical phenomena of interest. However, it must also be said that increasing model capacity runs the risk of severe overfitting when the training data size is small, as is the case for our datasets.

Heterogeneity of poetry data. It is possible that the heterogeneity of the poetry data (with some stanzas having high and others very low degrees of rhyming and alliteration) confuses the models (Hopkins and Kiela, 2017), particularly when some of these phenomena are very infrequent.

While the explanations above are specific to our task of style-conditioned poetry generation, the more general problem of deep models learning dataset artifacts or shortcuts that exploit low-level statistical regularities has often been remarked upon in recent years: Such observations have been made for natural language inference (Poliak et al., 2018) and argumentation mining (Niven and Kao, 2019), among many other tasks.

In the following subsection, we re-evaluate our findings using a different model architecture trained on more data.

4.4 Using GPT-2 instead of an RNN

In order to explore the capabilities of a transformer model over a generic RNN, we also tuned style-conditioned GPT-2 models on poetry. The main advantage of GPT-2 (based on transformer networks) over standardly trained RNNs is their transfer learning ability: While a standardly trained RNN needs to learn language modelling capabilities only from small amounts of poetry data, GPT-2 is already equipped with general language modelling abilities, obtained from having seen billions of tokens of written English. This makes the output of GPT-2 much more semantically coherent, fluent, and grammatically correct. Nevertheless, we believe that even GPT-2 may have the same problems learning poetry-specific characteristics such as rhyme, metre, and alliteration that our RNN

And much, in fact, this lesser world can show
Of grief and crime that in the greater grow
"You saw," said George, "in that still-hated school
How the meek suffer, how the haughty rule

Table 2: Example pseudo-quatrain in our data for GPT-2, with (predicted) rhyme scheme 'aabb', negative sentiment, and 18th-century time period.

had, presumably because these characteristics have low statistical support in the pre-trained data (e.g., Wikipedia) and for the other reasons named above. We therefore conducted two sets of experiments concerning the generation of style-conditioned poetry where the conditions include rhyme, sentiment, and the time epoch of the poetry to be produced, in English poetry.

Rhyme, sentiment, time epoch. We used a GPT-2 model[6] trained on a larger English poetry corpus[7] that additionally contains information about the time period in which the poems were written; we used pseudo-quatrains (any consecutive sequence of four lines), amounting to over 770K unique examples.[8] A sample pseudo-quatrain from the data is shown in Table 2.

Since GPT-2 can more naturally be run with discrete input text, we trained the model on the 24 different rhyming patterns ('abab', 'abcd', etc.) that we found in the corpus, three sentiment values (positive, negative, neutral), and five different time periods (16th to 20th centuries). We determine the sentiment of a newly generated poem as above but using thresholding (e.g., a poem has a positive sentiment if it has more positive than negative terms), rhyming is predicted as for the RNN, and the time period is learned on the annotated data using a multi-layer perceptron. We tested various hyperparameter configurations and in all cases observed the same result: the model has great difficulty learning rhyming, but can capture the other two stylistic features. The last two phenomena are informed largely by lexical choice while rhyming is informed by sublexical information. To illustrate, with the default configuration, we see accuracy of 7.5% for rhyming (with a random baseline performance of 4.2%), 63% for sentiment (random baseline of 33%) and 63% for time period (random performance of 20%) when generating about 5,000 poems with the desired styles. Even though the model is not entirely at the random guessing baseline for rhyming, it does not perform much above it and is considerably below the majority baseline of 22% (always producing a poem with 'aabb' rhyme style). Also, the rhyme scheme predicted best by far is 'abcd' involving no rhyming. This requires the model not to rhyme, which the model then also satisfies. Here, the model satisfies 53% of all such user constraints, while it satisfies only 0.4% of more complex constraints such as 'aabb'.

[6]https://huggingface.co/gpt2

[7]https://github.com/tnhaider/english-gutenberg-poetry

[8]Our model is available at https://github.com/thekhangnguyen/poetry-gen.

5 Concluding remarks

Traditional, 'formal' poetry is characterized by intricate form that often expresses itself on the sound level: among the key features are rhyme, metre, and alliteration. Computational treatments typically handle these phenomena by hard-coding them as in purely symbolic AI, by exploiting dataset-specific properties (Lau et al., 2018), or by having models pay special attention to the phenomena in question (e.g., making the models concentrate on the character-level structure of line-ending words) (Jhamtani et al., 2019). However, as we show, current models still seem unable to learn sound-level stylistic aspects from examples alone, without priming them, and *even when we specifically condition on such features*. This raises the question of where this discrepancy between human and machine text processing (Eger et al., 2019) comes from and how it can be addressed *in the general case*. We envision several solutions for future work: It is possible that end-to-end modelling of phonetic stylistic features is an *emergent* phenomenon that appears once dataset sizes are large enough; it is also possible that datasets need to be sufficiently clean (e.g., containing only very few different rhyming schemes); or that new model architectures, which do not operate on words or sublexical units, are required (Xue et al., 2021).

Acknowledgments

This work has been supported by the Austrian Science Fund (FWF) under project M 2625-N31 and by the German Research Foundation as part of the Research Training Group Adaptive Preparation of Information from Heterogeneous Sources (AIPHES) at the Technische Universität Darmstadt under grant GRK 1994/1.

References

Rajat Agarwal and Katharina Kann. 2020. Acrostic poem generation. In *Proc. EMNLP*, pages 1230–1240.

Manex Agirrezabal, Iñaki Alegria, and Mans Hulden. 2016. Machine learning for metrical analysis of English poetry. In *Proc. COLING*, pages 772–781.

Timo Baumann, Hussein Hussein, and Burkhard Meyer-Sickendiek. 2018. Style detection for free verse poetry from text and speech. In *Proc. COLING*, pages 1929–1940.

M. H. Beissinger. 2012. Oral poetry. In R. Greene *et al.*, editor, *The Princeton Encyclopedia of Poetry and Poetics*, 4th edition, pages 978–81. Princeton University Press.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel Ziegler, Jeffrey Wu, Clemens Winter, Chris Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901. Curran Associates, Inc.

Simon Colton, Jacob Goodwin, and Tony Veale. 2012. Full-FACE poetry generation. In *Proc. 3rd International Conference on Computational Creativity*, pages 95–102.

Steffen Eger, Gözde Gül Şahin, Andreas Rücklé, Ji-Ung Lee, Claudia Schulz, Mohsen Mesgar, Krishnkant Swarnkar, Edwin Simpson, and Iryna Gurevych. 2019. Text processing like humans do: Visually attacking and shielding NLP systems. In *Proc. NAACL*, volume 1, pages 1634–1647.

Alex Estes and Christopher Hench. 2016. Supervised machine learning for hybrid meter. In *Proc. 5th Workshop on Comp. Ling. for Literature*, pages 1–8.

Nigel Fabb and Morris Halle. 2010. *Meter in Poetry: A New Theory*. Cambridge University Press.

Jessica Ficler and Yoav Goldberg. 2017. Controlling linguistic style aspects in neural language generation. In *Proc. Workshop on Stylistic Variation*, pages 94–104.

Vikas Ganjigunte Ashok, Song Feng, and Yejin Choi. 2013. Success with style: Using writing style to predict the success of novels. In *Proc. EMNLP*, pages 1753–1764.

Pablo Gervás. 2001. An expert system for the composition of formal Spanish poetry. *Knowledge-based Syst.*, 14(3–4):181–188.

Marjan Ghazvininejad, Yejin Choi, and Kevin Knight. 2018. Neural poetry translation. In *Proc. NAACL*, volume 2, pages 67–71.

Marjan Ghazvininejad, Xing Shi, Jay Priyadarshi, and Kevin Knight. 2017. Hafez: An interactive poetry generation system. In *Proc. ACL (System Demonstrations)*, pages 43–48.

Hugo Gonçalo Oliveira. 2013. PoeTryMe: A versatile platform for poetry generation. In *Proc. Workshop on Computational Creativity, Concept Invention, and General Intelligence*, volume 1-2012, pages 21–26.

Hugo Gonçalo Oliveira, Raquel Hervás, Alberto Díaz, and Pablo Gervás. 2017. Multilingual extension and evaluation of a poetry generator. *Nat. Lang. Eng.*, 23(6):929–967.

Erica Greene, Tugba Bodrumlu, and Kevin Knight. 2010. Automatic analysis of rhythmic poetry with applications to generation and translation. In *Proc. EMNLP*, pages 524–533.

Thomas Haider and Steffen Eger. 2019. Semantic change and emerging tropes in a large corpus of New High German poetry. In *Proc. 1st International Workshop on Computational Approaches to Historical Language Change*, pages 216–222.

Thomas Haider, Steffen Eger, Kim Evgeny, Roman Klinger, and Winfried Menninghaus. 2020. PO-EMO: Conceptualization, annotation, and modeling of aesthetic emotions in German and English poetry. In *Proc. LREC*, pages 1652–1663.

Thomas Haider and Jonas Kuhn. 2018. Supervised rhyme detection with Siamese recurrent networks. In *Proc. LaTeCH-CLfL*, pages 81–86.

Mika Hämäläinen and Khalid Alnajjar. 2019. Let's FACE it. Finnish poetry generation with aesthetics and framing. In *Proc. 12th International Conference on Natural Language Generation*, pages 290–300.

Aurélie Herbelot. 2015. The semantics of poetry: A distributional reading. *Digit. Scholarsh. Humanit.*, 30(4):516–531.

J. Berenike Herrmann, Karina van Dalen-Oskam, and Christof Schöch. 2015. Revisiting style, a key concept in literary studies. *J. Lit. Theory*, 9(1):25–52.

Jack Hopkins and Douwe Kiela. 2017. Automatically generating rhythmic verse with neural networks. In *Proc. ACL*, volume 1, pages 168–178.

Minqing Hu and Bing Liu. 2004. Mining and summarizing customer reviews. In *Proc. ACM SIGKDD*, pages 168–177.

Roman Jakobson. 1960. Linguistics and poetics. In *Style in Language*, pages 350–377. MIT Press.

Harsh Jhamtani, Sanket Vaibhav Mehta, Jaime Carbonell, and Taylor Berg-Kirkpatrick. 2019. Learning rhyming constraints using structured adversaries. In *Proc. EMNLP/IJCNLP*, pages 6025–6031.

Justine T. Kao and Dan Jurafsky. 2015. A computational analysis of poetic style: Imagism and its influence on modern professional and amateur poetry. *Linguist. Issues Lang. Technol.*, 12(3):1–31.

Vaibhav Kesarwani, Diana Inkpen, Stan Szpakowicz, and Chris Tanasescu. 2017. Metaphor detection in a poetry corpus. In *Proc. LaTeCH-CLfL*, pages 1–9.

Diederik P. Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv e-prints*, 1412.6980.

Reinhard Kneser and Hermann Ney. 1995. Improved backing-off for *m*-gram language modeling. In *Proc. 1995 International Conference on Acoustics, Speech, and Signal Processing*, volume 1, pages 181–184.

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 *Proc. ACL*, volume 1, pages 1948–1958.

Bei Liu, Jianlong Fu, Makoto P. Kato, and Masatoshi Yoshikawa. 2018. Beyond narrative description: Generating poetry from images by multi-adversarial training. In *Proc. 26th ACM International Conference on Multimedia*, pages 783–791.

Enrique Manjavacas, Mike Kestemont, and Folgert Karsdorp. 2019. Generation of hip-hop lyrics with hierarchical modeling and conditional templates. In *Proc. 12th International Conference on Natural Language Generation*, pages 301–310.

Hisar Maruli Manurung, Graeme Ritchie, and Henry Thompson. 2000. Towards a computational model of poetry generation. In *Proc. AISB Symposium on Creative and Cultural Aspects and Applications of AI and Cognitive Science*, pages 79–86.

Winfried Menninghaus, Valentin Wagner, Eugen Wassiliwizky, Thomas Jacobsen, and Christine A Knoop. 2017. The emotional and aesthetic powers of parallelistic diction. *Poetics*, 63:47–59.

Rada Mihalcea and Carlo Strapparava. 2006. Learning to laugh (automatically): Computational models for humor recognition. *Comp. Intel.*, 22(2):126–142.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv e-prints*, 1301.3781.

Timothy Niven and Hung-Yu Kao. 2019. Probing neural network comprehension of natural language arguments. In *Proc. ACL*, pages 4658–4664.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. BLEU: A method for automatic evaluation of machine translation. In *Proc. ACL*, pages 311–318.

Adam Poliak, Jason Naradowsky, Aparajita Haldar, Rachel Rudinger, and Benjamin Van Durme. 2018. Hypothesis only baselines in natural language inference. In *Proc. 7th Joint Conference on Lexical and Computational Semantics*, pages 180–191.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. Article on openai.com.

Sravana Reddy and Kevin Knight. 2011. Unsupervised discovery of rhyme schemes. In *Proc. ACL*, pages 77–82.

Nils Reimers and Iryna Gurevych. 2017. Reporting score distributions makes a difference: Performance study of LSTM-networks for sequence tagging. In *Proc. EMNLP*, pages 338–348.

Ines Reinig and Ines Rehbein. 2019. Metaphor detection for German poetry. In *Proc. KONVENS*, pages 149–160.

Pablo Ruiz, Clara Martínez Cantón, Thierry Poibeau, and Elena González-Blanco. 2017. Enjambment detection in a large diachronic corpus of spanish sonnets. In *Proc. LaTeCH-CLfL*, pages 27–32.

Victor Shlovsky. 1965. Art as technique. In Lee T. Lemon and Marion J. Reis, editors, *Russian Formalist Criticism: Four Essays*, pages 3–24. University of Nebraska Press.

Ilya Sutskever, James Martens, and Geoffrey Hinton. 2011. Generating text with recurrent neural networks. In *Proc. 28th International Conference on Machine Learning*, pages 1017–1024.

Tim Van de Cruys. 2020. Automatic poetry generation from prosaic text. In *Proc. ACL*, pages 2471–2480.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems 30*, pages 5998–6008. Curran Associates.

Ulli Waltinger. 2010. GermanPolarityClues: A lexical resource for German sentiment analysis. In *Proc. LREC*, pages 1638–1642.

Kento Watanabe, Yuichiroh Matsubayashi, Satoru Fukayama, Masataka Goto, Kentaro Inui, and Tomoyasu Nakano. 2018. A melody-conditioned lyrics language model. In *Proc. NAACL*, volume 1, pages 163–172.

Linting Xue, Aditya Barua, Noah Constant, Rami Al-Rfou, Sharan Narang, Mihir Kale, Adam Roberts, and Colin Raffel. 2021. Byt5: Towards a token-free future with pre-trained byte-to-byte models.

Xingxing Zhang and Mirella Lapata. 2014. Chinese poetry generation with recurrent neural networks. In *Proc. EMNLP*, pages 670–680.