Neurosymbolic Deep Generative Models for Sequence Data with Relational Constraints

Anonymous authors
Paper under double-blind review

Abstract

Recently, there has been significant progress designing deep generative models that generate realistic sequence data such as text or music. Nevertheless, it remains difficult to incorporate high-level structure to guide the generate process. We propose a novel approach for incorporating structure in the form of relational constraints between different subcomponents of an example (e.g., lines of a poem or measures of music). Our generative model has two parts: (i) one model to generate a realistic set of relational constraints, and (ii) a second model to generate realistic data satisfying these constraints. To train model (i), we propose a novel program synthesis algorithm that infers the relational constraints present in the training data, and then train the models based on the resulting relational constraints. In our experiments, we show that our approach significantly improves over state-of-the-art approaches in terms of capturing high-level structure in the data, while performing comparably or better in terms of low-level structure.

1 Introduction

Over the past few years, there has been tremendous progress in designing deep generative models for generating sequence data such as natural language (Vaswani et al., 2017) or music (Huang et al., 2019). These approaches leverage the vast quantities of data available in conjunction with unsupervised and self-supervised learning to learn probabilistic models of the data; then, new examples can be generated by sampling from these models, with the possibility of conditioning on initial elements of the sequence.

Despite this progress, a key challenge facing deep generative models is the difficulty incorporating high-level structure into the generated examples—e.g., rhyming and meter across lines of a poem, or repetition across measures of a piece of music. The ability to capture high-level structure is important for improving the quality of the generated data, especially in low-data regimes where only small numbers of examples are available—intuitively, knowledge of the structure compresses the amount of information that the generative model has to learn. Furthermore, *explicit* representations of structure—i.e., in a symbolic way rather than implicitly in a vector embedding—can have the added benefit that users can modify the structure to guide the generative process.

Recently, Young et al. (2019) proposed a technique called *neurosymbolic generative models* for incorporating high-level structure into image generation, focusing on simple 2D repeating patterns in images of building facades (e.g., repeating windows). The basic idea is to leverage *program synthesis* to extract structure from data—in particular, given an example image x, they devise an algorithm $\mathcal{A}$ that extracts a program $c = \mathcal{A}(x)$ that represents the set of 2D repeating patterns present in training examples x. Then, using the pairs (x, c), they train two generative models: (i) a model $p_\phi(c)$ that generates a program, and (ii) a model $p_\theta(x \mid c)$ that generates an image that contains the structure represented by c.

However, their approach is heavily tailored to the image domain in several ways. First, their representation of structure is geared towards relatively simple patterns occurring in images of building facades. In addition, their algorithm $\mathcal{A}$ is specifically designed to extract this kind of program from an input image, as are their models $p_\phi(c)$ for generating programs and $p_\theta(x \mid \pi)$ for generating images conditioned on the program.

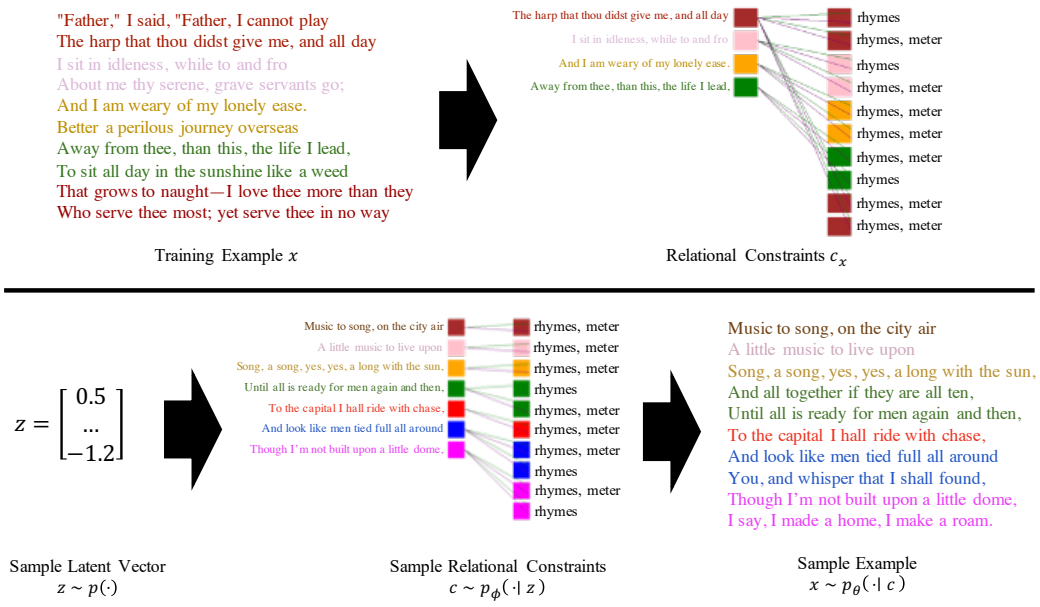

Figure 1: Top: Process for training. For each training example x, our algorithm uses program synthesis to infer the relational constraints $c_x = \mathcal{A}(x)$ present in x. Then, it (i) uses c_x to train $p_\phi(c) = \mathbb{E}_{z \sim p(\cdot)}[p_\phi(c \mid z) \cdot p(z)]$, and (ii) uses (c_x, x) to train $p_\theta(x \mid c)$. Bottom: Process for generating a sample x from the learned models $p_\phi(c \mid z)$ and $p_\theta(x \mid c)$. Lines with the same prototype are shown in the same color.

We represent relational constraints c_x present in an example x by relating each subcomponent w of a given example x with a *prototype* $\tilde{w}$, which can intuitively be thought of as the "original" subcomponent from which w is constructed. In particular, the relationship between w and $\tilde{w}$ is labeled with a set of relations R, which encodes the constraint that w and $\tilde{w}$ should satisfy relation r for each $r \in R$. Importantly, while each subcomponent is associated with a single prototype, each prototype may be associated with multiple subcomponents. As a consequence, different subcomponents associated with the same prototype are related in some way. This representation is compact, only requiring linearly many constraints in the number of subcomponents in x (assuming the number of prototypes is constant). Intuitively, compactness ensures the representation both generalizes well and is easy to generate.

Then, we design a program synthesis algorithm that can extract an optimal representation of the structure present in a training example x. We show how to express the synthesis problem as a constrained combinatorial optimization problem, which we solve using an SMT solver Z3 (De Moura & Bjørner, 2008). Next, we represent c as a sequence, and design $p_\phi(c)$ to be a specialized sequence VAE. Finally, we propose three possible designs of $p_\theta(x \mid c)$ based on trying to identify an example x that is realistic (e.g., according to a pretrained generative model $p_\theta(x)$) while simultaneously satisfies the constraints encoded by c.

We evaluate our approach on two tasks: poetry generation, where the relational constraints include rhyming lines or lines with shared meter, and music generation, where the relational constraints include equality in terms of pitch or rhythm, that one measure is a transposition of another (i.e., pitches shifted up or down by a constant amount), etc. We demonstrate that examples generated using our approach contain significantly more structure compared to state-of-the-art baselines while achieving similar or better performance realism as measured by a pretrained deep generative model. Importantly, this metric of "realism" is incomplete, since the pretrained deep generative model does not capture high-level structure, yet capturing structure can still help improve realism according to this model. Thus, our approach not only improves high-level structure, but can improve the entire generation process.

Finally, we also perform a qualitative evaluation where we show how the user can modify the high-level to generate examples that satisfy additional desired constraints. This ability demonstrates an important feature of our approach—i.e., that the user can guide the generative process by modifying the relational constraints as desired.

Example. Figure 1 illustrates how our approach is applied to generate poetry. During training, our approach uses program synthesis to infer relational constraints c_x present in the examples x, and uses both x and c_x to train the generative models. Here, c_x is a bipartite graph, where the LHS vertices are *prototypes*, and the RHS vertices correspond to lines of x. Each vertex on the right is connected to exactly one prototype, and is labeled with constraints on how it should relate to its prototype. To generate new examples, it first samples relational constraints c, and then samples an example x that satisfies c—i.e., we need to choose a line to fill each RHS node in a way that the line satisfies the relations with its prototype. Furthermore, a user can modify the sampled constraint c to guide the generative process. Thus, our approach enables users to flexibly incorporate domain knowledge on the high-level structure of the data into the generative process, both in terms of the relational constraints included and by allowing them to modify the generated relational constraints.

Related work. There has been recent interest in leveraging program synthesis to improve machine learning. For instance, it has been applied to unsupervised learning of latent structure in drawings (Ellis et al., 2015) and to reinforcement learning (Verma et al., 2018). These techniques have benefits such as improving interpretability Verma et al. (2018); Ellis et al. (2020), enabling learning from fewer examples (Ellis et al., 2015), generalizing more robustly (Inala et al., 2019), and being easier to formally reason about (Bastani et al., 2018). More recently, there has been work leveraging program synthesis in conjunction with deep learning, where the DNN handles perception and program synthesis handles high-level structure (Ellis et al., 2017), including work in the lifelong learning setting (Valkov et al., 2018). In contrast to these approaches, our focus is on generative models. In particular, we extend recent work leveraging these ideas in the setting of image generation to incorporating high-level relational structure into sequence generation tasks (Young et al., 2019).

There has been work using deep learning to generate music (Huang et al., 2019; OpenAI, 2019) and poetry (Liao et al., 2019); we show in our experiments that these approaches have difficulty generating realistic high-level structure. Some approaches to poetry generation incorporate expert-provided constraints such as rhyme and meter (Lau et al., 2018); unlike our approach, they cannot automatically learn and generate these constraints from data.

2 Background on Neurosymbolic Generative Models

Consider the problem of learning a generative model given training data from the underlying distribution. Given training examples $x_1, ..., x_k \sim p^*$, our goal is to learn a generative model $p_\theta \approx p^*$ from which we can draw additional samples $x \sim p_\theta$. We consider sequence data—i.e., an example $x \in \mathcal{X}$ is a sequence $x = (w_1, ..., w_m) \in \mathcal{W}^m$.[1] For example, each *subcomponent* w may be a line of a poem or a measure of music, and x may be a poem or song.

We are interested in domains where likely examples satisfy latent relational constraints $c \in \mathcal{C}$ over the subcomponents. For instance, c may say that two measures w_i and w_j of x start with the same series of pitches, or two lines w_i and w_j of x rhyme. We assume given a set of relations $\mathcal{R}$ (e.g., $r \in \mathcal{R}$ might be "rhyme" or "equal"), and a function $f : \mathcal{W} \times \mathcal{W} \times \mathcal{R} \to \mathbb{B}$ (where $\mathbb{B} = \{0, 1\}$) such that $f(w, w', r)$ indicates whether w and w satisfy relation r. Then, c is a compact representation of the relations present in an input x. We describe the structure of c in detail in Section 3.1; for now, the approach we describe works for any choice of c. In particular, we build on *neurosymbolic generative models* (Young et al., 2019), where c is itself generated based on a latent value $z \in \mathcal{Z}$—i.e.,

$$p_{\theta,\phi}(x) = \int \sum_{r \in \mathcal{R}} p_\theta(x \mid c) \cdot p_\phi(c \mid z) \cdot p(z) dz.$$

[1] We use a fixed m to simplify our exposition; our approach trivially extends to variable m.

Then, Young et al. (2019) considers the variational distribution

$$q_{\tilde{\phi}}(c, z \mid x) = q_{\tilde{\phi}}(z \mid c) \cdot q(c \mid x) \qquad \text{where} \qquad q(c \mid x) = \delta(c - c_x).$$

Here, δ is the Dirac delta function, and c_x is a single representative associated with x. In particular, c_x is generated from x using a program synthesis algorithm David & Kroening (2017)—i.e., an algorithm $\mathcal{A}$ that takes as input an example x and outputs a program $c = \mathcal{A}(x)$ encoding the relational constraints present in x. Next, Young et al. (2019) derive an evidence lower bound

$$\log p_{\theta,\phi}(x) \geq \log p_\theta(x \mid c_x) + \mathbb{E}_{q_{\tilde{\phi}}(z \mid c_x)}[\log p_\phi(c_x \mid z)] - D_{\mathrm{KL}}(q_{\tilde{\phi}}(z \mid c_x) \parallel p(z)). \tag{1}$$

where D_{KL} is the KL divergence and H is the information entropy. The first term of (1) is the log-likelihood of a generative model predicting the probability of example x given relational structure c_x, and the second and third terms form the loss of a variational autoencoder (VAE) $p_\phi(c \mid z)$ and $q_{\tilde{\phi}}(z \mid c)$ (Kingma & Welling, 2019). In summary, this approach separately learns (i) a VAE to generate c given z, and (ii) a generative model to generate x given c; the latter can be a second VAE or a generative adversarial network (GAN) (Goodfellow et al., 2014). This approach is called *synthesis-guided generative models (SGM)* since it uses program synthesis to guide training.

To leverage this framework, we have to instantiate (i) the space of relational constraints $\mathcal{C}$, (ii) the synthesis algorithm $\mathcal{A} : \mathcal{X} \to \mathcal{C}$ used to extract a program encoding the structure of x, and (iii) the architectures of $p_\phi(c \mid z)$, $q_{\tilde{\phi}}(z \mid c)$, and $p_\theta(x \mid c)$. In previous work, Young et al. (2019) used heuristics specific to the the image domain to achieve these goals—in particular, they used (i) simple equality constraints on sub-regions of the image designed to capture 2D repeating patterns, (ii) a custom synthesis algorithm that greedily adds constraints in the data to the program, and (iii) a representation of c_x as an image, in which case p_θ is a generative model over images, and $p_\phi, q_{\tilde{\phi}}$ based on an encoding of c as a fixed-length vector.

We design a synthesis algorithm that expresses the synthesis problem as a constrained combinatorial optimization problem, which it solves using Z3 (De Moura & Bjørner, 2008). In terms of (iii), our programs encode declarative constraints rather than imperative renderings, so the previous architectures of p_ϕ, and $q_{\tilde{\phi}}$ cannot be used. Instead, we use expert domain-specific heuristics, transformers (Vaswani et al., 2017), or graph neural networks (GNNs) (Kipf & Welling, 2017) for p_ϕ and $q_{\tilde{\phi}}$. For p_θ, we propose several methods for imposing the constraints encoded by c when generating an example x.

3 Relational Constraints for Sequence Data

In this section, we describe how we represent relational constraints r, as well as our algorithm $\mathcal{A}$ for synthesizing the relational constraints $c_x = \mathcal{A}(x)$ present in an example sequence x.

3.1 Graph Representation of Relational Constraints

Recall that our generative model operates by first generating a relational program c, and then generating an example x that satisfies c. Thus, we need to design relational programs c that encode constraints on the structure of an example x. Our programs c encode a set of *relational constraints*, each of which imposes a constraint that subcomponents of x should have certain kinds of relations. We begin by describing the structure of a single relational constraint, and then describe how c encodes a set of relational constraints.

First, a *relational constraint* $\phi \in \Phi = \mathcal{W} \times \mathcal{I} \times \mathcal{R}$, where $\mathcal{I} = \{1, ..., m\}$, is a tuple $\phi = (\tilde{w}, i, r)$; we call $\tilde{w} \in \mathcal{W}$ a *prototype subcomponent*. An example x *satisfies* ϕ (denoted $x \models \phi$) if $f(\tilde{w}, w_i, r) = 1$, where w_i is the ith subcomponent of x. In other words, ϕ says that the ith subcomponent w_i of x should have relation r with prototype subcomponent $\tilde{w}$. We can interpret ϕ as a function $\phi : \mathcal{X} \to \mathbb{B}$, where $\phi(x) = 1$ if x satisfies ϕ and $\phi(x) = 0$ otherwise.

Next, we represent c as an undirected labeled bipartite graph $c = (\tilde{V}, V, E)$ with vertices $\tilde{V}$ and V and edges $E \subseteq \tilde{V} \times V \times \mathcal{L}$, where $\mathcal{L}$ are the labels. The vertices $\tilde{w} \in \tilde{V}$ are

prototype subcomponents $\tilde{w} \in \mathcal{W}$; equivalently, they may be vector embeddings of prototype subcomponents. The vertices $i \in V = \{1, ..., m\}$ are the indices of subcomponents in x. The edges $e \in E$ are tuples $e = (\tilde{w}, i, R)$, where $R \subseteq \mathcal{R}$ is a set of relations. We impose the constraint that each $v \in V$ is part of a single edge $(\tilde{w}, v, R)$ (though $\tilde{v} \in \tilde{V}$ may be part of multiple edges). Finally, c encodes the set of relational constraints

$$\Phi_c = \{(\tilde{w}, i, r) \mid (\tilde{w}, i, R) \in E \wedge r \in R\}.$$

In other words, c includes the relational constraint that each subcomponent w_i of x should have all relations $r \in R$ with prototype $\tilde{w}$, where v is connected to $\tilde{w}$.

3.2 Synthesizing Relational Constraints

Recall that when training our generative model, we need to design a program synthesis algorithm $\mathcal{A}$ that synthesizes a relational program $c_x = \mathcal{A}(x)$ that best encodes the latent relational constraints present in each training example x. A key question is where the prototypes come from. We simply choose the prototypes $\tilde{w}$ to be actual subcomponents in x. Thus, c_x encodes that subcomponents of x are each related to one of a small number of distinguished subcomponents of x. We formulate the problem of synthesizing c_x as a constrained optimization problem, which we describe below.

Optimization variables. The variables are a binary vector $H \in \mathbb{B}^m$ and a binary matrix $K \in \mathbb{B}^{m \times m}$. Intuitively, H_i indicates whether subcomponent w_i of x is a prototype subcomponent in c, and K_{ij} indicates whether w_i is the prototype for subcomponent w_j.

Constraints. Our optimization problem has the following three constraints:

$$\psi_1 \equiv k_{\min} \leq \sum_{i=1}^{m} H_i \leq k_{\max}, \qquad \psi_2 \equiv \bigwedge_{j=1}^{m} \sum_{i=1}^{m} K_{ij} = 1, \qquad \psi_3 \equiv \bigwedge_{i=1}^{m} \sum_{j=1}^{m} K_{ij} \leq m \cdot H_i.$$

First, ψ_1 says that the number of prototype subcomponents is between $k_{\min}$ and $k_{\max}$. Next, ψ_2 says that every subcomponent w_j corresponds to exactly one prototype subcomponent w_i. Finally, ψ_3 says that for every i, if w_i is the prototype subcomponent of w_j according to K, then it must be a prototype subcomponent according to H as well.

Objective. The objective of our optimization problem is expressed in terms of a precomputed distance matrix $D \in \mathbb{R}^{m \times m}$, where D_{ij} measures the similarity between components w_i and w_j; smaller values indicate a greater degree of similarity. In particular, we define

$$D_{ij} = \frac{1}{|\mathcal{R}|} \sum_{r \in \mathcal{R}} \mathbb{1}(f(w_i, w_j, r) = 0),$$

i.e., D_{ij} is the fraction of relations that are not satisfied by w_i and w_j. Then, our objective (which is to be minimized) has the following three terms:

$$J_1 = \sum_{i,j=1}^{m} K_{ij} \cdot D_{ij}, \qquad J_2 = \sum_{i,j=1}^{m} \left(\prod_{k} K_{ki} \cdot K_{kj} \right) \cdot D_{ij}, \qquad J_3 = -\sum_{i,j=1}^{m} M_i \cdot M_j \cdot D_{ij}.$$

First, J_1 says that subcomponents should be similar to their prototypes. Second, J_2 says that subcomponents should also be similar to other subcomponents that share the same prototype. Third, J_3 says that different prototype subcomponents should be dissimilar.

Optimization problem. Our algorithm $\mathcal{A}$ uses Z3 to solve the optimization problem

$$(H^*, K^*) = \underset{H,K}{\arg\min} \{\lambda_1 \cdot J_1 + \lambda_2 \cdot J_2 + \lambda_3 \cdot J_3\} \qquad \text{subj. to} \qquad \psi_1 \wedge \psi_2 \wedge \psi_3,$$

where $\lambda_1, \lambda_2, \lambda_3 \in \mathbb{R}_{\geq 0}$ are hyperparameters. Finally, to construct c_x, $\mathcal{A}$ chooses

$$\tilde{V} = \{w_i \mid H_i^* = 1\}, \qquad V = \{1, ..., m\}, \qquad E = \{(w_i, j, R_{ij}) \mid K_{ij}^* = 1\},$$

where $R_{ij} = \{r \in \mathcal{R} \mid f(w_i, w_j, r) = 1\}$—i.e., $\tilde{V}$ are the prototype subcomponents according to H^*, E are the edges according to K^*, and R_{ij} are the relations satisfied by w_i and w_j.

4 Neurosymbolic Generative Models with Relational Constraints

In this section, we describe our deep generative model for generating examples x. Recall that our approach proceeds in two steps: (i) generate c, and (ii) generate x given Φ_c. We describe each of these steps in detail below.

4.1 Step 1: Generating Relational Constraints

The first step of our generative model is to generate relational constraints Φ_c using a VAE—i.e., $z \sim p(\cdot)$ and $c \sim p_\phi(\cdot \mid z)$, where $p_\phi(c \mid z)$ is a VAE and $p(z) = \mathcal{N}(z; 0, I)$ is a Gaussian distribution. The main choice is the architecture to use for the VAE. In particular, we consider a representation of c as a sequence $(s_1, ..., s_m)$, where $s_i \in \{0, 1, ..., m\}$ for each i; intuitively, s_i encodes that subcomponent w_i should have the same prototype subcomponent as w_{i-s_i}, or if $s_i = 0$, that w_i corresponds to a new prototype subcomponent.

More precisely, we initialize $\Phi_c = \varnothing$. Then, we generate the sequence $s_i \in \{0, 1, ..., m\}$ and $r_i \in \{0, 1, ..., m\}$ (where r_i is represented as a binary vector of length $n = |\mathcal{R}|$) using an LSTM-VAE. For each i, we generate $(\tilde{w}, i, R_i)$ based on s_i and r_i. If $s_i = 0$, we generate a new prototype subcomponent $\tilde{w}$ using a domain-specific generative model, generate with using another LSTM-VAE, and add $\phi_i = (\tilde{w}_i, i, r_i)$ to Φ_c. If $s_i > 0$, we let $\phi_i = (\tilde{w}_{i-s_i}, i, r_i)$.

4.2 Step 2: Generating Examples Given Relational Constraints

Next, we describe how we implement the second step $p_\theta(x \mid c)$ of our generative model. We propose three approaches for generating x given Φ_c; we give details in Appendix A.

Approach 1: Constrained sampling. We sample values $x \sim p_\theta(\cdot)$ by sequentially sampling $w_i \sim p_\theta(\cdot)$ from a pretrained generative model $p_\theta(w)$. We do so using rejecting sampling at each step—i.e., we sample $w_i \sim p_\theta(\cdot)$ until we find w_i satisfying $f(\tilde{w}, w_i, r) = 1$ for each $(\tilde{w}, i, r) \in \Phi_c$. In addition, to speed up sampling, at each step of sampling w_i (e.g., a word in a line or a pitch in a measure), we eliminate choices that violate Φ_c.

Approach 2: Constraint-aware embeddings. We train a conditional generative model $p_\theta(w_1, ..., w_m \mid c)$ (in the form of a graph convolutional network) that simultaneously generates all m subcomponents in a way that satisfies c, and sample $x = (w_1, ..., w_m) \sim p_\theta(\cdot \mid c)$.

Approach 3: Combinatorial optimization. We sample $x \sim p_\theta(\cdot)$ by sequentially generating w_i by solving an optimization problem whose objective is to maximize adherence to Φ_c plus additional terms encoding domain-specific heuristics encouraging w_i to be realistic.

5 Experiments

We evaluate our approach on music and poetry generation; see Appendix B.5 for details.

Music generation. We evaluated our approach on a music generation task. In this setting, x is a song, and w are measures of music. We consider 20 relations including equality, same rhythm, same pitch progression, etc.; a full list is given in Appendix B.2. We used songs from the Essen folk song corpus (Schaffrath, 1995), using 2000 for training and 500 for testing. For this dataset, we used each of the three approaches A1, A2, and A3 described in Section 4 to sample $x \sim p_\theta(\cdot \mid c)$. For A1, we use a pretrained transformer called MusicAutoBot (Shaw, 2020). For A2, we require a generative model that constructs vector embeddings of measures; we use the version of Magenta's MusicVAE which embeds single measures (Roberts et al., 2018). We finetune all models on our training examples. We use MusicAutoBot and an LSTM model with attention (AttentionRNN) (Waite, 2016) as baselines; we did not include MusicVAE since AttentionRNN strictly outperforms it.

We compare performance in terms of both high-level and low-level structure. For high-level structure, given a generated (or held-out) example x, we use our program synthesis algorithm to synthesize its relational structure $c_x = \mathcal{A}(x)$. Then, given a collection $C_{\text{gen}} =$

Model	Low-Level			High-Level	
	MusicAutoBot	MusicVAE	GraphVAE	RF Disc.	GCN Disc.
Ours (A1)	8.00	170.25	1051	0.89	0.54
Ours (A2)	8.11	**165.88**	**1046**	**0.79**	**0.63**
Ours (A3)	8.14	170.13	1049	0.91	0.43
MusicAutoBot	7.98	170.30	1076	0.95	0.51
AttentionRNN	**8.17**	170.21	1067	0.88	0.47
Human	8.15	*165.65*	*1039*	*0.51*	*0.69*

Table 1: Results for the music domain. To evaluate low-level structure, we use negative log likelihood according to MusicAutoBot, MagentaVAE, and GraphVAE. For high-level structure, we use accuracy of the random forest ("RF Disc.") and GCN ("GCN Disc."). The best (non-human) score in each column is bolded; the human score is italicized if best.

$\{c_x \mid x \in X_{\text{gen}}\}$ of synthesized structure for generated examples, along with a collection $C_{\text{human}} = \{c_x \mid x \in X_{\text{human}}\}$ of synthesized structure for the held-out human examples, we train a graph convolutional neural network (GCN) to try and discriminate C_{gen} from C_{human}. In addition, we also train a random forest (RF) over handcrafted features (described in B.4) to try and discriminate them. For low-level structure, we use the likelihood/score according to MusicAutoBot, MusicVAE, and our own GraphVAE (described in Section A.2).

We show results in Table 1. According to almost all metrics, our algorithm (using A2 for sampling $p_\theta(x \mid c)$) outperforms all others, and achieves performance very similar to human. The one exception is our measure of low-level structure according to MusicAutoBot, which rates AttentionRNN as performing slightly better than ours. Most impressively, we outperformed MusicAutoBot according to its measure of performance.

Poetry generation. We use Project Gutenberg's poetry collection (Parrish, 2018), filtered to focus on examples that contain rhymes and meter. We use 2700 10-line poems for training and 300 for testing. In this case, we were unable to apply A3 due to the large size of the vocabulary, making constrained optimization infeasible. We were also unable to apply A2 since state-of-the-art generative models such as BERT and GPT2 were unable to capture rhyming and meter, since they operate at the word level where this information is unavailable. In A1, rather than sample words going forward, we sample them backwards, making it easier to sample lines that satisfy rhyming constraints; see Appendix A. Thus, we use BERT to sample (Devlin et al., 2018), since it supports bidirectional sampling.

We compare to BERT and GPT2 (Radford et al., 2019), both finetuned on our dataset. We also consider a variant GPT2-Opt of GPT2 where we use beam search to choose line breaks in a way that maximizes occurrences of rhyme and meter. We also tried a variant of GPT2 that used constrained sampling to try and find poems that fit a given rhyme and meter scheme, but the search space was too large and it was unable to generate a single poem even after several hours. Finally, in addition to using BERT as a sequential generator, we considered an ablation where we perform constrained sampling, but with a uniformly random Φ_c rather than sampling it from a learned distribution.

As before, we compare both high-level structure and low-level structure. For high-level structure, we again use a GCN discriminator. For low-level structure, we use the negative log likelihoods according to each BERT and GPT2.

We show results in Table 2. Our approach significantly outperforms all baselines in terms of high-level structure. Furthermore, our approach outperforms BERT as a language model in terms of its own score. GPT2 and GPT2-Opt outperform our technique, most likely because they are better at natural language generation than BERT. If we could instead build on GPT2 (i.e., perform backwards sampling with GPT2), then our approach would likely achieve better performance; we leave this direction to future work. Furthermore, our approach performed comparably to GPT2-Opt in terms of BERT score, while significantly outperforming it in terms of structure. Finally, we noticed that one way the baselines (except for the ablation study) tended to perform well was by being very repetitive. Thus, we additionally measured the information entropy of the different models. As can be seen,

Model	Low-Level		High-Level	Diversity
	BERT	GPT2	GCN Disc.	Entropy
Ours (A1)	3.66	6.51	**0.69**	4.64
Ours (Ablation)	3.72	6.60	0.59	**4.67**
GPT2	**3.30**	**3.13**	0.47	4.32
GPT2-Opt	3.63	3.87	0.56	4.35
BERT	3.90	5.76	0.50	2.59
Human	4.02	4.98	*0.70*	*4.86*

Table 2: Results for the poetry domain. To evaluate low-level structure, we use the negative log likelihood of BERT and GPT2. For high-level structure, we use accuracy of the GCN ("GCN Disc."). We also show the information entropy. The highest (non-human) score in each column is bolded; the human score is italicized if it is the best.

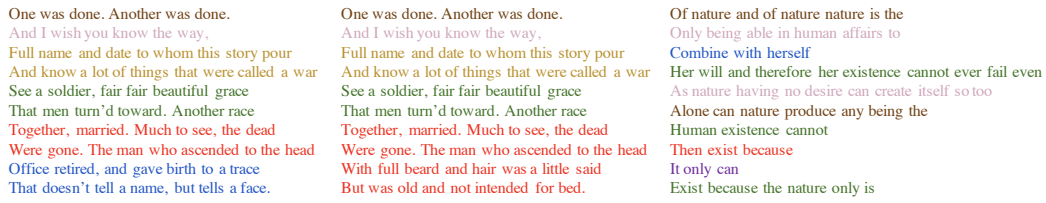

Figure 2: Left: Poetry generated using relational constraints $c \sim p_\phi(\cdot)$. Middle: user modified variant of c where the last two lines share a prototype with the two lines before them. Right: A poem generated by GPT2 optimized to maximize rhyme and meter. The colors indicate the relations synthesized by our algorithm *after* the examples were generated.

our approach (both with learned and random Φ_c) is by far closer to human in terms of human entropy, supporting our hypothesis that the baselines were overly repetitive.

In Figure 5, we show an example poem generated using our approach (left) along with one generated using GPT2-Opt (right). As can be seen, the GPT2-Opt poem does not capture structure in the same way human poems do—e.g., adjacent lines are unrelated, lines have very unequal length, and the only rhymes are the word "the" in the brown lines and the words "to" and "too" in the green lines. There is even less structure in poems generated using vanilla GPT2. Thus, GPT2 is completely unable to capture high-level structure present in the real poetry provided as training data. In contrast, our poem captures structure very similar to the human poem shown in Figure 1, such as rhyming adjacent lines.

User modifications. A key benefit of our approach is that the user can modify the relational constraints c (or construct their own from scratch) for use in the second step $p_\theta(x \mid c)$, giving the user a way to guide the generative process. Figure 5 demonstrates this process. We manually edited the part of the program corresponding to the last two lines so that they shared a prototype with the previous two lines. The example generated using the unmodified (sampled) constraints is shown on the left, and the example generated using the modified constraints is shown in the middle. The last four lines are all red; thus, the second step model successfully generated a poem that satisfies the constraints we added.

6 Conclusion

We have presented a novel approach for representing and synthesizing relational constraints on sequence data, and for generating examples whose relational structure resembles that of the training data. Our experiments demonstrate that we outperform existing approaches in terms of achieving human-like structure, while performing comparably or better on widely-used metrics which do not explicitly account for structure. Equally importantly, our approach gives the user a way to guide the generative process by modifying the relational constraints. Directions for future work include automatically discovering relational primitives, and improving the ability to sample from generative models subject to constraints.

References

Osbert Bastani, Yewen Pu, and Armando Solar-Lezama. Verifiable reinforcement learning via policy extraction. In *Advances in neural information processing systems*, pp. 2494–2504, 2018.

Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc V. Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *CoRR*, abs/1901.02860, 2019. URL http://arxiv.org/abs/1901.02860.

Cristina David and Daniel Kroening. Program synthesis: challenges and opportunities. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 375(2104):20150403, 2017. doi: 10.1098/rsta.2015.0403.

Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. Berlin, Heidelberg, 2008. Springer-Verlag. ISBN 3540787992.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805, 2018. URL http://arxiv.org/abs/1810.04805.

Kevin Ellis, Armando Solar-Lezama, and Josh Tenenbaum. Unsupervised learning by program synthesis. In *Advances in neural information processing systems*, pp. 973–981, 2015.

Kevin Ellis, Daniel Ritchie, Armando Solar-Lezama, and Joshua B. Tenenbaum. Learning to infer graphics programs from hand-drawn images. *CoRR*, abs/1707.09627, 2017. URL http://arxiv.org/abs/1707.09627.

Kevin Ellis, Catherine Wong, Maxwell I. Nye, Mathias Sablé-Meyer, Luc Cary, Lucas Morales, Luke B. Hewitt, Armando Solar-Lezama, and Joshua B. Tenenbaum. Dreamcoder: Growing generalizable, interpretable knowledge with wake-sleep bayesian program learning. *CoRR*, abs/2006.08381, 2020. URL https://arxiv.org/abs/2006.08381.

Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger (eds.), *Advances in Neural Information Processing Systems 27*, pp. 2672–2680. Curran Associates, Inc., 2014. URL http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

C. Horton and L. Ritchey. *Workbook for Harmony Through Melody: The Interaction of Melody, Counterpoint, and Harmony in Western Music.* Scarecrow Press, 2000. ISBN 9781461664147. URL https://books.google.com/books?id=XK3psrVckcAC.

Cheng-Zhi Anna Huang, Ashish Vaswani, Jakob Uszkoreit, Noam Shazeer, Ian Simon, Curtis Hawthorne, Andrew Dai, Matt Hoffman, Monica Dinculescu, and Douglas Eck. Music transformer: Generating music with long-term structure. 2019. URL https://arxiv.org/abs/1809.04281.

Jeevana Priya Inala, Osbert Bastani, Zenna Tavares, and Armando Solar-Lezama. Synthesizing programmatic policies that inductively generalize. In *International Conference on Learning Representations*, 2019.

Diederik P. Kingma and Max Welling. An introduction to variational autoencoders. *CoRR*, abs/1906.02691, 2019. URL http://arxiv.org/abs/1906.02691.

Thomas N. Kipf and Max Welling. Semi-Supervised Classification with Graph Convolutional Networks. In *Proceedings of the 5th International Conference on Learning Representations*, ICLR '17, 2017. URL https://openreview.net/forum?id=SJU4ayYgl.

Jey Han Lau, Trevor Cohn, Timothy Baldwin, Julian Brooke, and Adam Hammond. Deepspeare: A joint neural model of poetic language, meter and rhyme. *CoRR*, abs/1807.03491, 2018. URL http://arxiv.org/abs/1807.03491.

Yi Liao, Yasheng Wang, Qun Liu, and Xin Jiang. Gpt-based generation for classical chinese poetry. *CoRR*, abs/1907.00151, 2019. URL http://arxiv.org/abs/1907.00151.

OpenAI. Musenet, 2019. URL https://openai.com/blog/musenet.

Allison Parrish, 2018. URL https://github.com/aparrish/gutenberg-poetry-corpus.

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

Adam Roberts, Jesse H. Engel, Colin Raffel, Curtis Hawthorne, and Douglas Eck. A hierarchical latent vector model for learning long-term structure in music. *CoRR*, abs/1803.05428, 2018. URL http://arxiv.org/abs/1803.05428.

H. Schaffrath. The essen folksong collection in the humdrum kern format. 1995.

Andrew Shaw, 2020. URL https://github.com/bearpelican/musicautobot.

Lazar Valkov, Dipak Chaudhari, Akash Srivastava, Charles Sutton, and Swarat Chaudhuri. Houdini: Lifelong learning as program synthesis. In *Advances in Neural Information Processing Systems*, pp. 8687–8698, 2018.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. *CoRR*, abs/1706.03762, 2017. URL http://arxiv.org/abs/1706.03762.

Abhinav Verma, Vijayaraghavan Murali, Rishabh Singh, Pushmeet Kohli, and Swarat Chaudhuri. Programmatically interpretable reinforcement learning. In *International Conference on Machine Learning*, pp. 5045–5054, 2018.

Elliot Waite, 2016. URL https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn.

Halley Young, Osbert Bastani, and Mayur Naik. Learning neurosymbolic generative models via program synthesis. 2019.

A Generating Examples Given Relational Constraints

A.1 Approach 1: Constrained Sampling

In the music domain, we choose the pretrained generative model $p_\theta(w)$ to be a pretrained version of MusicAutoBot. To generate x, we sequentially sample each measure w_i conditioned on all prior measures $w_1, ..., w_{i-1}$. Each measure is sampled by sequentially sampling a sequence of pitch-duration pairs until the total duration is 16 beats (i.e., the length of a measure). During sampling, we mask pitch-duration pairs that cannot satisfy Φ_c (i.e., we set their sampling probability to zero and rescale the remaining probabilities). For instance, if the "has similar interval" relation is supposed to hold between the the prototype measure and measure i, and we are sampling the second note of measure i, then we mask any pitch j in measure i such that

$$|(\text{pitch}_j - \text{pitch}_{j-1}) - (\widetilde{\text{pitch}}_j - \widetilde{\text{pitch}}_{j-1})| \geq 3,$$

where $\widetilde{\text{pitch}}_j$ is pitch j in the prototype corresponding to w_i. In other words, we eliminate pitches that would cause sampling to violate this constraint.

In the the poetry domain, we finetune a pretrained BERT model on our dataset, by taking the pretrained models weights and then training the model on our dataset with a strong gradient weight decay. BERT has the ability to complete masked words in a sentence. We leverage this ability to sample lines that rhyme and have the same meter, which is a challenging task since such lines are a tiny fraction of the search space. We describe how we simultaneously handle rhyming and equal meter; the cases where only one of these two constraints has to hold are similar. Given a prototype $\tilde{w}$, we work backwards—on each step j, we sample from BERT a word word_j that has the same number of syllables as the corresponding word $\widetilde{\text{word}}_j$ in the prototype. More precisely, we feed BERT the sequence

$$\widetilde{\text{word}}_1, ..., \widetilde{\text{word}}_{j-1}, \text{MASK}, \text{word}_{j+1}, ...$$

and ask it to fill in the masked word, setting the probability of any word with different number of syllables as $\widetilde{\text{word}}_j$ to zero. In addition, we also set the probability of any word too similar to the original word in terms of cosine similarity to zero. For the last word (which we sample first), we additionally restrict to words that rhyme with $\widetilde{\text{word}}_j$. To increase diversity, we sample the remaining words twice—(i) backwards-to-forwards from word $k-1$ to word 1, where k is the number of words, and (ii) we resample each of the $k-1$ words (i.e., except the last word) in a random order. We discard any lines which after being sampled are determined to be too unlikely according to BERT.

A.2 Approach 2: Constraint-Aware Embeddings

In this approach, we start with a pretrained generative model $p_\theta(w)$ that ignores c; in particular, we assume that

$$p_\theta(w) = \int p_\theta(w \mid u) \cdot p(u) du,$$

where $p_\theta(w \mid u)$ is the decoder network of a VAE over w, and $p(u) = \mathcal{N}(0, I)$. Now, rather than sample $w_i \sim p_\theta(\cdot)$, we train another generative model

$$p_\psi(u_1, ..., u_m ... u_m \mid c) = \mathcal{N}((u_1, ..., u_m) \mid \mu_\psi(c), \Sigma_\psi(c))$$

to generate latent vectors $u_i \in \mathcal{U}$ such that $w_i \sim p_\theta(\cdot \mid u_i)$ are likely to satisfy Φ_c. More precisely, μ_ψ and Σ_ψ are the intermediate outputs of a graph convolutional network (GCN) (Kipf & Welling, 2017) that takes as input the graph c (where edge attributes between nodes encode Φ_c) and ultimately outputs a sequence $(u_1, ..., u_m)$.

Our approach can be considered to be a graph autoencoder in the sense that the objective function used during training rewards the reconstruction of the exact embeddings of the nodes and (implicitly, through the relationship consistency loss) their edge attribute. Our

graph encoder/decoder produces one latent vector per node, which are rewarded for close to i.i.d. Gaussian random variables with mean zero and variance one.

To train p_ψ, we construct a training example $(c_x, (u_1, ..., u_m))$ for each training example $x = (w_1, ..., w_m)$, where $u_1, ..., u_m$ are obtained by the encoder network $q_{\tilde{\theta}}(u \mid w)$—i.e., $u_i \sim q_{\tilde{\theta}}(\cdot \mid w_i)$. Then, we train p_ψ using the objective

$$J(\psi) = \sum_{(c,\vec{u})} D_{\mathrm{KL}}(\mathcal{N}(\mu_\psi(c), \Sigma_\psi(c)) \parallel \mathcal{N}(0, I)) + \sum_{i=1}^{m} \|u_i - \mu_\psi(c)_i\|_2^2 + J_{\mathrm{rel}}(\mu_\psi(c); c).$$

The first term enforces that the distribution of the latent vectors u should be Gaussian, and the second term enforces that each latent vector u should be close to its original value according to the VAE encoder $q_{\tilde{\theta}}(u \mid w)$. The third term is designed to enforce the satisfaction of the constraints Φ_c. In particular, we train a kind of "semantic discriminator" $p_\alpha(u, u'; r)$, that predicts whether $w \sim p_\theta(\cdot \mid u)$ and $w' \sim p_\theta(\cdot \mid u')$ satisfies relation r—i.e., $f(w, w', r) = 1$. The network p_α is trained on data generated from the given training examples x. Then, given p_α, we want $(u_1, ..., u_m) = \mu_\psi(c)$ to satisfy

$$p_\alpha(u_i, \tilde{u}, r) \approx \begin{cases} 1 & \text{if } (\tilde{w}, i, R) \in \Phi_c \wedge r \in R \\ 0 & \text{otherwise} \end{cases} \qquad \text{where} \qquad \tilde{u} \sim q_{\tilde{\theta}}(\cdot \mid \tilde{w}).$$

In other words, we want to generate an example x that satisfies the relations in c according to p_α. In particular, we use the loss

$$J_{\mathrm{rel}}(\vec{u}; c) = \sum_{i=1}^{m} \sum_{r \in \mathcal{R}} \mathrm{CE}\Big(p_\alpha(u_i, \tilde{u}, r), \mathbb{1}\big((\tilde{w}, i, R) \in \Phi_c \wedge r \in R\big)\Big) \qquad \text{where} \qquad \tilde{u} \sim q_{\tilde{\theta}}(\cdot \mid \tilde{w}),$$

and where CE denotes the cross-entropy loss. Once we have trained p_ψ, we generate sequences by sampling $\vec{u} \sim p_\psi(\cdot \mid c)$ and $w_i \sim p_\theta(\cdot \mid u_i)$, and constructing $x = (w_1, ..., w_m)$.

For the music domain, we use embeddings from a pretrained Magenta MusicVAE; unlike the MusicVAE used for evaluation, we finetuned it to decode only 1-2 measures of music from a 256-dimensional vector. Then,w e use the decoder portion of this model to convert the embeddings $u_1, ..., u_m \sim p_\psi(u_1, ..., u_m \mid c)$ sampled from the GCN-VAE p_ψ into measures. The graphs in the training set vary in size depending on the number of prototype measures.

A.3 Approach 3: Combinatorial Optimization

Given sampled program c, this approach attempts to generate values $x = (w_0, \ldots, w_m)$ such that $x \models \Phi_c$ by solving a system of constraint solving problem. However, the relational constraints $\phi \in \Phi_c$ are not always consistent with one another, so we relax the constraint $x \models \Phi_c$ as an objective—i.e.,

$$x = \arg\max_{x \in \mathcal{X}} \sum_{i=1}^{m} \sum_{r \in \mathcal{R}} \mathbb{1}(\mathcal{R}(\tilde{w}, w_i, n) \Leftrightarrow (\tilde{w}, i, n) \in \Phi_c).$$

Encoding this optimization problem as one Z3 can solve depends on the domain and relations. For this approach to work, we need to include additional, handcrafted terms in the objective that encourage the generated example x is realistic.

For the music domain, the optimization variables are the optimal sequence of pitches and their durations. The objective function is a linear combination of the degree to which x satisfies c, along with domain-specific heuristics—e.g., minimizing large jumps in pitch values (i.e., $|\text{pitch}_{i+1} - \text{pitch}_i| \geq 4$), not having any intervals of length 6 (i.e., $|\text{pitch}_{i+1} - \text{pitch}_i| = 6$) due to the unpleasant harmonic nature of that interval, and not having two consecutive jumps in pitch (i.e., $|\text{pitch}_{i+2} - \text{pitch}_{i+1}| \geq 5) \wedge |\text{pitch}_{i+1} - \text{pitch}_i| \geq 5)$. These heuristics are based on standard concepts from music theory (Horton & Ritchey, 2000).

B Evaluation Details

B.1 Experimental Setup

Evaluating low-level structure. We evaluate low-level structure by using negative log likelihood according to a deep generative model. In the music domain, we use both MusicAutoBot, a transformer based on the transformer-XL architecture (Dai et al., 2019), as well as the MusicVAE from Magenta, which is a hierarchical VAE that learns embeddings for each measure and then learns an LSTM-VAE on top of these embeddings. In the poetry domain, we use BERT and GPT2, both finetuned on our dataset and solely pretrained on a non-poetry dataset. For the evaluation metrics that explicitly captured structure, we computed the optimal program $\mathcal{A}(x)$ for every example x in the held-out validation human dataset as well as all of the generated examples.

High-level structure. We evaluate high-level structure by using our algorithm to synthesize the relational constraints in every generated example—i.e., $C_{\text{gen}} = \{\mathcal{A}(x) \mid x \in X_{\text{gen}}\}$, where X_{gen} is the set of examples generated using a model. Similarly, we can construct $C_{\text{human}} = \{\mathcal{A}(x) \mid x \in X_{\text{human}}\}$, where X_{human} is the set of human-created examples held-out from the training dataset. Then, we evaluate high-level structure by training a model to try and discriminate C_{gen} from C_{human}; if the model achieves lower performance, then the quality of high-level structure is higher. A general approach is to train a graph neural network (e.g., a graph convolutional network) to do so; this model takes as input the graph structure of relational constraints c, along with vector embeddings of the prototype subcomponents, and outputs whether $c \in C_{\text{gen}}$ or $c \in C_{\text{human}}$. We balance the data so it consists of 50% human data and 50% generated data. We report the cross-entropy (CE) loss; higher values correspond to better generative models. In the music domain, we additionally used a random forest (RF) trained on a manual featurization of c. We report the accuracy of the RF; lower values (i.e., closer to 50%) correspond to better generative models.

B.2 Musical Relations Used

The following are the relations $r \in \mathcal{R}$ used in the music domain:

1. Measures i and j have the same pitch classes.
2. Measures i and j have the same pitch class prefix.
3. Measures i and j have the same pitch class suffix.
4. Measures i and j's pitches have an edit distance of 1.
5. Measures i and j have approximately the same interval structure.
6. Measures i and j have the same interval prefix.
7. Measures i and j have the same interval suffix.
8. Measures i and j have the same note (pitch + duration) prefix.
9. Measures i and j have the same note (pitch + duration) suffix.
10. Measures i and j have the same rhythm.
11. Measures i and j's rhythm has an edit distance of ≤ 2.
12. Either measure i's onsets are a subset of measure j's onsets, or measure j's onsets are a subset of measure i's onsets.
13. Measures i and j have the same rhythmic and melodic contour.
14. Measures i and j have the same rhythmic and melodic contour prefix.
15. Measures i and j have the same rhythmic and melodic contour suffix.
16. Either the first or second half of measures i and j are identical.
17. Either both or neither of measures i and j have leaps.
18. Measures i and j fit within the same diatonic scale.
19. Either both or neither of measures i and j have syncopation.

20. Either both or neither of measures i and j have consecutive notes shorter than an eighth note.

B.3 Poetry Relations Used

The following are the relations $r \in \mathcal{R}$ used in the poetry domain:

1. Lines i and j have the same end rhyme.
2. Lines i and j have the same meter.

B.4 Random Forest Features

The following are the manually constructed features used in the random forest discriminator for the music domain:

1. Mean number of relations between prototype and sequence measures.
2. Variance of number of relations between prototype and sequence. measures
3. Variance in histogram of prototype measure mappings.
4. Longest sequence $i \dots j$ such that $w_i \dots w_j$ all have the same prototype measure.
5. Number of pairs (i, j) such that $\tilde{w}_i = \tilde{w}_j$ and $\tilde{w}_{i+1} = \tilde{w}_{j+1}$.
6. Mean distance between two measures with the same prototype.
7. Variance in distance between two measures with the same prototype.

B.5 Qualitative Resultss

Music domain. In addition to quantitative measurements, we evaluated the strengths and weaknesses of our approach using A2 (which was the best according to quantitative metrics) compared to the AttentionRNN baseline (which performed best out of the baselines). According to our observations, the strengths of A2 include clearer phrases with obvious resolutions, likely and plausibly repetitive rhythms, intervals between notes which seemed plausible but not overly repetitive, and less variance in quality. However, the results were not very rhythmically diverse, and certain idiomatic patterns of resolutions of intervals between notes and at the end of phrases were not followed. Furthermore, AttentionRNN does better in terms of creating realistic chord progressions (we did not explicitly consider chord progressions in our model; doing so is a promising direction for future work). Finally, while global structure is much better than the baselines, examples still relatively infrequently had the full four-bar repetitions characteristic of much folk music.

Examples. We show an example of generated songs using our approach with each A1, A2, and A3 in Figure 3, Figure 4, and Figure 5, respectively, and show an example generated using each of the baselines AttentionRNN and MusicAutoBot in Figure 6 and Figure 7, respectively. Qualitatively, the generated music and poetry appears plausible, exhibiting realistic high-level structure without sacrificing low-level structure. An example of generation based on user-modified relational constraints in the music domain is shown in Figure B.5.

We also give examples of poetry generated using our baselines—in particular, GPT2 finetuned and optimized for rhyme and meter in Figure 10, BERT finetuned as a language generation model in Figure 11, and our ablation (i.e., use BERT in conjunction with a uniformly randomly sampled Φ_c) in Figure 12.

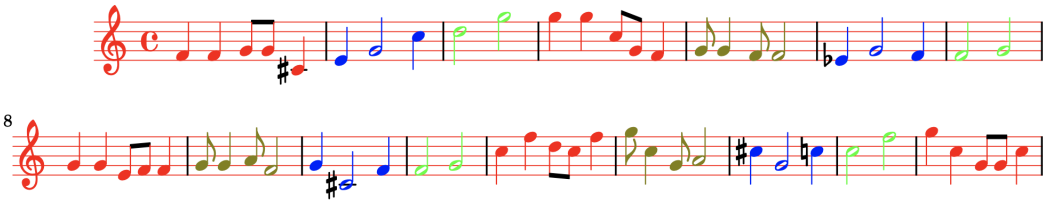

Figure 3: An example of a song generated using our approach (A1). Measures that have the same prototype are shown in the same color. Note the existence of repeating four-bar phrases, found commonly in folk songs.

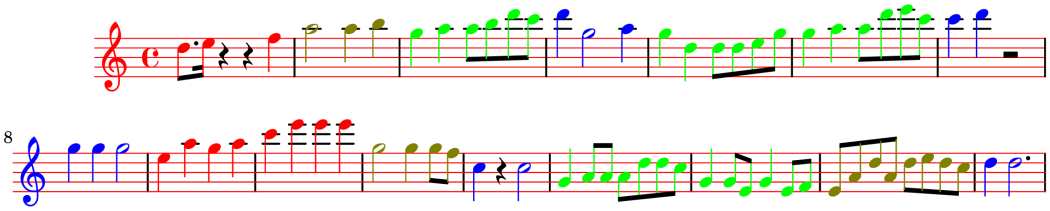

Figure 4: An example of a song generated using our approach (A2). Measures that have the same prototype are shown in the same color. Note the existence of clear phrase endings marked by long notes or rests, particularly the recurring pattern of fast notes resolving into long notes.

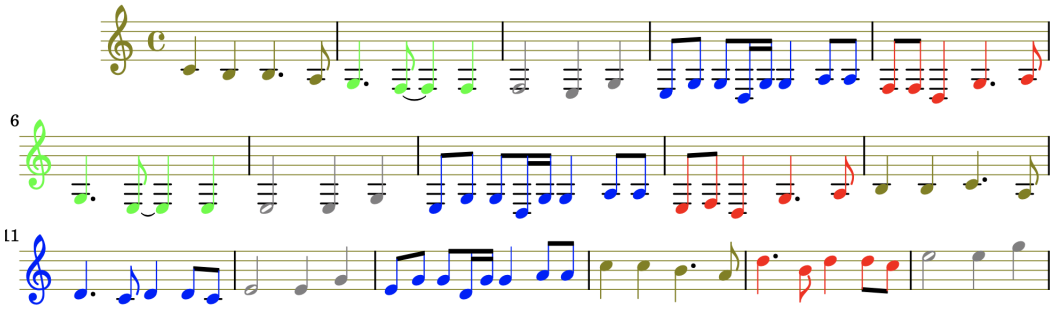

Figure 5: An example of a song generated using our approach (A3). Measures that have the same prototype are shown in the same color. The existence of two-bar and three-bar phrases is apparent, but the close note and rhythm similarities among different prototypes weaken the overall clarity of the song's melody.

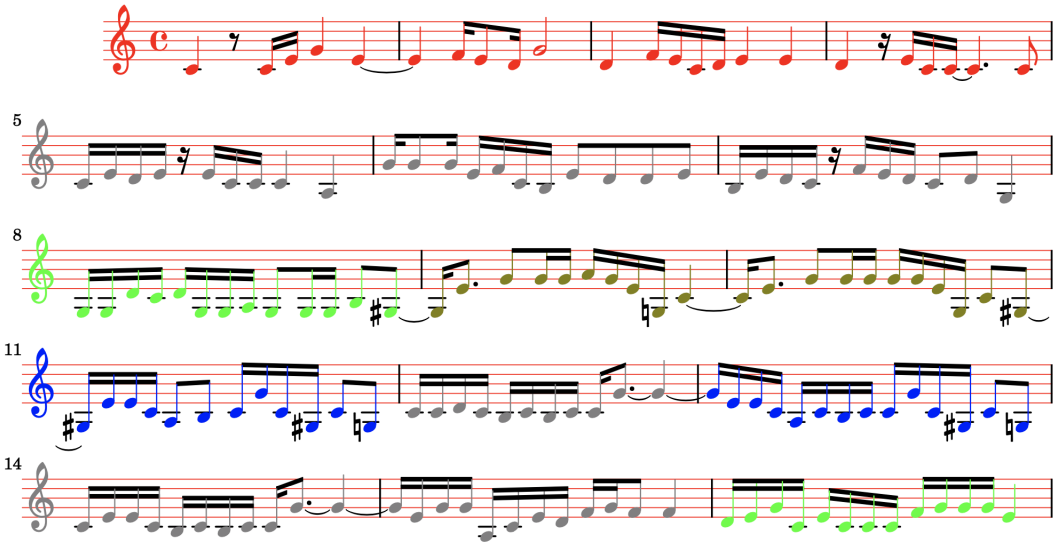

Figure 6: An example of a song generated using AttentionRNN trained on our dataset. Note the existence of erratic rhythms and unclear structure, which are common traits of custom-trained AttentionRNN models.

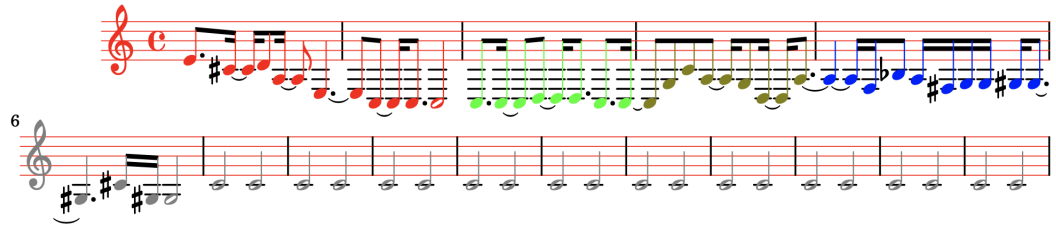

Figure 7: An example of a song generated using MusicAutoBot. Note the repetitive nature and stark contrast between the first half and second half of the song, which are common problems with transformer models.

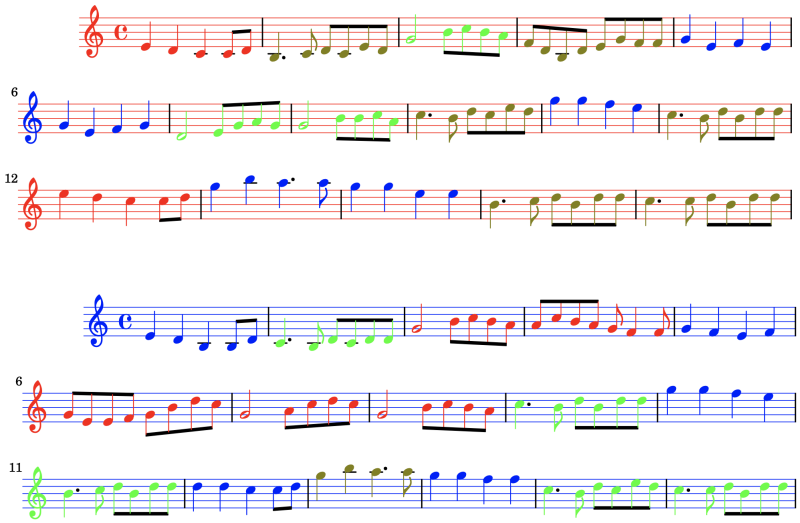

Figure 8: A song generated using our approach A3 (top), and a nearly identical song generated where part of the sampled relational constraints c were manually modified. These pieces were generated using A3, and the same reference measures $\tilde{w}$ were used, but Φ_c was slightly perturbed (the similarity relations were changed).

I know many things, and therefore I forgot,
Though I needed time to look ahead,
To understand something, time to let it fade away
As though it was yesterday as they
Were common things, free, rather—free, to go like the tide;
But another is to make no one, as it does.
Perhaps you know it. A queen, her beautiful son,
And another woman who has to go without one.
The voices like their cries of war,
They let us believe in a good restore!

Figure 9: An example of poetry generated using our approach. Lines that have the same prototype are shown in the same color.

Through the air and through the sky
And through all the world
I saw the sun the moon a star shine
In the midst of the stars
The stars were shining in my eyes my heart
Was throbbing with joy I felt
My heart was beating with love
I was
A little child in a little town
Where the little boys play

Figure 10: A poem generated using GPT2-Opt. It is more plausible than BERT in terms of of global structure, which may be due to the fact that GPT2 is a better text generation tool than BERT, but it is still somewhat repetitive and its structure is not very human-like.

all all and and
and and and
and and all
all all all
and and o
and and and
o and and
o o and o and
and of of of and o o o but and and of
and and a and and last last last of of and and

Figure 11: A poem generated using BERT. It is clearly overly repetitive and not very semantically coherent, and lacks high-level structure.

the first - independent , like for
the new songs for morning ,
a little world , they asked them for a way .
she asked them for a night ,
with two beds but sometimes lying on a light -
bed , the first for women , with another , one band ,
with one paul simon never got a play
on the subject , she ' d bought
a different dress for a different tent ,
and one dress for warning . .

Figure 12: A poem generated using our ablation. While it is much more coherent, it lacks the idiomatic rhyme and meter structure of our approach.