Abstract:

Supervised learning with next-step prediction is a common way to train a sequence prediction model; however, it suffers from known failure modes and is notoriously difficult to train models to learn certain properties, such as having a coherent global structure. Reinforcement learning can be used to impose arbitrary properties on generated data by choosing appropriate reward functions. In this paper we propose a novel approach for sequence training, where we refine a sequence predictor by optimizing for some imposed reward functions, while maintaining good predictive properties learned from data. We propose efficient ways to solve this by augmenting deep Q-learning with a cross-entropy reward and deriving novel off-policy methods for RNNs from stochastic optimal control (SOC). We explore the usefulness of our approach in the context of music generation. An LSTM is trained on a large corpus of songs to predict the next note in a musical sequence. This Note-RNN is then refined using RL, where the reward function is a combination of rewards based on rules of music theory, as well as the output of another trained Note-RNN. We show that this combination of ML and RL can not only produce more pleasing melodies, but that it can significantly reduce unwanted behaviors and failure modes of the RNN

1 Introduction

Generative modeling of music with deep neural networks is typically accomplished by training a Recurrent Neural Network (RNN) such as a Long Short-Term Memory (LSTM) network to predict the next note in a musical sequence using maximum likelihood (ML) (e.g. [6]). Similar to a Character RNN [21], these Note RNNs can be used to generate novel melodies by initializing them with a short sequence of notes, then repeatedly sampling from the output distribution generated by the model to obtain the next note. While compositions generated in this way have recently garnered attention1 , this type of model tends to suffer from common failure modes, such as excessively repeating notes, or producing sequences that lack a consistent global structure. Music compositions adhere to relatively well-defined structural rules, making music an interesting sequence generation challenge. For example, music theory tells us which note intervals sound most harmonious, which sets of notes belong to the same key, and common temporal structures for compositions, such as call and response phrases. Our research question is therefore whether these music-theory-based constraints can be learned by an RNN, while still allowing it to maintain note probabilities learned from data.

To approach this problem we propose a novel sequence learning approach in which RL is used to impose structure on an LSTM trained on data. We begin by training a deep Q-network (DQN) using a modified reward function comprising both a reward based on rules of music theory, and the probability output of a trained Note RNN. We show that this objective function can be related to stochastic optimal control (SOC) and derive two additional off-policy methods for refining the RNN, by penalizing KL-divergence from its original policy. In this framework, the model learns to adhere to a set of composition rules, while still maintaining information about the transition probabilities originally learned from the training data. We show that not only do the models successfully learn the desired behaviors, but that they can produce varied compositions which are more melodic, harmonious, and interesting than those of the Note RNN. We suggest that this method of combining ML and RL could have potential applications in a number of areas as a general way to refine existing recurrent models trained on data by imposing constraints on their behavior.

2 Background

2.1 Deep Q-Learning

In reinforcement learning (RL), an agent interacts with an environment. Given the state of the enviornment s, the agent takes an action a according to its policy π(a|s), receives a reward r(s, a), and the environment transitions to a new state, s 0 , according to its dynamics p(s 0 |s, a). The agent's goal is to maximize reward over a sequence of actions, with a discount factor of γ applied to future rewards. The optimal deterministic policy π ∗ is known to satisfy the following Bellman optimality equation,

$$Q(s_t, a_t; \pi^*) = r(s_t, a_t) + \gamma \mathbb{E}_{p(s_{t+1}|s_t,a_t)}[\max_{a_{t+1}} Q(s_{t+1}, a_{t+1}; \pi^*)] \quad (1)$$

where $Q^\pi(s_t, a_t) = \mathbb{E}_\pi[\sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'})]$ is the Q function of a policy π.

Q-learning techniques [30, 34] learn this optimal Q function by iteratively minimizing the Bellman residual. The optimal policy is given by π ∗ (a|s) = arg maxa Q(s, a). Deep Q-learning [22] uses a neural network called the deep Q-network (DQN) to approximate the Q function Q(s, a; θ). The network parameters θ are learned by applying stochastic gradient descent (SGD) updates with respect to the following loss function,

$$L(\theta) = \mathbb{E}_{\beta}[((r(s,a) + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s,a;\theta))^2] \quad (2)$$

where β is exploration policy, and θ − is the parameter of the Target Q-network [22] and held fixed during the gradient computation. The moving average of θ is used as θ − as proposed in [19]. The -greedy method is used for exploration. Additional standard techniques such as replay memory [22] and Deep Double Q-learning [33] are used to stablize and improve learning in our experiment.

2.2 Music generation with LSTM

The model is trained to predict the next note in a monophonic melody; therefore, we call it a Note RNN. Often, the Note RNN is implemented using a Long Short-Term Memory (LSTM) network. LSTMs are networks in which each recurrent cell learns to control the storage of information through the use of an input gate, output gate, and forget gate. The first two control whether information is able to flow into and out of the cell, and the latter controls whether or not the contents of the cell should be reset. Due to these properties, LSTMs are better at learning long-term dependencies in the data, and can adapt more rapidly to new data . A softmax function can be applied to the final outputs of the network, in order to obtain the probability the network places on each note. Training the LSTM can be accomplished using softmax cross-entropy loss and back propagation through time (BPTT) .

To generate melodies from this model, it is first primed with a short sequence of notes. Then, at each time step, the next note is chosen by sampling from the output distribution given by the model's softmax layer. The note that is sampled is fed back into the network as the input at the next time step. However, as previously described, the melodies generated by this model tend to wander, and lack musical structure. In the next section, we will describe how to impose rules of music theory on our model using reinforcement learning.

3 Model Design

3.1 Refining RNN with RL

Given a trained Note RNN, the goal is to teach it concepts about music theory, while still maintaining the information about typical musical compositions originally learned from data. To accomplish this task, we propose a novel sequence training method based on reinforcement learning. We use a trained Note RNN to supply the initial weights for three networks in our model: the Q-network and Target Q-network in the DQN algorithm as described in Section 2.1, and a Reward RNN. The Reward RNN is held fixed, and used to supply part of the reward value used to train the model. In order to formulate musical composition as a reinforcement learning problem, we treat placing the next note in the composition as taking an action. The state of the environment s consists of both the notes placed in the composition so far and the internal state of the LSTM cells of both the Q-network and the Reward RNN. To calculate the reward, we combine probabilities learned from the training data with knowledge of music theory. We define a set of music-theory based rules (described in Section 3.3) to impose constraints on the melody that the model is composing through a reward signal rMT (a, s). For example, if a note is in the wrong key, then the model receives a negative reward. However, it is necessary that the model still be “creative,” rather than learning a simple composition that can easily exploit these rewards. Therefore, we use the Reward RNN — or

equivalently the trained Note RNN — to compute log p(a|s), the log probability of a note a given a composition s, and incorporate this into the reward function. Figure 1 illustrates these ideas.

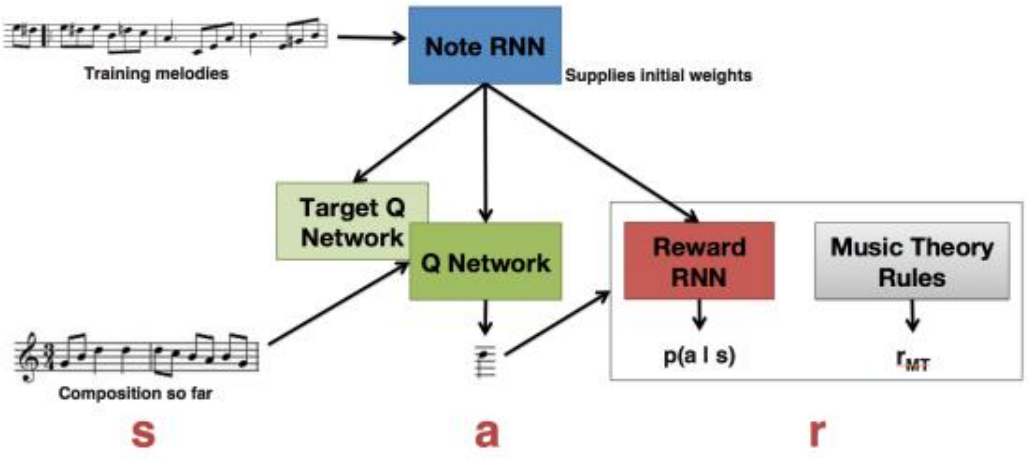


Figure 1: A Note RNN is trained on MIDI files and supplies the initial weights for the Q-network, Target-Q-network, and Reward RNN, which are used to act in the environment and compute rewards. The total reward given at time t is therefore:

$$r(s,a) = \log p(a|s) + \frac{1}{c} r_{MT}(a,s) \qquad (3)$$

where c is a constant controlling the emphasis placed on the music theory reward. Given the DQN loss function in Eq. 2 and modified reward function in Eq. 3, the new loss function and learned policy for our model are,

$$L(\theta) = \mathbb{E}_{\beta}[(\log p(a|s) + \frac{1}{c} r_{MT}(a,s) + \gamma \max_{a'} Q(s', a'; \theta^{-}) - Q(s,a;\theta))^2] \quad (4)$$

$$\pi_{\theta}(a|s) = \delta(a = \arg\max_{a} Q(s,a;\theta)) \quad (5)$$

Thus, the modified loss function forces the model to learn that the most valuable actions in terms of expected future rewards are those that conform to the music theory rules, but still have the high probability in the original data.

3.2 Relationship to Stochastic Optimal Control

The technique described in Section 3.1 has a close connection with stochastic optimal control (SOC) [17, 26, 32]. SOC defines a prior dynamics or policy, and derives a variant of the control or RL problem as performing approximate inference in a graphical model. Let τ be a trajectory of state and action sequences, p(τ) be a prior dynamics, and r(τ) be the reward of the trajectory. Then, SOC 3 introduces an additional binary variable b and defines a graphical model as p(τ, b) = p(τ)p(b|τ), where p(b = 1|τ) = e r(τ)/c and c is the temperature variable. The approximate posterior for p(τ |b = 1) using the variational free-energy method defines the following RL problem with an additional penalty based on the Kullback-Leibler (KL) divergence from the prior trajectory,

$$\log p(\tau|b=1) = \log \int p(\tau)p(b|\tau)d\tau \tag{6}$$

$$\geq \mathbb{E}_{q(\tau)}[\log p(\tau)p(b|\tau) - \log q(\tau)] \tag{7}$$

$$= \mathbb{E}_{q(\tau)}[r(\tau)/c - \mathbf{KL}[q(\tau)||p(\tau)]] = L_v(q) \tag{8}$$

where q(τ) is the variational distribution. Rewriting the variational objective Lv(q) in Eq. 6 in terms of policy πθ, we get the following RL objective with KL-regularization,

$$L_v(\theta) = \mathbb{E}_\pi[\sum_t r(s_t, a_t)/c - KL[\pi_\theta(\cdot|s_t)||p(\cdot|s_t)]] \tag{9}$$

In contrast, the objective in Section 3.1 is,

$$L_v(\theta) = \mathbb{E}_\pi[\sum_t r(s_t, a_t)/c + \log p(a_t|s_t)] \tag{10}$$

The difference is that Eq. 9 includes an entropy regularizer, and thus its optimal policy is no longer deterministic and a different off-policy method from Q-learning is required. Ψ-learning [24] and Glearning [7]2 are two off-policy methods to solve the KL-regularized RL problem, where additional Ψ and G functions are defined and learned instead of Q. We implement both of these algorithms as well, treating the prior policy as the conditional distribution p(a|s) defined by the trained Note RNN. To the best of our knowledge, this is the first application of KL-regularized off-policy methods with deep neural networks on sequence modeling tasks. The two methods are given below respectively,

$$L(\theta) = \mathbb{E}_{\beta}[(\log p(a|s) + \frac{1}{c} r_{MT}(s,a) + \gamma \log \sum_{a'} e^{\Psi(s',a';\theta^{-})} - \Psi(s,a;\theta))^2] \tag{11}$$

$$\pi_{\theta}(a|s) \propto e^{\Psi(s,a;\theta)} \tag{12}$$

$$L(\theta) = \mathbb{E}_{\beta}[(\frac{1}{c} r_{MT}(s,a) + \gamma \log \sum_{a'} e^{\log p(a'|s') + G(s',a';\theta^{-})} - G(s,a;\theta))^2] \tag{13}$$

$$\pi_{\theta}(a|s) \propto p(a|s) e^{G(s,a;\theta)} \tag{14}$$

The main difference between the two methods is the definition of the action-value functions Ψ and G. In fact G-learning can be directly derived from Ψ-learning by reparametrizing $\Psi(s, a) = \log p(a|s) + G(s, a)$. The G-function does not give the policy directly but instead needs to be dynamically mixed with the prior policy probabilities. While this computation is straight-forward for discrete action domains as here, extensions to continuous action domains require additional considerations such as normalizability of advantage function parametrizations [13]. The SOC-based derivation also has another benefit in that the stochastic policies can be directly used as an exploration strategy, instead of heuristics such as -greedy or additive noises [19, 22]. The derivations for both methods are included in the appendix for completeness

3.3 Music-theory based reward

Music structure and representation are investigated in fields such as music theory, music psychology, musicology, linguistics and machine learning. Our goal here is to investigate how theory-based constraints can be used to improve the performance of deep music generation. To achieve this, we selected several composition principles from a commonly-used music composition primer [8]. Following the principles set out on page 42 of Gauldin's book [8], we define the reward function rMT (a, s) to encourages compositions to have the following characteristics. All notes should belong to the same key, and the composition should begin and end with the tonic note of the key; e.g. if the key is C-major, this note would be middle C, or 14 in our encoding. This note should occur in the first beat and last 4 beats of the composition. Unless a rest is introduced or a note is held, a single tone should not be repeated more than four3 times in a row. To encourage variety, we penalize the model if the composition is highly correlated with itself at a lag of 1, 2, or 3 beats. The penalty is applied when the auto-correlation coefficient is greater than .15. The composition should avoid awkward intervals like augmented 7ths, or large jumps of more than an octave. Gauldin also indicates good compositions should move by a mixture of small steps and larger harmonic intervals, with emphasis on the former; the reward values for intervals reflect these

requirements. When the composition moves with a large interval (a 5th or more) in one direction, it should eventually be resolved by a leap back or gradual movement in the opposite direction. Leaping twice in the same direction is negatively rewarded. The highest note of the composition should be unique, as should the lowest note. Finally, the model is rewarded for playing motifs, which are defined as a succession of notes representing a short musical “idea”; in our implementation, a bar of music with three or more unique notes. Since repetition has been shown to be key to emotional engagement with music [20], we also sought to train the model to repeat the same motif within a composition. We do not claim these characteristics are exhaustive, strictly necessary for good composition or even particularly interesting. They simply serve the purpose of guiding our model towards traditional composition structure. It is therefore crucial to apply our framework to retain the knowledge learned from real songs in the training data.

4 Related Work

Generative modeling of music with recurrent neural networks has been explored in a variety of contexts, including generating Celtic folk music [29], or performing Blues improvisation [6]. Other approaches have examined using a Dynamical Bayesian Network for transcription from raw audio [3], or RNNs with richer expressivity or latent-variables for notes or raw audio synthesis [2, 4, 14]. Recently, impressive performance in generating music from raw audio has been attained with convolutional neural networks with receptive fields at various time scales [5]. Although the application of RL to RNNs is a relatively new area, recent work has attempted to combine the two approaches. MIXER (Mixed Incremental Cross-Entropy Reinforce) [25] uses BLEU score as a reward signal to gradually introduce a RL loss to a text translation model. After initially training the model using cross-entropy, the training process is repeated using cross-entropy loss for the $T - \Delta$ tokens in a sequence (where T is the length of the sequence), and using RL for the remainder of the sequence. Another approach [1] applies an actor-critic method and uses BLEU score directly to train a critic network to output the value of each word, where the actor is again initialized with the policy of an RNN trained with next-step prediction. Reward-augmented maximum likelihood [23] augments the standard ML with a sequence-level reward function and connects it with the above RL training methods. These approaches assume that the complete task reward specification is available. They pre-train a good policy

with supervised learning so that RL can be used to learn with the true task objective, since training with RL from scratch is difficult. Our work instead only uses rewards to correct certain properties of the generated data, while learning most information from data. This is important since in many sequence modeling applications such as music or language generation, the true reward function is not available or imperfect and ultimately the model should rely on learning from data. Our methods provide a nice framework for correcting undesirable behaviors of RNNs that can arise from limited training data or imperfect training algorithms. SeqGAN [35] applies RL to an RNN by using a discriminator network — similar to those used in Generative Adversarial Networks (GANs) [10] — to classify the realism of a complete sequence, and this classifier-based reward is used as a reward signal to the RNN. The approach is applied to a number of generation problems, including music generation. Although the model obtained improved MSE and BLEU scores on the Nottingham music dataset, it is not clear how these scores map to the subjective quality of the samples [15], and no samples are provided with the paper. In contrast, we provide both samples and quantitative results demonstrating that our approach improves the metrics defined by the reward function. Therefore, we show that our approach can be used to explicitly correct undesirable behaviors of an RNN, which could be useful in a broad range of applications. Our work also relates to stochastic optimal control (SOC), in particular the two off-policy

methods, Ψ-learning [26] and G-learning [7]. Both approaches solve a KL-regularized RL problem, in which a term is introduced to the reward objective to penalize KL divergence from some prior policy. While our methods rely on similar derivations presented in these papers, there are some key differences. First, these techniques have not been applied to DQN or RNNs, or as a way to fine-tune a pre-trained RNN with additional desired charateristics. Secondly, our methods have different motivations and forms from the original papers: original Ψ-learning [26] restricts the prior policy to be the policy at the previous iteration and solves the original RL objective with conservative, KL-regularized policy updates, similar to conservative policy gradient methods [16, 24, 27]. The original G-learning [7] penalizes divergence from a simple prior policy in order to cope with over-estimation of target Q values, and includes scheduling for the temperature parameter. Lastly, our work includes the Qlearning objective with additional cross-entropy reward as a comparable alternative, and provides for the first time comparisons among the three methods for incorporating prior knowledge in RL.

5 Experiments

To train the Note RNN, we extract monophonic melodies from a corpus of 30,000 MIDI songs. Melodies are quantized at the granularity of a sixteenth note, so each time step corresponds to one sixteenth of a bar of music. We encode a melody using two special events plus three octaves of notes. The special events are used to introduce rests and notes with longer durations, and are encoded as 0 = note off, 1 = no event. Three octaves of pitches, starting from MIDI pitch 48, are then encoded as 2 = C3, 3 = C#3, 4 = D3, ..., 37 = B5. For example, the sequence {4, 1, 0, 1} encodes an eighth note with pitch D3, followed by an eighth note rest. Because the melodies are monophonic, playing another note implicitly ends the last note that was played without requiring an explicit note off event. Thus the sequence {2, 4, 6, 7} encodes a melody of four sixteenth notes: C3, D3, E3, F3. A length-38 one-hot encoding of these values is used for both network input and network output. The architecture of the Note RNN consisted of one LSTM layer of 100 cells. The network was trained for 30,000 iterations with a batch size of 128. Optimization was performed with Adam [18], and gradients were clipped to ensure the L2 norm was less than 5. The learning rate was initially set to .5, and a momentum of 0.85 was used to exponentially decay the learning rate every 1000 steps. To regularize the network, a penalty of $\beta = 2.5 \times 10{-}5$ was applied to the L2 norm of the network weights. Finally, the losses for the first 8 notes of each sequence were not used to train the model, since it cannot

reasonably be expected to accurately predict them with no context. The trained Note RNN eventually obtained a validation accuracy of 92% and a log perplexity score of .2536. The learned weights of the Note RNN were used to initialize the three sub-networks in our RL RNN model. We then trained the RL RNN model for 3,000,000 iterations, using a batch size of 32, and clipping gradients in the same way. The Adam optimizer was used and we set the reward discount factor $\gamma = .5$. The Target-Q-network's weights $\theta -$ were gradually updated to be similar to those of the Q-network (θ) according to the formula $(1 - \eta)\theta - + \eta\theta$, where $\eta = .01$ is the Target-Qnetwork update rate. The weight placed on the music-theory rewards c was set to 0.5. Exploration was accomplished -greedily; we initial set = 1.0, and linearly annealed it to = .01 over the first 1,500,000 steps. We used -greedy for all methods for fair comparison in this paper, but prior work [28] shows that on-policy, Boltzmann exploration could be a better alternative for Ψ and G. We compare three methods for implementing fine tuning with RL: Q-learning; Ψ-learning and Glearning, where the policy defined by the trained Note RNN is used as the cross entropy reward in Q-learning and the prior policy in G- and Ψ-learning. These approaches are compared to both the original performance of the Note RNN, and a model trained using only RL and no prior policy. Model evaluation is performed every 100,000 training epochs, by generating 100 compositions and assessing the average rMT and log $p(a|s)$.

6 Results

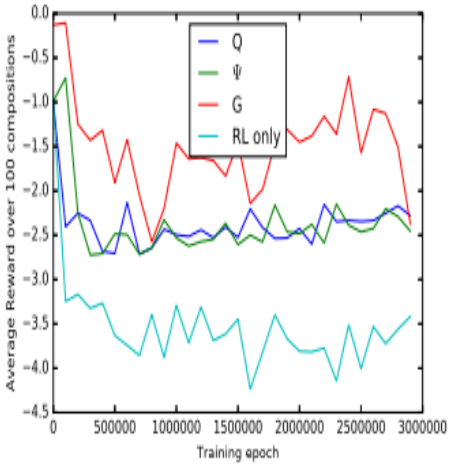
Objective assessment of generative models is difficult; often, using a metric such as MSE or likelihood is an inappropriate measure of the perceptual quality of the samples [31] [15]. In this case, we can provide quantitative results in the form of performance on the music theory rules to which we trained the model to adhere; for example, the fraction of notes played by the model which belonged to the correct key, or the fraction of melodic leaps that were resolved. Therefore, we randomly generate 100,000 compositions from each model, and compute statistics about how well these compositions adhered to the music theory rules (shown in Table 1).

Metric	**Note RNN**	**Q**	**Ψ**	**G**
Notes not in key	0.09%	1.00%	0.60%	28.7%
Mean autocorrelation - lag 1	-.16	**-.11**	**-.10**	.55
Mean autocorrelation - lag 2	.14	**.03**	**-.01**	.31
Mean autocorrelation - lag 3	-.13	**.03**	**.01**	17
Notes excessively repeated	63.3%	**0.0%**	**0.02%**	**0.03%**
Compositions starting with tonic	0.86%	**28.8%**	**28.7%**	0.0%
Leaps resolved	77.2%	**91.1%**	**90.0%**	52.2%
Compositions with unique max note	64.7%	56.4%	59.4%	37.1%
Compositions with unique min note	49.4%	51.9%	**58.3%**	**56.5%**
Notes in motif	5.85%	**75.7%**	**73.8%**	**69.3%**
Notes in repeated motif	0.007%	**0.11%**	**0.09%**	0.01%

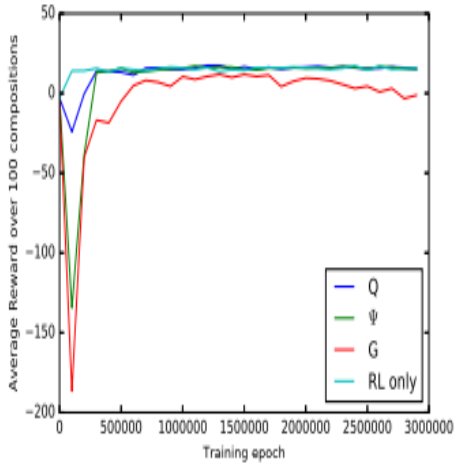
Table 1: Statistics of music theory rule adherence based on 100,000 randomly initialized compositions generated by each model. The top half of the table contains metrics that should decrease, while the bottom half contains metrics that should increase. Bolded entries represent significant improvements over the Note RNN baseline. The results above demonstrate that the application of RL is able to correct almost all of the targeted “bad behaviors” and failure modes of the Note RNN. For example, the original LSTM model was extremely prone to repeating the same note; after applying RL, we see that the number of notes belonging to some excessively repeated segment has dropped from 63% to nearly 0% in all of the RL models. While the metrics for the G model did not improve as consistently, the Q and Ψ models successfully learned to play in key, resolve melodic leaps, and play motifs. The number of compositions that start with the tonic note has also increased, composition auto-correlation has decreased, and repeated motifs have increased slightly. The degree of improvement on these metrics is related to the magnitude of the reward given for the behavior. For example, a strong penalty of -100 was applied each time a note was excessively repeated, while a reward of only 3 was applied at the end of a composition for unique extrema notes (which most likely explains the lack of improvement on this metric). The reward values could be adjusted to improve the metrics further, however we found that these values produced the most pleasant compositions. While the metrics indicate that the targeted behaviors of the

RNN have improved, it is not clear whether the models have retained information about the training data. Figure 2a plots the average log p(a|s) as output by the Reward RNN for compositions generated by the models every 100,000 training epochs; Figure 2b plots the average rMT . Included in the plots is an RL only model trained using only the music theory rewards, with no information about log p(a|s). Since each model is initialized with the weights of the trained Note RNN, we see that as the models quickly learn to adhere to the music theory constraints, log p(a|s) falls from its initial point. For the RL only model, log p(a|s) reaches an average of -3.65, which is equivalent to an average p(a|s) of approximately 0.026. Since there are 38 actions, this represents essentially a random policy with respect to the distribution defined by the Note RNN. Figure 2a shows that each of our models (Q, Ψ, and G) attain higher log p(a|s) values than this baseline, indicating they have maintained information about the data probabilities. The G-learning implementation scores highest on this metric, at the cost of slightly lower average rMT . This compromise between data probability and adherence to music theory could explain the G model's poorer performance on the music theory metrics in Table 1. Finally, while c = 0.5 produced compositions that sounded better subjectively, we found that by increasing the c parameter it is possible to train all the models to have even higher average log p(a|s). The question remains whether the fine-tuned models actually produce more pleasing melodies. We encourage readers to judge for themselves by listening to

samples from each model: goo.gl/ XIYt9m. For those with a trained eye, Figure 3 plots compositions from each model. The compositions produced by the Note RNN are sometimes dischordant and usually dull; the model tends to place rests frequently (remember that note 0 is note off and note 1 is no event), produce melodies with little variation, and select the same notes repeatedly. In contrast, the melodies produced by the RL models are more varied and interesting. The G model tends to produce more energetic and chaotic compositions, which include sequences of repeated notes (see the repeated sequences in Figure 3d). This repetition is likely because the G policy as defined in Eq. 14 directly mixes p(a|s) with

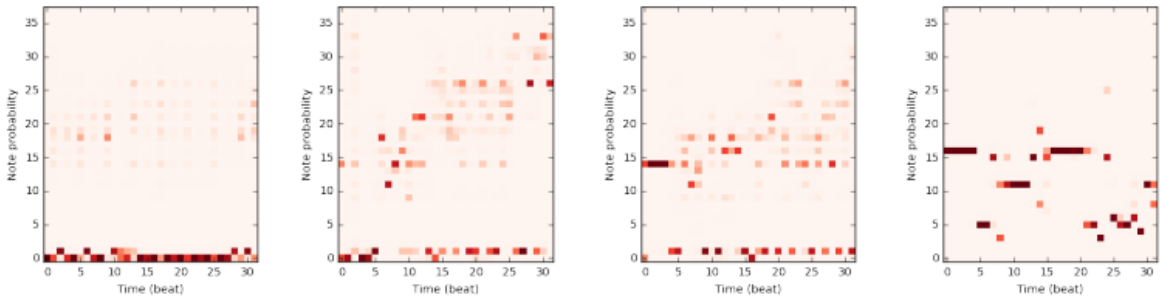


(a) *Note RNN* reward: $\log p(a|s)$



(b) Music theory reward

Figure 2: Average reward obtained by sampling 100 compositions every 100,000 training epochs. The three models are compared to a model trained using only the music theory rewards rMT . the output of the G network, and the original Note RNN strongly favours repeating notes. However, the most pleasant-sounding compositions are generated by the Q and Ψ models. These melodies stay firmly in key and frequently choose more harmonious interval steps, leading to melodic and pleasant compositions. However, it is clear they have retained information about the training data; for example, the sample q2.wav ends with a seemingly familiar riff.



(a) *Note RNN* (b) Q (c) Ψ (d) G

Figure 3: Compositions generated by each model. The probability placed on playing each note is shown on the vertical axis, with red indicating higher

7 Conclusion

We have derived a novel sequence learning framework which uses RL rewards to correct certain properties of sequences generated by an RNN, while keeping much of the information learned from supervised training on data. We proposed and evaluated three alternative techniques for achieving this, and showed promising results on music generation tasks. In addition to the ability to train models to generate pleasant-sounding melodies, we believe our approach of using RL to fine-tune RNN models could be promising for a number of applications. For example, it is well known that a common failure mode of RNNs is to repeatedly generate the same token. In text generation and automatic question answering, this can take the form of repeatedly generating the same response (e.g. “How are you?” → “How are you?” → “How are you?” ...). We have demonstrated that with our approach we can correct for this unwanted behavior, while still maintaining information that the model learned from data. Although manually writing a reward function may seem unappealing to those who believe in training models end-to-end based only on data, that approach it is limited by the quality of the data that can be collected. When the data contains hidden biases, such an approach can lead to highly undesirable consequences. In contrast, our approach allows for encoding high-level domain knowledge into the RNN, providing a general, alternative tool for training sequence models.