**Max David Gupta**
**APMA4903: Senior Seminar in Applied Mathematics**
**Machine Learning Enabled Music: LSTM Generators Tuned with Reinforcement Learning**

## Abstract

As the world undergoes a digital revolution, the musical arts are undergoing their own computer revolutions in making and sharing sound. Since the 90's, computers have become increasingly involved in all aspects of musical production. At first, it was the digitization of sound itself, transitioning from analog to digital and vinyl to MP3. Nowadays, most of the revolutions in computerized music are coming from artificial intelligence and machine learning. More recent advances in the field have come from deep learning, specifically training RNN models end-to-end on large corpuses of musical data. In this research project, we examine the foundations of RNN models and apply domain knowledge of musical theory, through reinforcement learning, to improve upon state-of-the-art models, following code from Google Magenta.

In this report, I begin with a theoretical background on the types of neural network architectures used in our study (some background in math is required). I go on to introduce the field of automatic composition for chord progressions and provide a review of previous uses of our architecture types applied historically to generate chords. We then look at the real work of the project: training a full LSTM model and tuning the output with reinforcement learning and musical theory.

## Introduction

Automated musical composition has been a computational challenge for decades spanning back to the 1940's and involving a plethora of mathematical formulae and algorithms with frequently mind-boggling complexities. Somewhat ironically, however, music is often thought of as a creative, 'intuitive' endeavor, at least it seems to be so for the world's most skilled artists, who often can't (and don't usually have to) explain the exact science of their music-making. At a high level, music production seems to become an incredibly intricate, personalized, and automatic internal creative process. In fact, most artists don't necessarily enjoy spelling out the why's and how's of their creative processes. "Works of art make rules; rules do not make works of art" claimed Claude Debussy, one of history's most talented and prolific pianists. Many musicians old and new will agree with Debussy, claiming that no set of rules, however intricate, and no algorithm, however nuanced, could ever imitate the lucid intuition of a brilliant artist. Algorithmic composition aims to do exactly the opposite: to boil creativity and musical skills down to fundamental, replicable patterns. But the intentions of algorithmic composition are in no way malicious in trying to disprove musicians, but rather to create new human-computer interfaces for artistic expression and performance: a musical man-machine synergy. Computer scientists, psychologists, and musicologists alike have put a vast amount of effort over the past few decades into this field, unraveling music cognition and the perception of pleasant-sounding music. Increased performance in automatic music generation tasks have led to advanced applications in:

- Helping novice musicians compose pleasant sounding musical pieces
- Generating accompanying music for music, games, and programs based on scene context
- Accompanying musicians during live jam sessions, enabling a dynamic musical environment for solo musicians
- Musical pedagogy and practice environments

In algorithmic music composition, arguably the simplest technique involves constructing a song through the selection of notes sequentially according to a table of transitional probabilities that specifies the mathematical chance of the next note occurring as a function of the preceding note. This is what we call a Markov Chain for music, and is the mechanism I had previously used to compose basic songs while writing my junior thesis in applied mathematics at Paris with IRCAM. Markov Chains are useful mathematical constructs, their use in music yields accuracy results that have long ago curtailed to a plateau. In the meantime, with the rise of deep learning, machine learning based algorithms and data processing techniques have since the late 90's taken over as the benchmark for musical generation tasks.

Since chord progressions are the underlying harmonic structure of most of today's music, automatic chord generation has immediate applications to all of these above areas. Real-time music improvisation systems thus need to be able to predict chords in real time along with a human musician across the timed duration of the song. Two aspects are important here: beat-to-beat accuracy (short term) and pattern to pattern accuracy (long term). The computer, like the human, needs to be able to track and play with both temporal horizons in mind.

While neural networks can already produce novel songs entirely algorithmically, the familiar chord cadence of mainstream music is usually difficult to attain. Currently available methods can predict musical chord sequence with high accuracy in single-step scenarios – that is, where the chord in a score depends only on the chord appearing just before it. However, most models fail to accurately predict chord progression based on a sequence of chords (Nika, Carsault, 2019). This lack of multi-step prediction inhibits the applicability of automatic music generation in more complex forms of musical scores. For example, when one chorus of a popular song is completely novel from the other, the listener notices and is usually thrown off. Our proposition in this project following Magenta, was that networks need to understand musical order and theory in order to generate contextually appropriate melodies.

**Neural Nets and Deep Learning: A Theoretical Background for Automatic Composition**

With the beginning of the 2000's came a widespread use of new data-driven techniques to generate musical chord progressions. With growing computational power and a larger amount of easily accessible, processable musical data came stronger "deep" learning models and higher accuracies. Machine learning performance increases proportionally with data volume and processing power and since both of the latter have exponentially increased with next-generation semiconductors chips and high-speed GPU's, it's no wonder that deep learning has come to influence the music industry too.

In our experiments, we trained three distinct types of neural networks, each with varying computational complexities: vanilla Multilayer Perceptron (MLPs) networks, and word-based Long-Short Term Memory (LSTMs) networks. In this section, we will first go through the computational theory behind each network briefly, as it relates to automatic composition, before describing the training process in our experiments.

**Previous Work in Neural Network Composition**

<u>Mozer, Neural Network Composition by Prediction</u>
In 1994, Michael C. Mozer realized that Markov chains had reached a natural plateau in performance for musical creation. Ultimately, they were temporarily restricted by the Markov assumption that the current state should derive its identity from only the previous state. Thus, Markov chains end up following a form of musical *random walk,* regulated only by a table of transition probabilities. While

this produces original and interesting examples, this linear technique of note-by-note composition lacks the architecture to scale up to entire song lengths. Mozer proposed a recurrent network architecture in CONCERT, his generative model, whose input is an entire melody instead of a single note, and whose output at each step is then a single next note in the melody. CONCERT made use of the most sophisticated RNN procedures of his time: log-likelihood objective functions, probabilistic interpretation of output values, and even 'psychologically-realistic' representations of melody, chords, and duration. The conclusions of the experiment revealed, however, that the back-propagation mechanisms in the RNN were not sufficiently powerful, *especially for contingencies that span long temporal intervals*. The network was good to learn relationships between two events separated by only a few unrelated intervening events, but as the number of intervening events grows, a point is quickly reached where the relationship cannot be learned (Mozer, 1994).

Although RNN networks outperformed Markov approaches from the 50's and 60's, they failed in all cases to find global structure (ibid). A song is, after all, much more than just a linear string of notes.

It was soon discovered that the *real* reason for Mozer's findings had to do with the *vanishing gradient problem* inherent in RNNs, a technical issue that exponentially compiles error over longer time periods. This structural deficiency made RNNs incapable of dealing with extended time series data.

Our goal then, is to find and implement a neural network architecture that is capable of creating locally *and* globally coherent musical patterns. Just three years after Mozer's discovery of RNN limitation, there emerged a solution: Long-Short Term Memory, developed by the German computer scientists Sepp Hochreiter and Jurgen Schmidhuber. LSTM introduced a revolutionary change to the vanilla RNN architecture by introducing Constant Error Carousel (CEC) units to eliminate the *vanishing gradient problem*, which was causing the global incoherence in Mozer's experiments. LSTMs feature a more powerful method of back propagation, using *forget gates* that allow errors to flow backwards through unlimited numbers of virtual layers. With this added mechanism, LSTMs can, in theory, learn tasks that require memories of events that happened thousands or even millions of time steps in the past (Goodfellow, 2017). LSTMs soon became the new benchmark for time-series analysis with machine learning, replacing vanilla RNNs almost completely, and becoming the benchmark for word-based Natural Language Processing tasks used by Apple in Auto-Type, Google in Google Translate, and Facebook in their machine translation models.

Keunwoo Choi's text-based LSTM for automatic composition

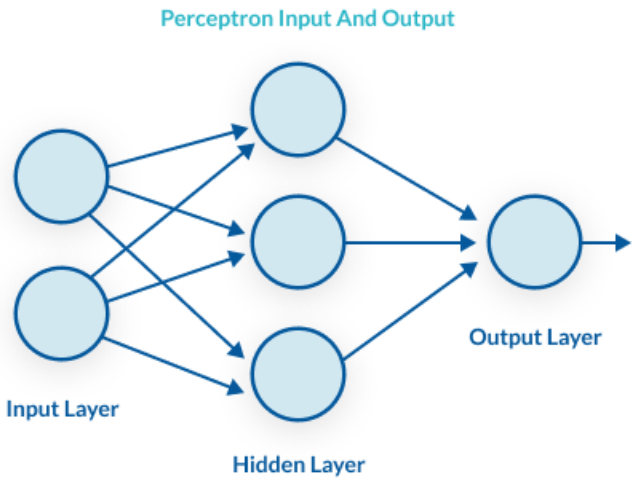
Recently, we have seen numerous studies in the automatic composition field attempt to harness the features of LSTMs for song generation. A recent success was implemented by Keunwoo Choi et al. of the Center For Digital Music, Queen Mary University of London. In this study, Choi implements character-based LSTMs and word-based LSTMs (pioneered by Andrej Karpathy) for automatic composition for the first time. Specifically, Choi trains the networks on chord progressions from the Jazz Realbook (the comprehensive compilation of lead sheets for jazz standards). In this way, Choi extracted all the chord information from the jazz dataset into text-based XLAB files and assigned an LSTM to learn musical patterns from the text itself.

**Neural Network Architectures for Chord Generation**

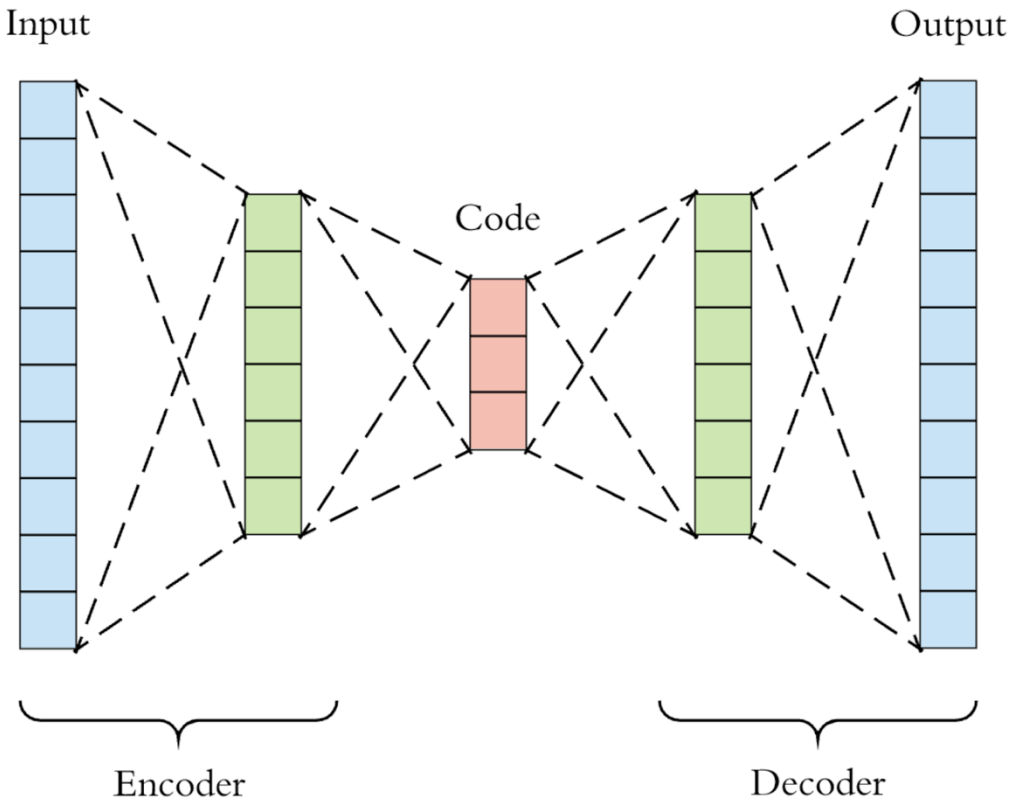
The Vanilla Multilayer Perceptron : MLP-ED

The Multilayer Perceptron is a term used to describe any type of feedforward artificial neural network (ANN) and is composed of any number of layers of *perceptrons*. The perceptron is the original

fundamental building block of the simplest task in machine learning: binary classification. The perceptron acts as a simple linear classifier, combining a set of weights from its inputs, applying a mathematical activation function, and outputting a binary prediction. The original perceptron learning algorithm created by Frank Rosenblatt on behalf of the US Navy was modeled after the neurons in the human brain, with much mathematical simplification of biological processes. A famous roadblock in the career path of the perceptron, however, was its initial inability to learn an XOR function (a logical operation that outputs true only when its inputs differ). After years of stagnation, it was revealed that *multiple layers* of these neuron-like perceptrons stacked on top of each other solved the XOR learning problem. This gave rise to the current multi-layer perceptron models and algorithms we use today, consisting of at least three feedforward layers, and utilizing the famous backpropagation technique for error optimization in training (Goodfellow, 2017).




In our application, we are passing in multiple chord elements from a previous time frame to output multiple predicted chord elements to be placed in the current time frame of a song. However, we want flexibility in the number of chords we choose to input into the model and the number we choose to predict. A glaring limitation of vanilla MLP networks is that their API is too constrained for such problems, known as *sequence to sequence* prediction problems (Goodfellow, 2017). For such problems, we want to add encoder-decoder structure to our MLP architecture to encode input chord data into one-hot vector format before the decoder translates the vector back into chord format. In our experiments, we actually work with a purely *textual* dataset (described later), so the input is a chord string, EG "G:major", which is formatted into a one-hot vector, passed through an MLP encoder-decoder architecture (with ReLU activation) as shown below:

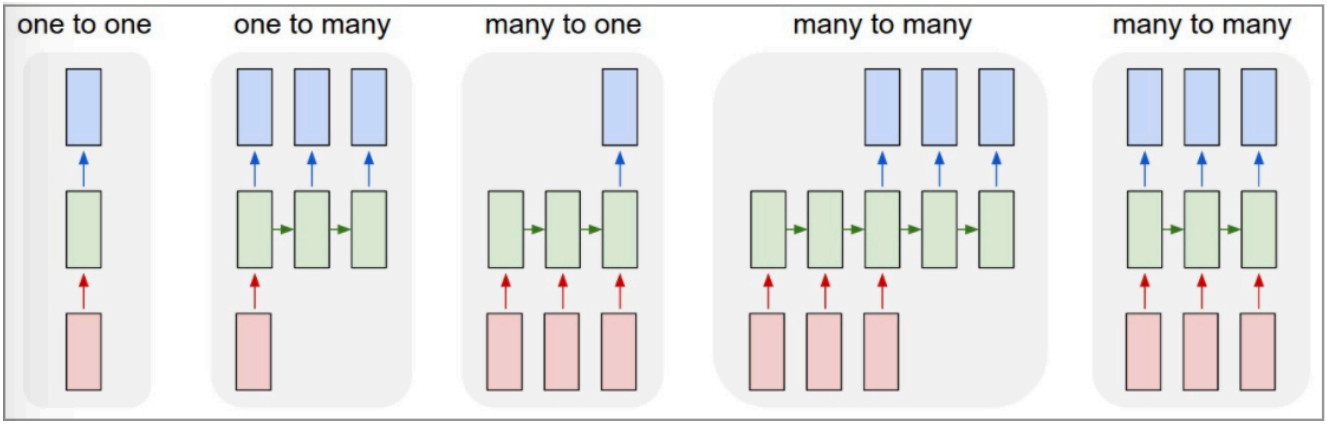



$$f(x) = x^+ = \max(0, x)$$

The architecture of the multi-layer perceptron encoder-decoder (MLP-ED) model used in this study is adopted from the MLP model proposed by Nika and Carsault in the 2019 paper, "Multi-Step Chord Sequence Prediction Based on Aggregated Multi-Scale Encoder-Decoder Networks". In this work, we use 2 encoder layers, 2 decoder layers, with a total of 500 hidden units. All encoder-decoder blocks are fully-connected layers with ReLU activation, with dropout layers (p = 0.5) between each layer and a Softmax layer at the output of the decoder block. The network is trained with ADAM at a learning rate of1e-4.

The Word-RNN Based LSTM

While adding encoder-decoder functionality to vanilla MLPs is indeed one way to allow for sequence-to-sequence prediction, there exists another class of neural networks much better naturally fitted for our task: Recurrent Neural Networks or RNNs (Karpathy, 2015). What this means is that they accept a fixed-size vector as the input (e.g. our 8 beat jazz chord progressions) and produce a fixed-size vector as the output (e.g. our computer generated 8 beat chord progressions). Not only that, but these models perform this mapping using a fixed number of computational steps (IE the number of *layers* in the network model). The core reason that recurrent neural nets are more useful is that they allow us to operate over *sequences* of vectors: sequences in the input, the output, or indeed in both. Below is a diagram illustrating these key differences between MLP's (one-to-one mapping) and RNN's (sequence mapping).




In the above diagram, each rectangle is a vector containing numericized chord information (mostly binary integers, simply 1's and 0's) and arrows represent functions (e.g. matrix multiplication or vector addition). Input vectors are red, output vectors are blue, and "*hidden" states* (intermediary states between the output and input) are green. From left to right: **(1)** Vanilla MLP, processing from fixed-size input to fixed-size output (see our MLP-ED model in the next section). **(2)** Sequence output RNN, processing a fixed-size input and outputting a many-element sequence output (we can input a single chord into our RNN model and come up with an entire predicted progression). **(3)** Sequence input RNN, processing a many-element sequence input and simplifying to a single-size output (conversely, we can input an entire chord progression and expect a single predicted chord). **(4)** Sequence input and sequence output RNN (this is the type of exercise we perform in our model: going from input chord progression to predicted next chord progression, both of varying and unlimited sizes). **(5)** Beat-aligned (synced) sequence input and output RNN (if we wish to beat-align our predicted chords, we sync the input, hidden, and output states in our RNN as shown).

The Mathematics of RNN's and LSTM's

A Vanilla RNN has a simple API, accepting an input vector x with three matrices (call them *m1*, *m2*, and *m3*), performing matrix multiplication on *m1* and the previous hidden state, on *m2* and the current input, adding these results, and then 'squashing' the sum with a *tanh* activation function to get the current hidden state:

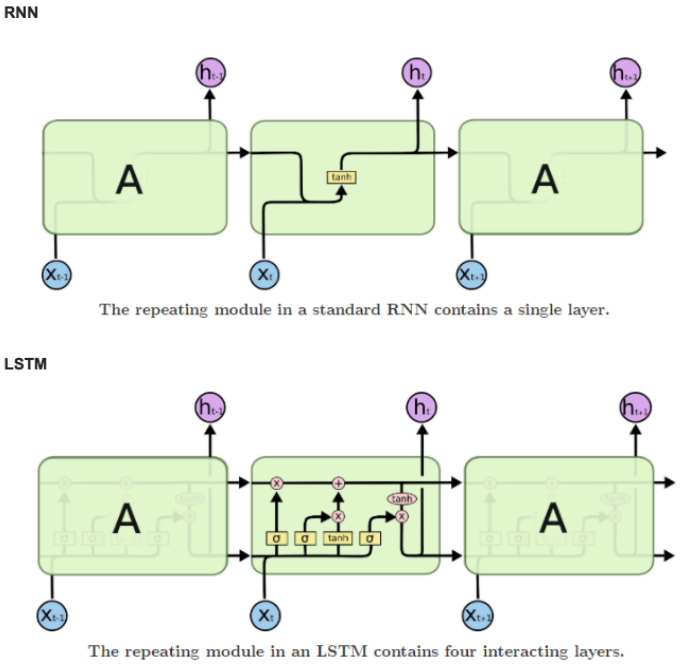
$$h_t = \tanh(W_{hh}h_{t-1} + W_{xh}x_t)$$

Then, the current hidden state is compounded by matrix multiplication with *m3*, the last input matrix, creating the output after one 'forward pass' (essentially a forward pass, also known as forward propagation is one iteration of the neural network). For best performance, as we are in the field of *deep* learning (several layers), we stack this operation on itself and compound the math above many times. In our model, we use two RNN layers, each of which has 512 hidden states (units). In other words, we have two separate RNNs: One RNN is receiving the input vectors and the second RNN is receiving the output of the first RNN as its input. Except neither of these RNNs know or care - its all just vectors coming in and going out, with loss being lessened (ideally) at each step through the gradient-optimization process known as *backpropagation*.

Now in our model, we implement a particular, more powerful type of RNN, a *Long Short-Term Memory* (LSTM) network. The reason we do *not* use the vanilla RNN described above is that these networks have

been shown previously to cause the gradient of the loss function to decay exponentially with time - a phenomenon known as the *vanishing gradient problem*. Without diving into the math, this means that vanilla RNNs cannot handle long-term temporal dependencies, like those we find in music (or any other event carried out over a longer time-series, like stock prediction, weather reports, etc.).

The LSTM solves the vanishing/exploding gradient problem by introducing new 'gates', such as input and 'forget' gates (pictured below at bottom, as compared to a vanilla RNN at top), which allow for better control over the gradient flow and enable better *preservation of long-range dependencies*. Thus, LSTMs are essentially RNN's specifically adapted for learning time-sensitive pattern information, like musical chords. In the literature, it is sometimes common to use the terms RNNs and LSTMs interchangeably, however, from here on, we refer to this specific chord prediction mechanism as the *LSTM* (Karpathy, 2015).




Applying Reinforcement Learning

In our experiments generating MIDI melody files from the LSTM model trained on 3.6GB of melody data, the results were often incoherent and wandering. Several excerpts are referenced in our seminar presentation. In fact, we learned these shortcomings were common amongst LSTM generators: excessively repeated notes, sequences lacking consistent them, and unpleasant randomness were commonly cited in Graves 2013. Following the 2017 paper by Jacques et al, we decided to experiment with implementing detailed musical domain knowledge to further train and condition the LSTM model. In our experiments, we trained a full LSTM Note Generator model (named Note RNN), implementing the learning mechanics described above, and incorporated reinforcement learning procedures to condition outputs. Following key musical theory guidelines from Robert Gauldin's *A Practical Approach to 18th Century Counterpoint* (2013), the Note RNN is fed into a Deep Q Network (Watkins, 1989) and compounded with reward rules from Gaudin's book on counterpoint. The Deep Q Network determines and refines an effective optimal policy function from the previous set of musical states to a set of next best actions. In the DQN, Q-Learning (ibid) is implemented to find the optimal

policy from the previous action at each state (in this case, each note or grouping of notes). The optimal deterministic policy π* is known to satisfy the following Bellman optimality equation:

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

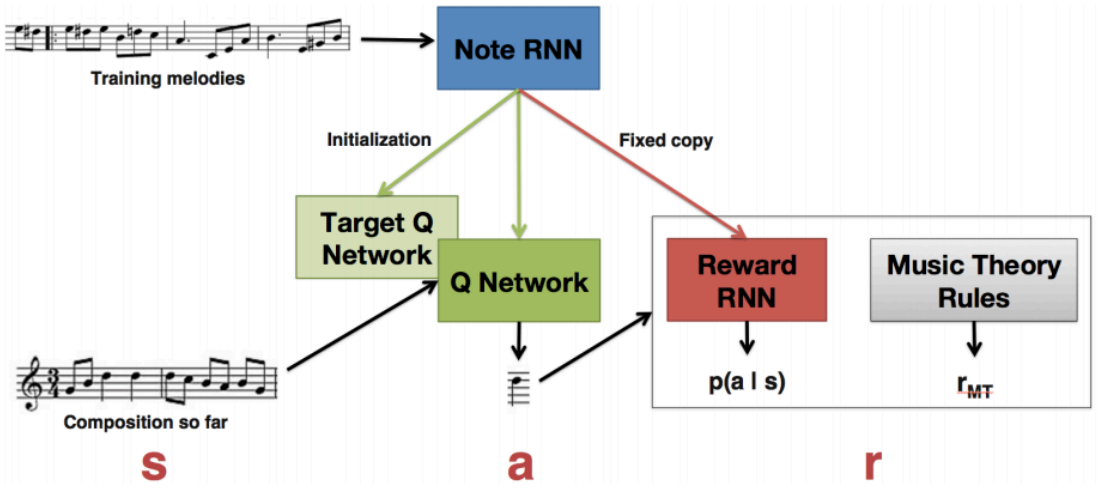
Where $Q^{\pi}(s_t, a_t) = \mathbb{E}_{\pi}[\sum_{t'=t}^{\infty} \gamma^{t'-t} r(s_{t'}, a_{t'})]$ is the Q function of a policy π. Q-learning techniques (ibid) learn the optimal Q function by iteratively minimizing the Bellman residual in a manner similar to that previously explored with RNN's, although optimizing to a different loss function with stochastic gradient descent:

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

Where Beta is the 'exploration' policy and $\theta^-$ is the set of parameters of the Target Q-network (Mnih et al 2013) we are incorporating in conjunction with the Q network.

Combining the Note RNN and Q-Networks

In order to generate musical composition as a reinforcement learning problem, we treat placing the next note in the composition as taking an action. The state of an environment *s* consists of both the notes placed in composition thus far and the internal state of the LSTM cells of both the Q-Network and the Reward RNN (Note RNN conditioned on Gauldin's Counterpoint rules in a fixed copy, as shown below). To calculate the reward, we follow Jacques et al. in combining probabilities learned from the training data with imposed constraints on the melody.



Music Theory Reward Functions

A central question of this study is whether reinforcement learning can add coherence and structure to a deterministically generated sequence by introducing domain knowledge from music theory. Jacques et al took principles straight from page 42 of Gauldin's book to define a reward function, which we encoded in our experiments. One particular example we analyzed in our seminar talk was the rewarding of compositions starting and ending with 'tonic' notes, defined as starting in the first scale degree of the diatonic scale and the tonal center. The reward function for tonic notes is called and defined in the adjoining screenshots below.

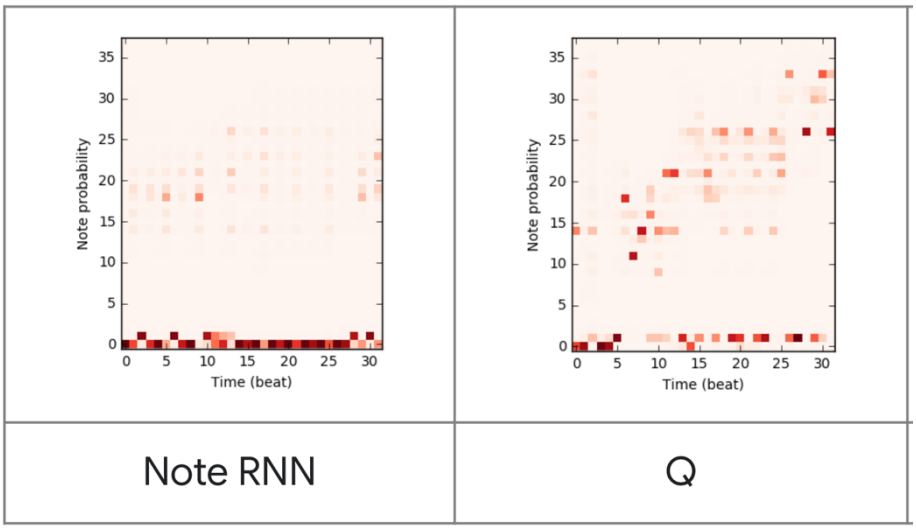
```
def reward_music_theory(self, action):
  """Computes cumulative reward for all music theory function

  Args:
    action: A one-hot encoding of the chosen action.
  Returns:
    Float reward value.
  """
  reward = self.reward_key(action)
  tf.logging.debug('Key: %s', reward)
  prev_reward = reward

  reward += self.reward_tonic(action)
  if reward != prev_reward:
    tf.logging.debug('Tonic: %s', reward)
  prev_reward = reward

  reward += self.reward_penalize_repeating(action)
  if reward != prev_reward:
    tf.logging.debug('Penalize repeating: %s', reward)
  prev_reward = reward
```

```
def reward_tonic(self, action, tonic_note=rl_tuner_ops.C_MAJOR_TONIC,
                 reward_amount=3.0):
  """Rewards for playing the tonic note at the right times.

  Rewards for playing the tonic as the first note of the first bar, and the
  first note of the final bar.

  Args:
    action: One-hot encoding of the chosen action.
    tonic_note: The tonic/1st note of the desired key.
    reward_amount: The amount the model will be awarded if it plays the
      tonic note at the right time.
  Returns:
    Float reward value.
  """
  action_note = np.argmax(action)
  first_note_of_final_bar = self.num_notes_in_melody - 4

  if self.beat == 0 or self.beat == first_note_of_final_bar:
    if action_note == tonic_note:
      return reward_amount
  elif self.beat == first_note_of_final_bar + 1:
    if action_note == NO_EVENT:
      return reward_amount
  elif self.beat > first_note_of_final_bar + 1:
    if action_note == NO_EVENT or action_note == NOTE_OFF:
      return reward_amount
  return 0.0
```

Furthermore, the generator is rewarded if the composition starts and ends in the same key, if repetition of a single tone happens no more than four times in a row, and if the model remains lowly correlate with itself at a lag of 1, 2, or 3 beats. Nuances like penalties for awkward intervals and large octave jumps are also introduced. We also looked in detail at the idea of rewarding *motifs*: recurrent 'ideas' of identifiable, successive notes, in our code defined as a bar of music with three or more unique notes. Research was done by Livingstone et al in 2012 to verify the emotional engagement effect of repetition in music.

Experiments and Results

The resulting differences after incorporating Q-Learning and musical theory rewards were quite remarkable. Seeing as we followed the exact methodologies of Jacques et al from 2017, we ended up with the same results, shown below tabularly and graphically.




Graphically, we can see the vast improvements by the Q-networks on the Note RNN's previous redundancy issues. Namely, the Q-Network introduces much more sophisticated distribution of notes across time. We demonstrated this difference audially in our seminar presentation: the Note RNN giving into repetitive drones of single G notes, and the Q-Network taking the same data and improvising a dynamic G-based riff on piano.

| Behavior | Note RNN | Q |
|---|---|---|
| Notes excessively repeated | 63.3% | **0.0%** |
| Notes not in key | 0.1% | 1.0% |
| Mean autocorrelation (lag 1,2,3) | -.16, .14, -.13 | **-.11, .03, .03** |
| Leaps resolved | 77.2% | **91.1%** |
| Compositions starting with tonic | 0.9% | **28.8%** |
| Compositions with unique max note | 64.7% | 56.4% |
| Compositions with unique min note | 49.4% | 51.9% |
| Notes in motif | 5.9% | **75.7%** |
| Notes in repeated motif | 0.007% | **0.11%** |

We see the further benefits of adding musical theory on the behavior of the reinforcement learning tuned model as well. Namely, the RL-tuned model *eliminates* excessive note repetition, as discussed, reduces autocorrelation, reduces large leaps, increases tonic starting notes, and successfully incorporates motifs and repeated motifs.

## Concluding Discussion

As we have seen in our experiments generating music with LSTMs, the differences in purely data-derived sound and domain-reinforced sound can be significant. I think reinforcement learning tuning on RNN models will have wide-reaching applications and consequence going forward. While some machine learning purists will insist on end-to-end data-based training, I think domain knowledge is always essential in training deep learning models to effectively gain insight into the complex tasks to which they are applied. Music theory is not something that jumps out easily to a machine, even when trained on tens of thousands of songs, as we saw in our experiments. Adding reward and penalty constraints from expert sources greatly improved performance, while eliminating several major drawbacks of RNN music: repetitiveness, randomness, and discord. Ultimately, in all future applications of machine-enabled music, we want to be conditioning machines to understand what makes music pleasant to human ears, and I sincerely hope work in this domain will continue to expand horizons for human-machine creative collaboration.

References:

*Carsault, Tristan, et al. "Multi-Step Chord Sequence Prediction Based On Aggregated Multi-Scale Encoder-Decoder Networks." 2019 IEEE 29th International Workshop on Machine Learning for Signal Processing (MLSP), 2019, doi:10.1109/mlsp.2019.8918813*

*Choi, Keunwoo & Fazekas, George & Sandler, Mark. (2016). Text-based LSTM networks for Automatic Music Composition*

*Eck, D., and J. Schmidhuber. "Finding Temporal Structure in Music: Blues Improvisation with LSTM Recurrent Networks." Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing, doi:10.1109/nnsp.2002.1030094*

*Giorgi, Bruno Di, et al. "A Data-Driven Model of Tonal Chord Sequence Complexity." IEEE/ACM Transactions on Audio, Speech, and Language Processing, 2017, pp. 1–1., doi:10.1109/taslp.2017.2756443.*

*Goodfellow, Ian, et al. Deep Learning. The MIT Press, 2017*

*Jacques, N. (2017, April). Tuning Recurrent Neural Networks with Reinforcement Learning. Retrieved from https://arxiv.org/pdf/1611.02796v2.pdf*

*Karpathy, Andrej. The Unreasonable Effectiveness of Recurrent Neural Networks, karpathy.github.io/2015/05/21/rnn-effectiveness/*

*Mozer, Michael C. "Neural Network Music Composition by Prediction: Exploring the Benefits of Psychoacoustic Constraints and Multi-Scale Processing." Connection Science, vol. 6, no. 2-3, 1994, pp. 247–280., doi:10.1080/09540099408915726*

*Paiement, J., et al. [PDF] A Probabilistic Model for Chord Progressions: Semantic Scholar. 1 Jan.1970,www.semanticscholar.org/paper/A-Probabilistic-Model-for-Chord-Progressions-Paiement-Eck/7262b1592fef5e4aea6700b96d32390fde727724*

*Schmidhuber, Jürgen. "Deep Learning in Neural Networks: An Overview." Neural Networks, vol. 61, 2015, pp. 85–117., doi:10.1016/j.neunet.2014.09.003*

*Vaswani, Ashish, et al. "Attention Is All You Need." ArXiv.org, 6 Dec. 2017, arxiv.org/abs/1706.03762*