Machine Learning Enabled Music: Reinforcement Learning Tuned RNN’s

Max, Sally, Tassneen

Why Music and Machine Learning?

References

The RL Tuner Model (Jacques et al. 2017)

RL Tuner Blog (Magenta, 2016)

Magenta Code (Github)

Understanding LSTM Networks (Olah, 2015)

A Practical Approach to 18th Century Counterpoint, Robert Gauldin 2013

Outline

1. Defining the Field:

A) Algorithmic Composition
B) State of the Art: deep learning models

2. An Introduction to the Melody RNN:

A) Demo
B) LSTM Walkthrough
C) Shortcomings

3. An Introduction to the RL Tuner Mechanics

a) Reinforcement Learning and Markov Decision Process
b) Optimal Policy Functions and Bellman Equation
c) Q-Learning
d) Deep Q-Learning

4. Music Theory:

a) Contextualizing the Machine Learning with domain knowledge
b) Music Theory learning constraints

5. Music Demo - Comparison

6. Futures

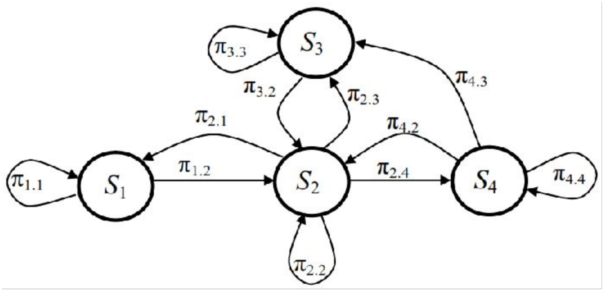
7. Questions

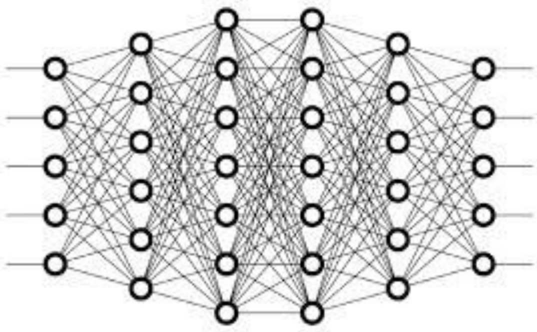
Algorithmic Composition

1950’s - 70’s: Birth of Stochastic Music with Markov Chains: Xenakis

1980’s - 90’s: Revitalization of Neural Networks: Michael Mozer’s RNN’s

2000’s - Present: Deep Learning





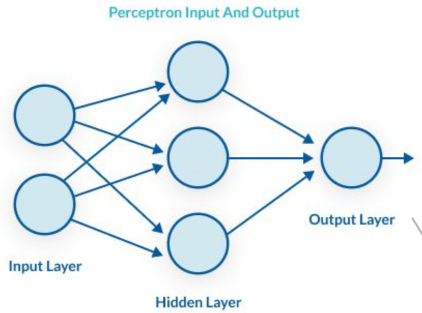
Deep Learning
State of the Field

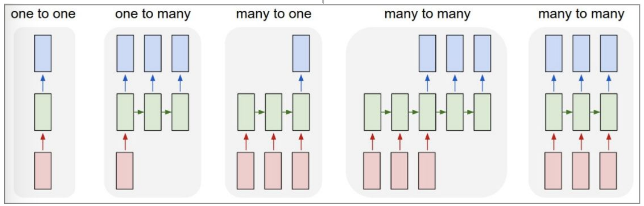
Multilayer Perceptron

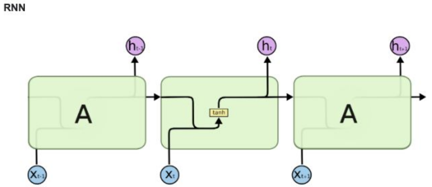
Recurrent Neural Networks (vanishing gradient)

Long Short-Term Memory

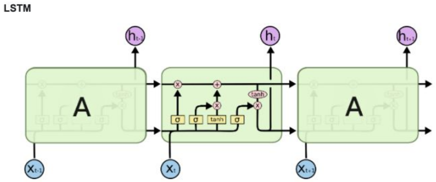
LSTM with Reinforcement Learning (to be discussed)







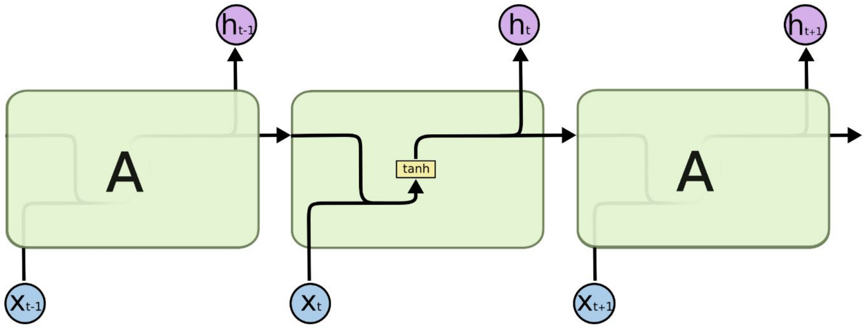
The repeating module in a standard RNN contains a single layer.



The repeating module in an LSTM contains four interacting layers.

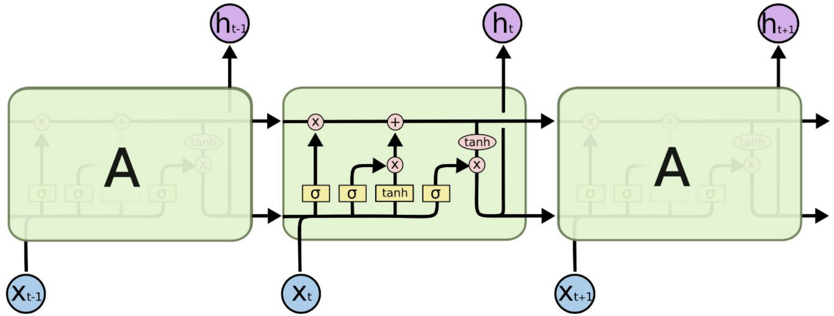
http://karpathy.github.io/2015/05/21/rnn-effectiveness/

Recurrent Neural Nets vs. LSTM's



The repeating module in a standard RNN contains a single layer.

Vanilla RNN's (1986 Rumelhart): non-effective at contextualizing prior information

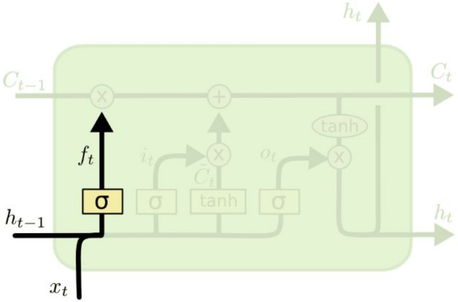


The repeating module in an LSTM contains four interacting layers.

LSTM's (1997 Hochreiter, Schmidhuber) : designed to handle long term dependencies in sequence data

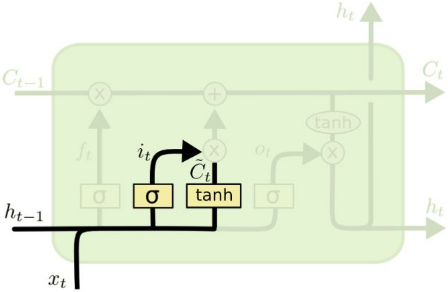
Musical Break!

Step by Step LSTM Walkthrough: Understanding Sequence Memory



$$f_t = \sigma\left(W_f \cdot [h_{t-1}, x_t] \ + \ b_f\right)$$

1) Forget Gate (Sigmoid 0/1)

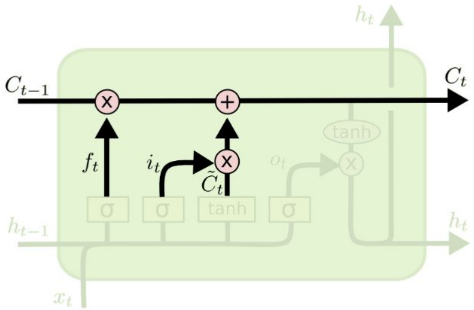


$$i_t = \sigma\left(W_i \cdot [h_{t-1}, x_t] \ + \ b_i\right)$$
$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] \ + \ b_C)$$

2) 'Remember' Gate
   a) Input Gate (Sigmoid)
   b) New Inputs (Tanh)

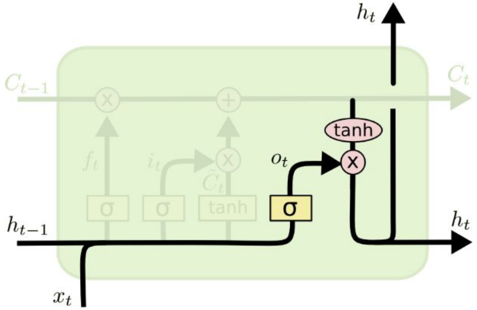
http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LSTM Continued



$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

3) Update old state by dropping forget params and adding new input params



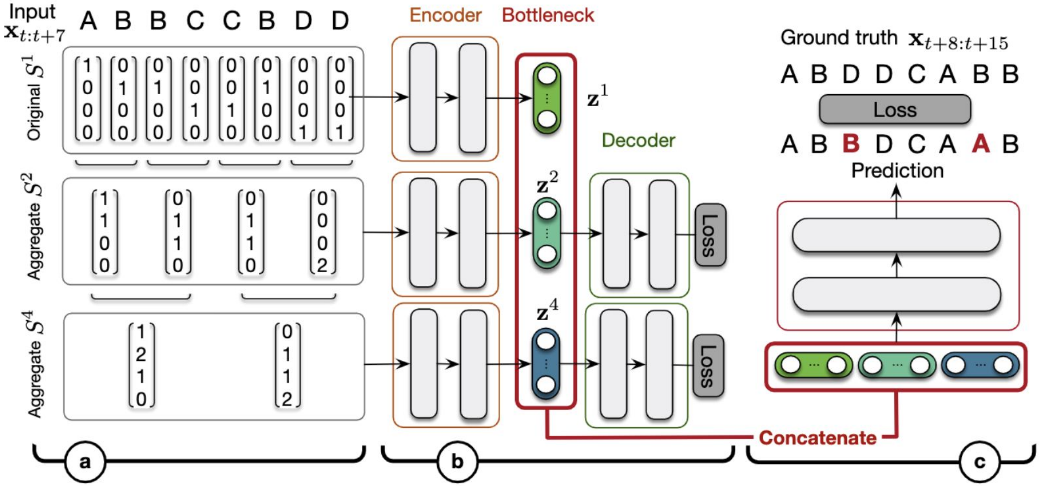
$$o_t = \sigma\left(W_o\ [h_{t-1}, x_t]\ +\ b_o\right)$$
$$h_t = o_t * \tanh\left(C_t\right)$$

4) Output:

a) Sigmoid: decide on output of cell state
b) Tanh on cell state -> [-1, 1]

http://colah.github.io/posts/2015-08-Understanding-LSTMs/

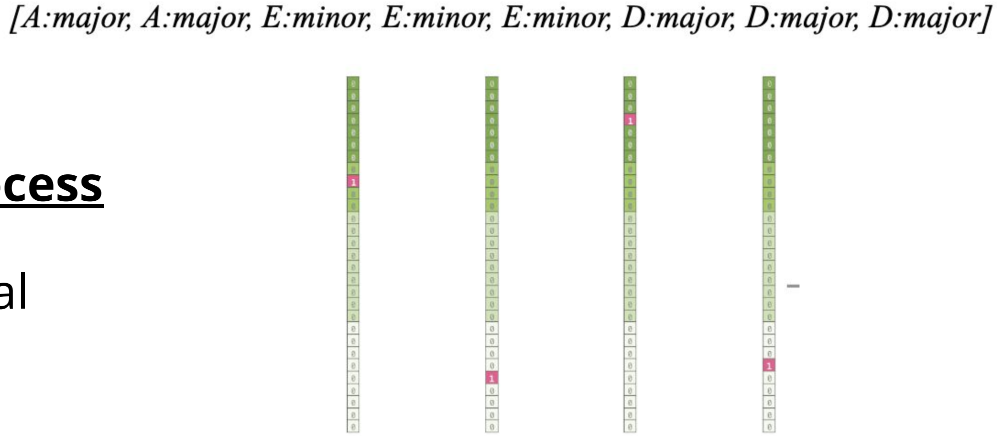
Musical RNN Architecture



A rough sketch of the Melody RNN system we are using, formed into an encoder-decoder structure for sequence aggregation

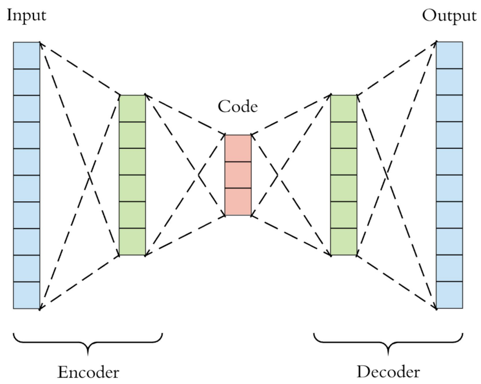
MLSP19, Nika et al. 2019

Sequence Vectors in Music



<u>Dataset Creation and Training Process</u>

MIDI file -> Notes -> *One-Hot* -> Neural Network Layers -> New Notes



https://towardsdatascience.com/applied-deep-learning-part-3-autoencoders-1c083af4d798

Code Implementation

Code by Magenta:

- RNN Implementation
- One-Hot Encoding
- MIDI Generation

Music Demo

- MIDI file output
- Monotonous
- Wandering, random qualities

RNN Melody Sample

https://magenta.tensorflow.org/2016/11/09/tuning-recurrent-networks-with-reinforcement-learning

Failures of RNN

- Multi-Step Prediction Issue
- Global Incoherence
- Wandering randomness, no ‘musical’ structure
- Random repetitions and deviations are less pleasing

How do we solve this?

Out of randomness, enforce ***good*** habits, penalize ***bad*** habits, establish baseline rules. Principles of reinforcement learning.

“We adore chaos because we love to produce order” - M. C. Escher

RL Tuner: Tuning RNN with RL

- Goal: given a Note RNN, want to teach it music theory while maintaining information learnt from data through LSTM
- Method: trains a deep Q-network (DQN) with a reward function comprising both a music-theory based reward and the probability output of a trained Note RNN

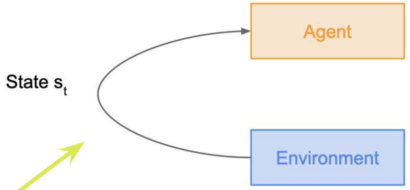
Reinforcement Learning

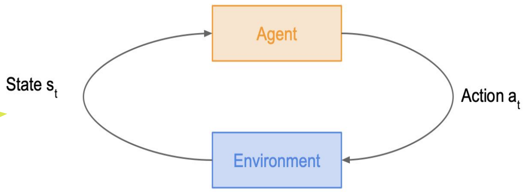
- No labelled input/output pairs needed
- Formal definition: An *agent* interacting with an *environment* that provides numeric reward signals learns to take actions in order to maximize *return* $R = \sum_{t=0}^{\infty} \gamma^t r_t$
- Objective: find an **optimal policy function** π^* mapping from the set of states to the set of actions
- Mathematical formation: **Markov Decision Process**

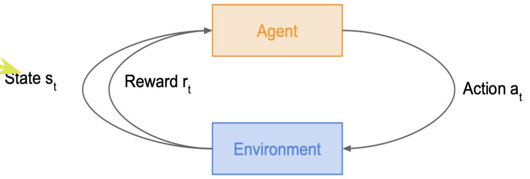
http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf Stanford CS231N, Fei-fei Li Et al, 2017

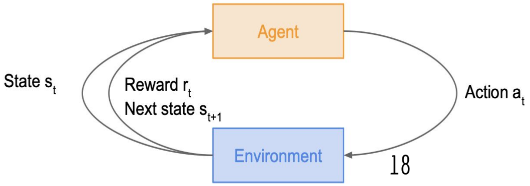
Markov Decision Process

- (S, A, P, R, γ)
- At time t=0, environment samples an initial state s_0 from $p(s_0)$
- For t=0 until $s=s_{terminal}$:
  - agent takes an action a_t from A according to π
  - environment gives back a reward $r_t \sim R(. \mid s_t, a_t)$
  - environment samples next state $s_{t+1} \sim P(. \mid s_t, a_t)$
  - agent receives r_t and s_{t+1} from environment









S: set of states
A: set of actions
P: transition probability from current state and action
R: reward probability from state s and action a
γ : discount factor for calculating return
π : policy function, probability of taking action a in state s

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf Stanford CS231N, Fei-fei Li Et al, 2017

Optimal Policy Function π*

- Return $R = \sum_{t=0}^{\infty} \gamma^t r_t$
- Policy $\pi(a, s) = \Pr(a_t = a \mid s_t = s)$
- Optimal Policy Function $\pi^* = \arg\max_{\pi} \mathbb{E}\left[\sum_{t \geq 0} \gamma^t r_t | \pi\right]$ with $s_0 \sim p(s_0), a_t \sim \pi(\cdot|s_t), s_{t+1} \sim p(\cdot|s_t, a_t)$
- How good is a state? $V^\pi(s) = E[R \mid s, \pi]$
  - Value function: expected return starting with state s and following policy π
- How good is a (state, action) pair? $Q^\pi(s, a) = \mathrm{E}[R \mid s, a, \pi]$
  - Q-value function: expected return starting with state s and taking action a and following policy π
- After finding out the optimal policy π^*, the agent chooses the action from $Q^{\pi^*}(s, \cdot)$ with the highest value at each state s

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf Stanford CS231N, Fei-fei Li Et al, 2017

Bellman Equation

- recall from last slide: optimal Q-value function

$$Q^*(s,a) = \max_{\pi} \mathbb{E}\left[\sum_{t\geq 0} \gamma^t r_t | s_0 = s, a_0 = a, \pi\right]$$

- $Q^*(s,a)$ satisfies the **Bellman Equation**:

$$Q^*(s,a) = \mathbb{E}_{s'\sim\mathcal{E}}\left[r + \gamma \max_{a'} Q^*(s',a') | s, a\right]$$

  - optimal strategy: take the action that maximizes the expected value of $r + \gamma Q^*(s',a')$
- **Q-Learning** is an algorithm that solves for the optimal policy function.

Q-Learning

- Formal definition: A model-free RL algorithm to learn Q* (quality of actions telling an agent what action to take under what circumstances)
- The iterative learning process:

$$Q_{i+1}(s,a) = \mathbb{E}[r + \gamma max_{a'} Q_i(s',a')|s,a]$$

  - initial value Q_0 is chosen arbitrarily
  - at each stage with an updated (state, action), Q(s,a) is updated
  - as i approaches infinity, this converges to Q*
- Value iteration has problems → use a function approximator to estimate the Q-function instead
- If the function approximator is a neural network => **Deep Q-Learning**

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf Stanford CS231N, Fei-fei Li Et al, 2017
Chris Watkins, 1989

Deep Q-Learning

- Deep Q-network (**DQN**) is used to approximate the Q-function

$$Q(s, a; \theta) \approx Q^*(s, a)$$

- θ learned by applying *stochastic gradient descent (SGD)* updates:
  - Loss function: $$L_t(\theta_t) = (r_t + \gamma \max_{a'} Q(s', a'; \theta_{t-1}) - Q(s, a; \theta_t))^2$$
  - first two terms: the Q-function the network is trying to learn
  - last term: actual value output by the Q-network at step t
  - θ_{t-1} is held fixed and not updated
  - i.e. the prediction error in estimating the Q function made by the Q-network

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf Stanford CS231N, Fei-fei Li Et al, 2017
https://magenta.tensorflow.org/2016/11/09/tuning-recurrent-networks-with-reinforcement-learning/ Natasha Jaques, 2016
https://arxiv.org/pdf/1312.5602v1.pdf DeepMind, Vlad Mnih Et al, 2013

DQN

- As the agent interacts with the environment, the tuples $<s_t, a_t, r_t, s_{t+1}>$ are stored in an *experience replay buffer*.
- Training the Q-network: randomly sampling batches from the experience buffer to compute the loss instead of using consecutive batches
- Why is experience buffer essential?
    - Samples are correlated; current θ determines next training samples → bad feedback loops → local minimum or diverge
- DQN could lead to overestimation sometimes → **Deep Double Q-Learning**

http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture14.pdf Stanford CS231N, Fei-fei Li Et al, 2017
https://magenta.tensorflow.org/2016/11/09/tuning-recurrent-networks-with-reinforcement-learning/ Natasha Jaques, 2016

Deep Double Q-Learning

- An additional *Target Q-network* is used to estimate expected future return, while the Q-network is used to choose the next action.

Algorithm 1 : Double Q-learning (Hasselt et al., 2015)

Initialize primary network Q_θ, target network $Q_{\theta'}$, replay buffer $\mathcal{D}$, $\tau << 1$
for each iteration **do**
  for each environment step **do**
    Observe state s_t and select $a_t \sim \pi(a_t, s_t)$
    Execute a_t and observe next state s_{t+1} and reward $r_t = R(s_t, a_t)$
    Store (s_t, a_t, r_t, s_{t+1}) in replay buffer $\mathcal{D}$
  for each update step **do**
    sample $e_t = (s_t, a_t, r_t, s_{t+1}) \sim \mathcal{D}$
    Compute target Q value:
      $Q^*(s_t, a_t) \approx r_t + \gamma\ Q_\theta(s_{t+1}, argmax_{a'} Q_{\theta'}(s_{t+1}, a'))$
    Perform gradient descent step on $(Q^*(s_t, a_t) - Q_\theta(s_t, a_t))^2$
    Update target network parameters:
      $\theta' \leftarrow \tau * \theta + (1 - \tau) * \theta'$

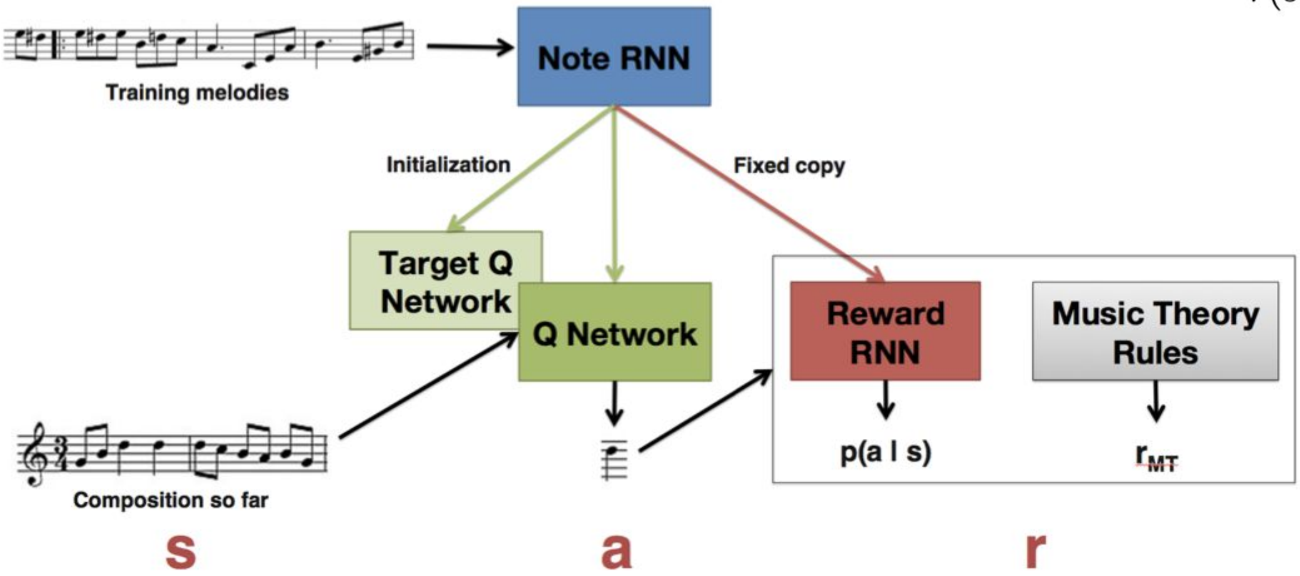
https://arxiv.org/pdf/1509.06461.pdf DeepMind, Hado van Hasselt, 2015

https://towardsdatascience.com/double-deep-q-networks-905dd8325412, Chris Yoon, 2019

RL Tuner

- Action: form next note in composition
- State: notes we had in the composition + internal states of LSTM
- Reward: probs learnt form data + knowledge of music theory (e.g. if it's a wrong key according to music theory, then reward=-1)

$$r(s,a) = \log p(a|s) + r_{MT}(a,s)/c$$



https://arxiv.org/pdf/1611.02796v2.pdf Tuning Recurrent Neural Networks with Reinforcement Learning, Natasha Jaques Et al, 2017

How does it work?

1. Train the Note RNN on monophonic melodies from a corpus of MIDI files
2. Encode the melodies:
   - each time step corresponds to 1/16 of a bar of music
   - 0=note off, 1=no event
   - three octaves of pitches, starting from MIDI pitch 48, are encoded as: C3=2, C3#=3, D3=4, …, B5=37
   - e.g. {4,1,0,1} encodes an ⅛ note with pitch D3, followed by an ⅛ note rest
3. Initialize the 3 networks in RL Tuner using the learnt weights of the RNN
4. Train the 3 networks in RL Tuner, and choose a music theory reward function or a combination of several music theory reward function

Next:

-explaining music theory rewards in details

Where is the *music*? : Music Theory Rewards

- The gist of Music Theory as a Constraint:
    - Reward good music theory behavior
    - Negatively reward bad music theory
- Will go into detail about what is good vs bad musical behavior
    - It all ties in with Music Theory

Why does Music Theory matter?

- Music theory is the backbone of all musical compositions, and if we want to automate the composition process, we need to apply those rules into our program, otherwise there would be no way to make “good-sounding,” recognizable music
- “Good” music:
  - Something people can easily recognize as music
  - Melody/ Cantus firmus
  - Defined by rules based in music theory
  - We don’t want 21st century polyphonic dissonant music (good music, but the general public might not get it)

Constraints: Good and Bad Musical Behavior

- Positive Reward:
    - Good musical behavior = things that make a composition more listenable, adheres to the most basic rules of *counterpoint*, or the composition of melodies

Examples: tonic notes at the beginning and ending of a melody, notes should not be repeated a certain amount of times, there should be specific intervals between each note

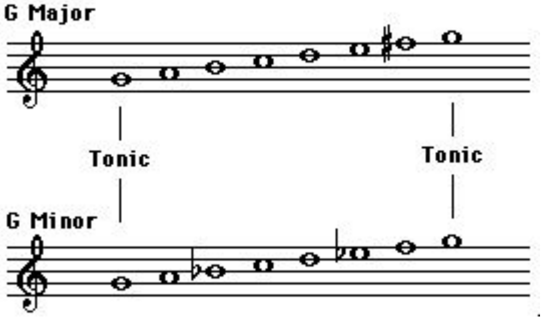


Figure 1: https://people.carleton.edu/~jellinge/m101s12/Pages/12/12ScaleDegrees.html

Figure 2: Arnold Schoenberg, Preliminary Exercises in Counterpoint, Faber and Faber Limited, London, 1963 (First published), 1970 (This edition), ISBN 571 09275 6.

- Negative Reward:
  - Bad musical behavior = things that complicate the composition, increasing the chances that the piece doesn't sound like music

Examples: awkward intervals (augmented/diminished intervals, jumps, intervals greater than an octave), static movement, too many leaps

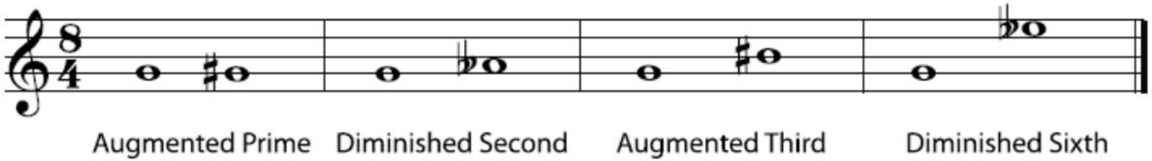


Figure 3: http://www.opentextbooks.org.hk/ditatopic/2315

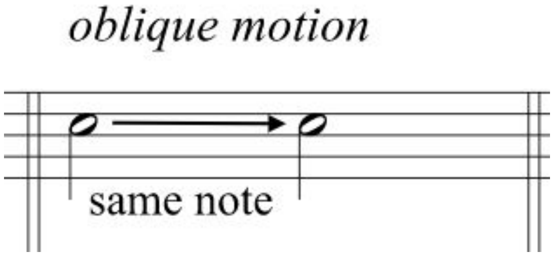


Figure 4: http://musictheory.pugetsound.edu/mt21c/TypesOfMotion.html

Reward Function: Music Theory rules

We define a general music theory reward function $r_{MT}(a, s)$

This function will encourage workable melodies following certain music theory characteristics (according to Gauldin's book on counterpoint):

- Establishing a key
- Tonic notes at beginning and end of the melody
- Penalize repeating notes
- Etc.

Each of these constraints will be written as their own individual reward functions, which we will see demonstrated in the code

Source: A Practical Approach to 18th Century Counterpoint, Robert Gauldin 2013

How it's represented in the code

A snippet of General Reward:

```
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
```

From:Magenta Code (Github)
magenta/magenta/models/rl_tuner/rl_tuner.py, line 1014-1034

Ex: Tonic Reward Function- rewarding beginning and end notes that match the key (note of C for key in C major)

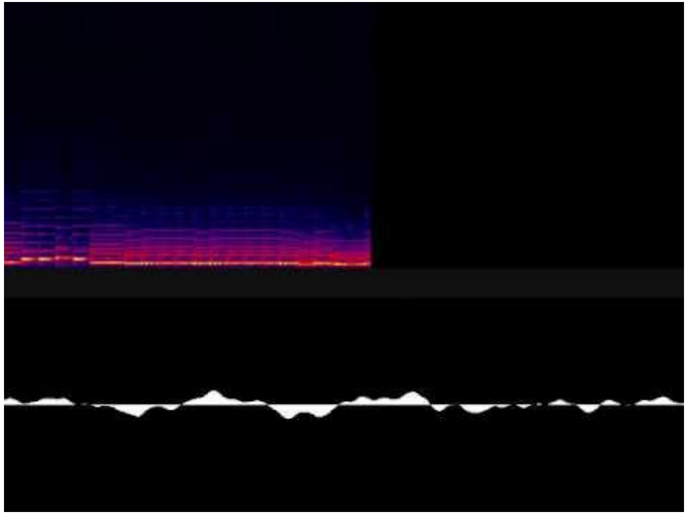
```python
def reward_tonic(self, action, tonic_note=rl_tuner_ops.C_MAJOR_TONIC,
                 reward_amount=3.0):
  """Rewards for playing the tonic note at the right times.

  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
```

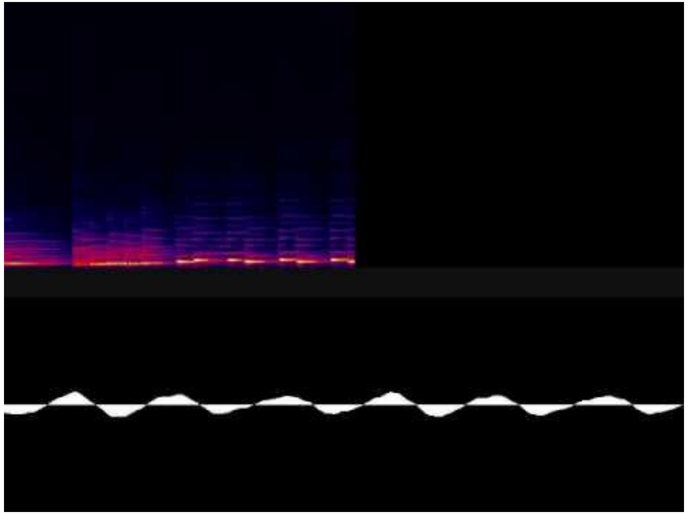
From:Magenta_Code (Github)
magenta/magenta/models/rl_tuner/rl_tuner.py,

Results and Conclusions

[Colab Demo]

https://colab.research.google.com/notebooks/magenta/hello_magenta/hello_magenta.ipynb





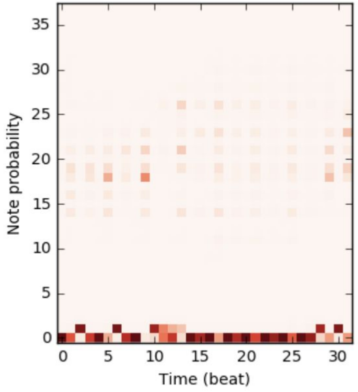
https://magenta.tensorflow.org/2016/11/09/tuning-recurrent-networks-with-reinforcement-learning

Comparing the Models

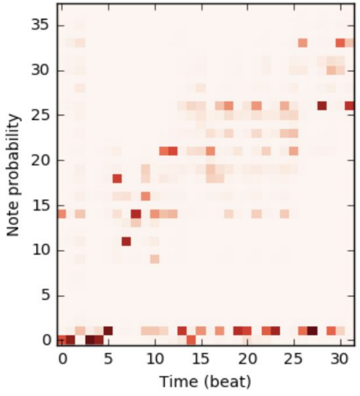
Music Theory Benefits

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%**

Probability Variance w/ Time



Note RNN



Q

https://magenta.tensorflow.org/2016/11/09/tuning-recurrent-networks-with-reinforcement-learning

Future Work

- Possible extensions using g learning and Ψ learning (Jacques et al 2017)
- Multi-agent system could be introduced to generate polyphonic music

Personal and Career Applications

Tassneen
Sally
Max

Thank You!

Questions?