RNN Music Composer

Alvin Vuong & Michael Evans
Psych 186B

The Problem

How do we compose music with a neural network (NN)?

- Feed-Forward model (FFNN) is not sufficient; what to use instead?
- Still supervised learning, but how to calculate our error/cost?
- How to convert MIDI data into a usable format for a NN?
- What constitutes as a “good enough” composition?

Why is it interesting?

- Music composition is intuitively assumed to be a humanly creative fine art
- Humans have to build off of other musicians, so why is a NN different?
- *Can you tell which song is made by a human?*

Past Literature

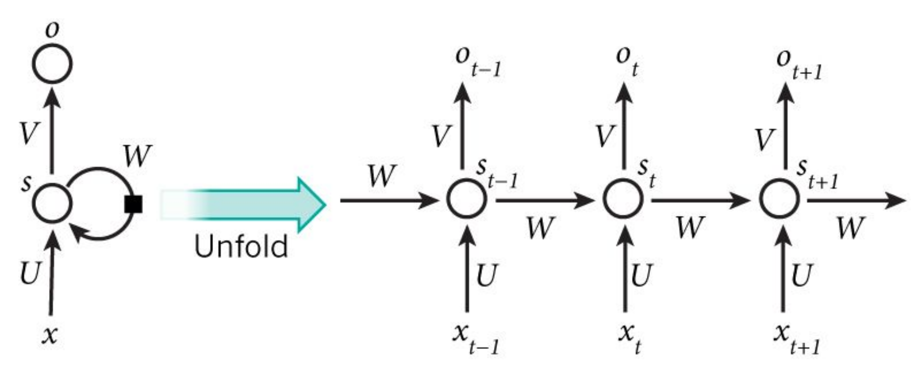
- Acoustic Modeling for Speech Recognition (Hak 2014) (Google)
- Facial Recognition from Video Sequences (Ranzato 2016) (Facebook)
- Character-Level Language Models (Karpathy 2015)

What we did - Overview

- FFNN vs. RNN
  - FF for independent/fixed length data
  - R for sequential & dependent data
- LSTM
  - Vanishing/Exploding Gradients
- Biaxial LSTM RNN

Recurrent NN

- Takes in sequence data, and outputs sequence
- This differs from a FFNN because it can take time into account



Cross-Entropy Cost/Error

- Also used for FFNNs
- Compute cost:
    - Compute output of NN
    - Compare output to original song(s)

Activation function

Regularization Parameter

$$J(\Theta) = -\frac{1}{m}\left[\sum_{i=1}^{m}\sum_{k=1}^{K} y_k^{(i)}\, log((h_\Theta(x^{(i)}))_k) + (1 - y_k^{(i)})\, log(1 - (h_\Theta(x^{(i)}))_k)\right] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(\Theta_{j,i}^{(l)})^2$$

Output

Input

Output

Input

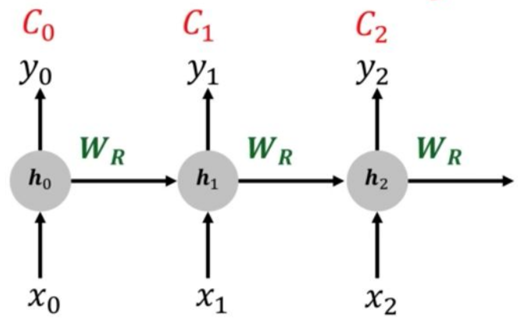
Regularization Term

Vanishing/Exploding Gradients pt 1

- Cost fed into backpropagation algorithm to adjust weights (W_R)
- Used ADADELTA (Zeiler, 2012)

$$\boldsymbol{h}^{(t)} = g_h(W_\mathrm{I}\boldsymbol{x}^{(t)} + W_\mathrm{R}\boldsymbol{h}^{(t-1)} + \boldsymbol{b}_h)$$

$$\boldsymbol{y}^{(t)} = g_y(W_y\boldsymbol{h}^{(t)} + \boldsymbol{b}_y)$$



Combine via:

$$\frac{\partial C}{\partial W_R} = \sum_t \frac{\partial C_t}{\partial W_R}$$

Example gradient:

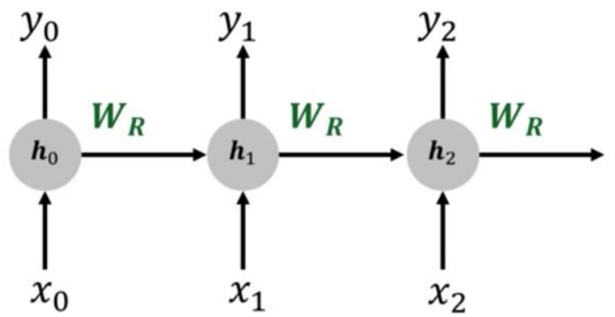
$$\frac{\partial C_2}{\partial W_R} = \frac{\partial C_2}{\partial y_2}\frac{\partial y_2}{\partial h_2}\frac{\partial h_2}{\partial g}\frac{\partial g}{\partial a}\frac{\partial a}{\partial W_R}$$

$$a = (W_\mathrm{I}x_2 + W_\mathrm{R}h_1 + b_h)$$

Depends on W_R too!

Vanishing/Exploding Gradients pt 2

- With too many timesteps, gradients can approach become:
  - very small (too little change)
  - very big (too much change)
- Prevents us from properly training our network

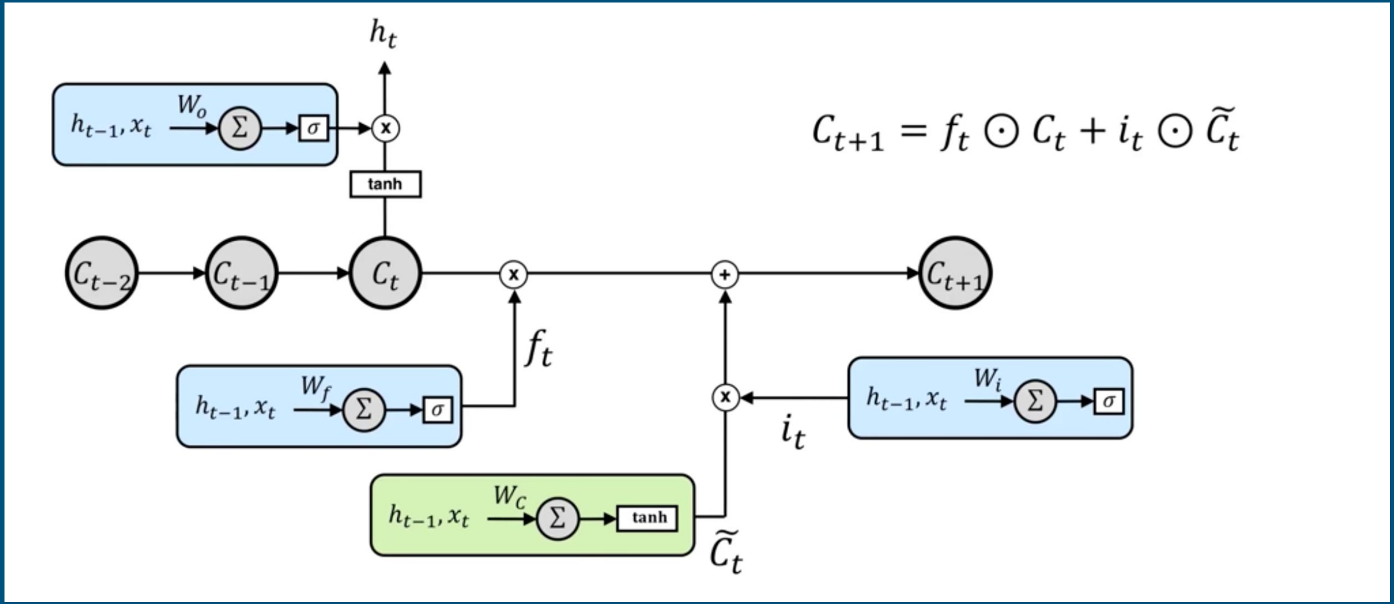


$$\frac{\partial C_{100}}{\partial W_R} = \frac{\partial C_{100}}{\partial y_{100}} \dots W_R \frac{\partial g_{100}}{\partial a_{100}} \dots W_R \frac{\partial g_{99}}{\partial a_{99}} \dots$$

$$\frac{\partial C_T}{\partial W_R} \propto |W_R|^T \left|\frac{\partial g}{\partial a}\right|^T$$

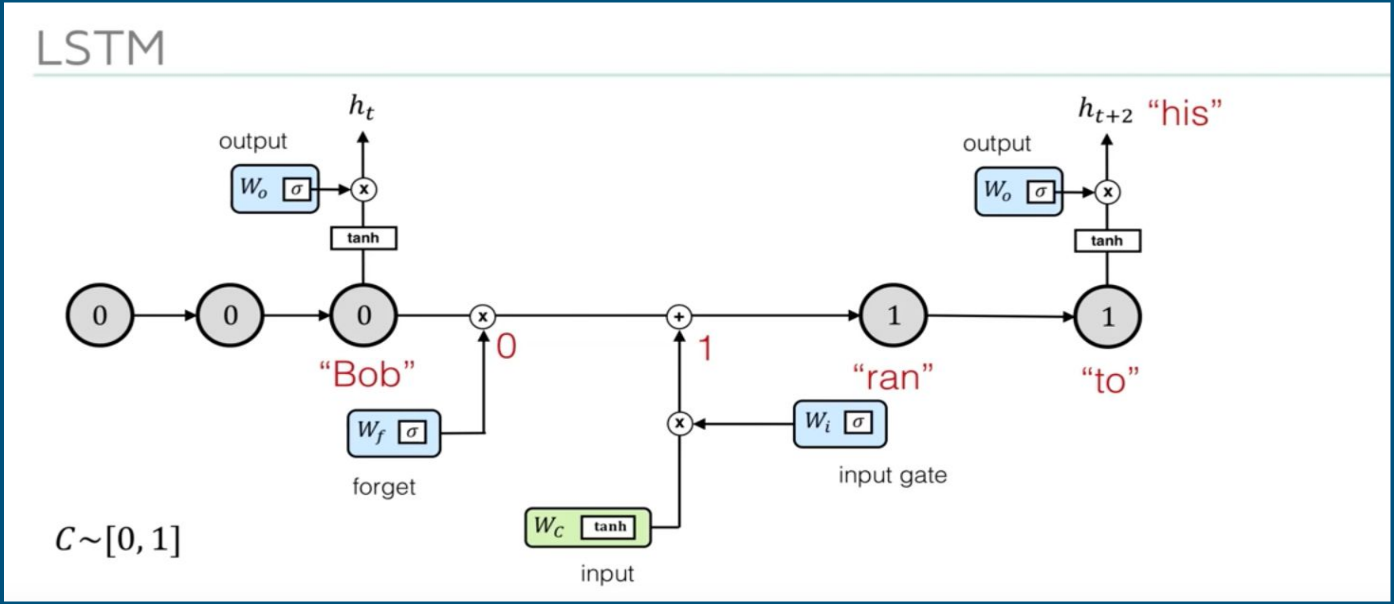
LSTM (Long Short-Term Memory)

- Solves the exploding/vanishing problem by “forgetting”



LSTM Example

- Binary output: Gender of subject (0 = female, 1 = male)



MIDI Files

- 'song.mid'
  - Tracks
    - Events
      - Time signature
      - Key signature
      - **Note**
        - **Ticks (time)**
        - **Pitch (frequency)**
        - Velocity (loudness)

MIDI Parsing

- NoteEvents are converted into binary tuples that represent whether a note is played or articulated (or both) at a certain timestep
  - [1,1] = Played and articulated
  - [1,0] = Played but not articulated
  - [0,0] = Not played or articulated
  - [[0,0], [1,1], [1,0]… [0,0]]   (78 possible notes, so 78 tuples)

- A part of a song is then represented as a list of these lists of tuples
  - [[[0,0], [1,1], … [0,0]],   t = 0
[[0,0], [1,1], … [0,0]],   t = 1
[[0,0], [1,1], … [0,0]], …,   t = 2 ...
[[0,0], [1,1], … [0,0]]]   t = N

MIDI Binarization (Input Layer)

- Need to convert parsed data into usable input to NN
  (1 timestep = 1 input vector)
  - Position: [3] -> C_1 (Index of tuple)
  - Pitch Class: A, Bb, C, C#, D, D#, etc. (Index of tuple % 12)
    [0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0] -> C
  - Previous Vicinity: ± 1 octave in last timestep (played or articulated)
  - Previous Context: number of each pitch class in last timestep (relative to note)
    [1, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0] -> C_{maj}
  - Beat: measures split into 1/16ths (reverse binary encoded)
    [0, 1, 0, 1] -> 10; 10/16 -> 5/8 -> 2/4 + 1/8

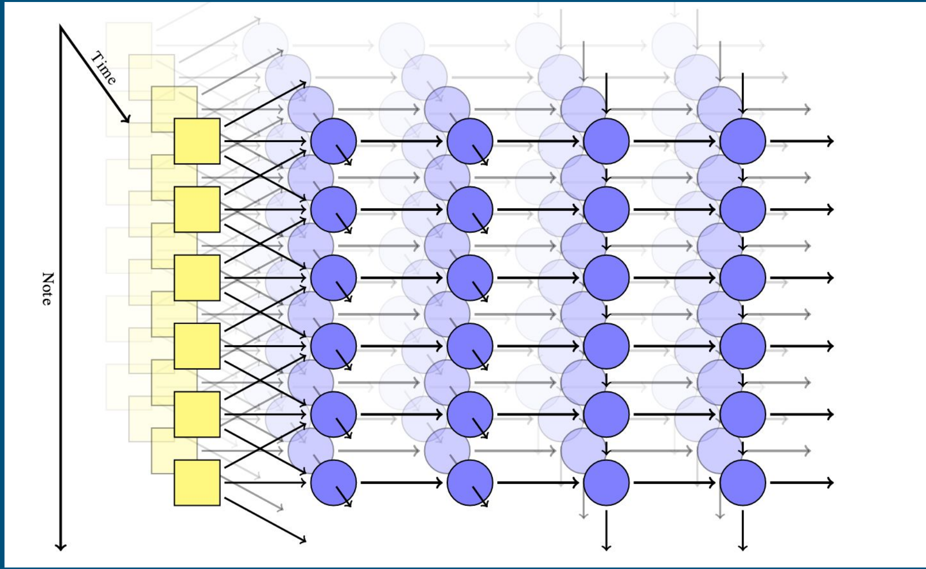
Biaxial NN Architecture (Hidden Layers)

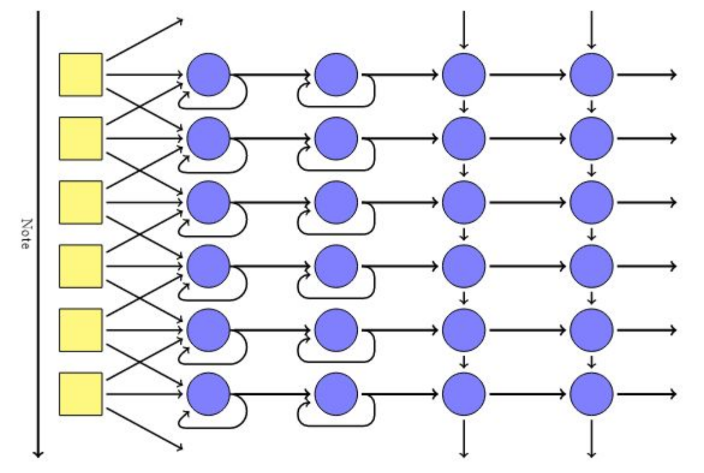
Novel NN architecture: “Biaxial LSTM RNN” (Johnson 2017)

- 2 Time Layers
  - Recurrent in time axis, independent in note axis
  - Learns the sequential patterns (rhythm/beat/melody)

- 2 Pitch Layers
  - Recurrent in note axis, independent in time axis
  - Learns to make notes sound pleasant together (chords/harmony)

Structure of the Model





MIDI Reconstruction (Output Layer)

For each note:

- Play Probability
  - the probability that this note should be chosen to be played
- Articulate Probability
  - the probability the note is articulated, given that it is played.
    (This is only used to determine rearticulation for held notes.)

Results

- How it evolves:
  - Neural network layer sizes
  - Number of iterations
  - Input variability
- Recordings
  - 10, 10, 5, 5, (5 iterations) Error = ~37,000
  - 50, 50, 20, 10, (500 iterations) Error = ~6,000
  - 200, 200, 100, 50, (1900 iterations) Error = ~150
- A composition can be judged using the Turing Test
  - Ask people if it was computer generated or not amongst real samples of songs

Conclusion

- LSTM Recurrent Biaxial Neural Network
- The more songs, the more “average” sounding
- Interesting models do not necessarily have the smallest error
- Implications from this model stretch far beyond music

Breaking the Model

- Loading in too many songs
  - “Music composition by committee”
  - Naturally removes all creativity and variation

- Loading in only one song
  - Error of zero could actually copy the song; this defeats the purpose of the NN

Next Steps

- CSSA Conference - “Where Science Meets the Arts/Senses”
- Train it on various genres of music
- Create a neural network to judge the output
- See if the network learns off of reversed songs
- Reciprocal training neural networks

Works Cited

- Johnson, Daniel (2017). "Generating Polyphonic Music Using Tied Parallel Networks." EvoMusArt 2017. http://www.hexahedria.com/files/2017generatingpolyphonic.pdf
- Kreuger, Bernd (2016). "Classical Piano MIDI Page." http://www.piano-midi.de/midicoll.htm
- Nervana (2016). "Recurrent Neural Networks." https://www.youtube.com/watch?v=Ukgii7Yd_cU
- Ranzato, M., Chopra, S., Auli, M., Zaremba, W. (2016). "Sequence Level Training with Recurrent Neural Networks." ICLR 2016. https://research.fb.com/wp-content/uploads/2016/11/sequence_level_training_with_recurrent_neural_networks.pdf
- Sak, H., Senior, A., Beaufays, F. (2014). "Long Short-Term Memory Recurrent Neural Network Architectures for Large Scale Acoustic Modeling." Interspeech 2014. http://193.6.4.39/~czap/letoltes/IS14/IS2014/PDF/AUTHOR/IS141304.PDF
- Zeiler, M.D. (2012). "ADADELTA: An Adaptive Learning Rate Method." https://arxiv.org/abs/1212.5701