VGM-RNN: RECURRENT NEURAL NETWORKS FOR VIDEO GAME MUSIC GENERATION

PRESENTATION BY NICOLAS MAUTHES

THE IDEA

- Create a recurrent neural network specifically designed to model early video game music
- Train it using a MIDI dataset
- Generate original video game music algorithmically

EXAMPLE

INTRODUCTION

EARLY ALGORITHMIC MUSIC

- Dates back to 18th century with musical dice games of Haydn, Mozart
- Mid-20th century composers experiment with stochastic/indeterminate music (e.g. Cage, Xenakis)
- Hiller and Isaacson compose *Illiac Suite* in 1957 using Markov chains and generative grammars

CONTEMPORARY APPROACHES

- During late 20th century rule-based and expert systems become dominant approach
- E.g. *Experiments in Musical Intelligence* (EMI) by Cope in 1996
- Later Hidden Markov Models (HMMs) become popular
- Deep learning era sees increased interest in neural network-based models

COMPUTATIONAL CREATIVITY

- Relatively new field, but core ideas date from dawn of AI
- Asks whether computers can achieve human-level creativity, create original art
- A major concern is whether computers can create art indistinguishable from humans (artistic Turing test)
- The final frontier of AI? (Colton)

MAGENTA

- Offshoot of Google Brain team, led by Douglas Eck
- Attempt to answer the question, “Can machines be creative?”
- Use a deep learning-based approach to modeling creative activities including drawing, music

BACKGROUND

EARLY VIDEO GAME MUSIC

- Arcade and home console games of 70s and early 80s used simple tone generators (e.g. Atari 2600)
- Commodore 64 (1982) is one of the first computers with a specially-designed sound chip
- Later consoles like Sega Genesis (aka MegaDrive) and SNES use FM-synthesis and sampling respectively
- Pre-CD VGM is generally simpler than recorded music, with fixed number of voices (polyphony)

THE NES

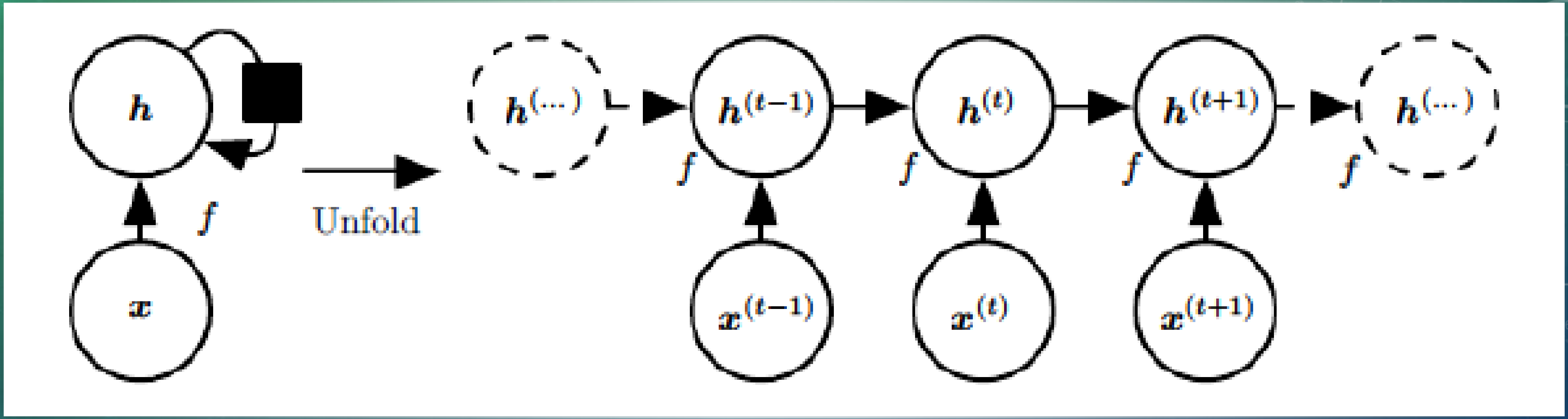
- One of the most popular consoles of all-time
- Games and music are fondly remembered by many
- Ricoh RP2A03 sound chip allowed for 5 simultaneous voices: 2 square, 1 triangle, noise and DPCM
- Theory: Simpler music of NES is easier for RNN to learn
- Dataset is drawn from collection of NES MIDI files downloaded from VGMusic.com

MIDI

- Specification defined by American and Japanese synthesizer manufacturers in 1983
- Originally designed to allow for computer control and synchronization of synthesizers
- All MIDI data (messages) are transmitted serially as 8-bit words
- MIDI files organize messages into tracks, can be played back on most computers that support General MIDI
- Still in use today despite age and somewhat archaic features

RECURRENT NEURAL NETWORKS

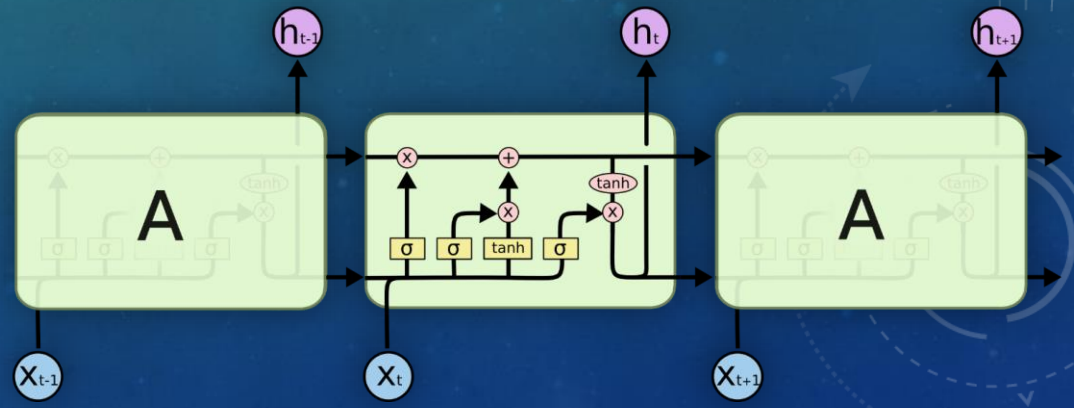
- In contrast to feedforward NNs, RNNs allow previous output to be fed back into network as input
- Operate on sequences where each step represents a time or position
- Information about output at previous steps is propagated forward in time (“memory”)
- To learn correct weights, backpropagation through time (BPTT) is performed on unfolded graph to find gradient of cost



RNN HIDDEN LAYER AS A COMPUTATIONAL GRAPH UNFOLDED ACROSS TIME

LONG SHORT-TERM MEMORY

- With vanilla RNNs there is a limit to how far through time the gradient can be effectively propagated
- This is called "the problem of long-term dependencies"
- LSTM (Hochreiter & Schmidhuber, 1997) helps by adding a memory cell
- Cell contains several gates (e.g. "forget") to help determine what information should be kept or not
- Ubiquitous for sequence-learning tasks; e.g. NLP, machine translation



RELATED WORK

- Eck and Schmidhuber use LSTM to learn blues chords and melody in 2002
- Boulanger-Lewandowski, Bengio and Vincent use RNN-RBM to model polyphonic piano music in “Modeling Temporal Dependencies” (2012)
- Daniel Johnson replaces RBM in Boulanger-Lewandowski architecture with “bi-axial” LSTM (2016)
- Models by Choi and Huang encode music in text format and use language modeling techniques

PROJECT

ACQUIRING THE DATASET

- We create a simple file scraper (*midi_scraper.py*) to collect the dataset from VGMusic.com
- Use *requests* library for HTTP requests and *BeautifulSoup* for parsing HTML
- Start by collecting all the links that end with ".mid" indicating a MIDI file (see below)

```
source = requests.get(args.url).text
soup = BeautifulSoup(source, 'lxml')

links = soup.find_all('a', href=True)
links = [l for l in links if l['href'].endswith('.mid')]
```

- Next iterate through the links and download to specified folder

```
errors = 0
for i, link in enumerate(links[:args.max_files]):
    print(f'Downloading file {i + 1} of {len(links) if args.max_files >= len(links)
else args.max_files}')

    try:
        resp = requests.get(urljoin(args.url, link['href']))
        open(os.path.join(args.data_folder, link['href']), 'wb').write(resp.content)
    except requests.exceptions.Timeout:
        errors += 1
```

PARSING THE DATASET

- We use *pretty_midi* for manipulating MIDI files and *numpy* for arrays
- First filter files in dataset by time signature and key, allow only songs in 4/4 and exclude C major
- Next transpose each MIDI song to common key of C (major or minor)
- Sample each song at 16^{th} note intervals and convert to piano roll matrix (see right)

$$\boldsymbol{P} = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

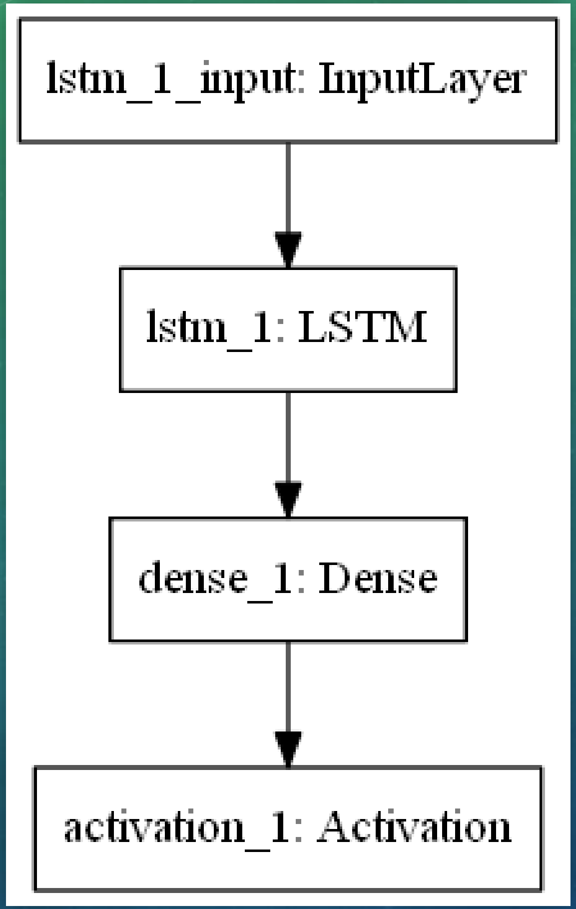
TRAINING

- Before training, we concatenate the piano rolls
- Next, split into training set *X* and set of labels *Y* consisting of sequences of length 64 steps (4 measures)
- During training, a sequence in *X* has as its label the sequence in *Y* immediately following it

```python
def split_xy(data, seq_length):
    x = []
    y = []

    # Split data into training/labels
    for i in range(0, len(data) - seq_length, seq_length):
        x.append(data[i:i + seq_length])
        y.append(data[i + seq_length: i + seq_length * 2])

    x = np.asarray(x)
    y = np.asarray(y)

    return x, y
```

TRAINING CONTD.

- We use Keras with TensorFlow backend to build network, for training and generation
- Use Adam as optimizer and categorical cross-entropy as cost function
- LSTM layer has 256 units, during training we use a learning rate of 0.01 and train in batches of 50
- Generates coherent output after ~20-50 epochs
- Training typically takes 1-1.5 hours using *tensorflow_gpu*



MODEL ARCHITECTURE

GENERATION

- Once model is trained, we can use it to generate new sequences
- Choose a primer sequence at random from the dataset
- After rebuilding model and loading trained weights, predict a new sequence given primer sequence
- Output is matrix of note probabilities; to get piano roll, threshold the probability matrix (we use threshold of 0.35)
- Finally, convert piano roll to *pretty_midi* then write to disk as MIDI file

```
note_probs = model.predict(primer_sequence)[0]

piano_roll = prob_matrix_to_piano_roll(note_probs, threshold=SAMPLING_THRESHOLD)
generated_mid = piano_roll_to_pretty_midi(piano_roll, subdivision=SUBDIVISION,
                                          program=MIDI_PROGRAM,
                                          pitch_offset=MIN_MIDI_NOTE)

generated_mid.write(os.path.join(GENERATED_MIDI_FOLDER, args.generated_filename))
```

CODE TO GENERATE A NEW MIDI SEQUENCE

RESULTS

EXPERIMENTS

- We found that generated sequences exhibit similarity and coherence with regard to primer sequences
- Model tends to learn tonality well (plays correct notes in key)
- Has more difficulty learning rhythmic structure, rhythms tend to wander
- We found that increasing complexity of model/adding regularization (e.g. dropout) had negative effect on quality of output
- It has been shown (e.g. by Karpathy) that sometimes even simple RNN models can achieve good results

EXAMPLE OF GENERATED OUTPUT

SURVEY

- To evaluate output of model, we conduct a survey using Google Forms
- Participants asked about musical experience, i.e. whether they listen to music casually, play an instrument, or have studied music theory
- Next, shown five 10-second clips of music either generated by model or from training dataset
- Asked to rate for quality from 1 (“Worst thing I’ve ever heard”) to 10 (“I love it”)
- Also asked whether it was composed by a human or computer (musical Turing test)

VGM-RNN - Clip 1

Please listen to the first clip below, then answer the following questions:

How would you rate this clip on a scale of 1-10?

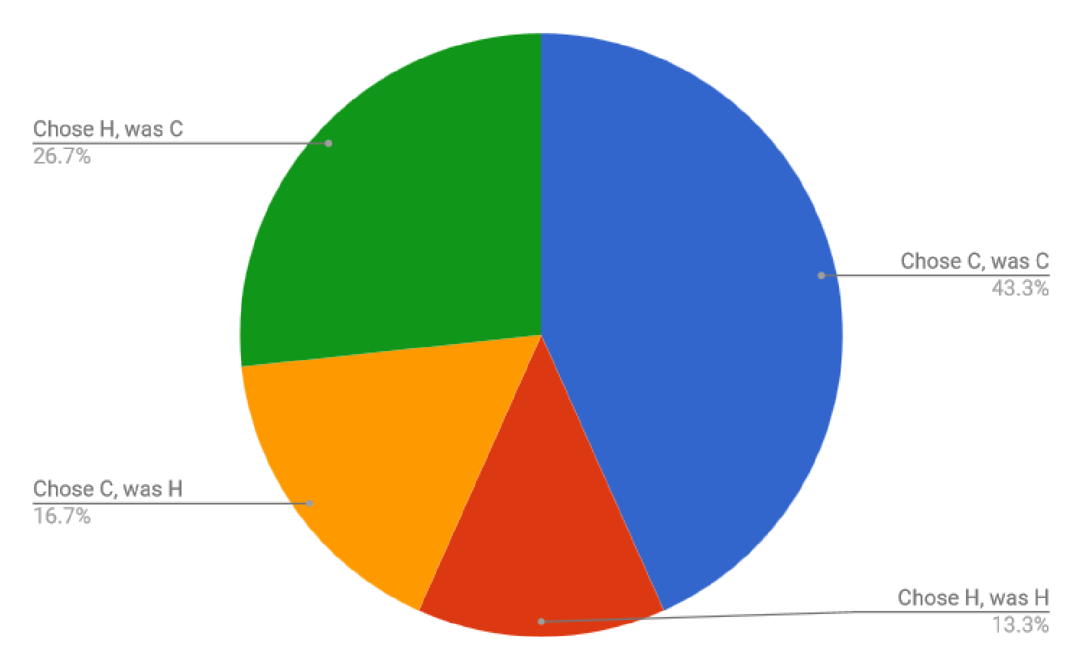
	1	2	3	4	5	6	7	8	9	10	
Worst thing I've ever heard	○	○	○	○	○	○	○	○	○	○	I love it

Do you think this clip was created by a human or a computer?

- ○ Human
- ○ Computer

SURVEY RESULTS

- In total, six respondents; fewer than expected but time constraints preclude larger study
- Not much musical experience (expected since mostly CS students)
- Average rating for quality was 5.56, highest rating was 6.33 for clip generated by system
- Respondents only able to correctly distinguish between clips 56.6% of the time, similar to BachBot



CONCLUSION

GENERAL CONCLUSIONS

- We found that network was able to model features of melody, harmony, rhythm
- Survey participants could not reliably differentiate between music generated by system and music created by humans
- More layers does not always equal better results
- Model could certainly be improved, but results are promising

APPLICATIONS

- Could be applied for automatic music transcription?
- Maybe not, since VGM transcriptions are widely available
- More interesting application might be for generating music for independent game designers on a budget
- Retro-style games are popular, so there could be demand for early VGM style

FUTURE WORK

- Currently only melodic instruments are considered, future implementations could also model drums
- Note durations and short-term rhythmic structure could also be modeled
- Model could be trained on other datasets (e.g. SNES, Genesis)
- More sophisticated architecture (e.g. biaxial LSTM) or representation (e.g. symbolic language modeling, embedding)


GAME
OVER