Seq2SeqP4P: A Sequence-to-Sequence model for Monophonic Music Continuation

Eric Nichols
Microsoft
Eric.Nichols@microsoft.com

ABSTRACT

I developed a sequence-to-sequence (seq2seq) neural network model for the 2018 MIREX "Patterns for Prediction" task. This entry addresses the monophonic version of task 1 (sequence continuation); task 2 and polyphonic inputs are not handled. The model takes symbolic (MIDI-style) inputs and produces a short continuation to follow the input. Inputs are preprocessed into a format composed of *note-on*, *note-off*, and *time-shift* commands inspired by PerformanceRNN [3]. The chosen format is straightforward to represent for neural network purposes as it consists of three separate one-hot-encoded input vectors; this format also is used to represent the output for the model. The model is a seq2seq model using two stacked LSTM layers of 1024 nodes in each layer, trained using teacher forcing.

1. INTRODUCTION

The name of this task, "Patterns for Prediction" suggests that the goal is to predict musical continuations by discovering repeated patterns. I chose not to try to solve the pattern-recognition problem explicitly (*e.g.*, by writing algorithms to explicitly do pattern-matching), and instead attempted to have a machine learning model learn how to map inputs (priming sequences) to outputs (true continuations) in an end-to-end manner.

In the spirit of end-to-end learning, no musically-aware features were computed based on the input symbolic data, which consisted of time-stamped *note-on* and *note-off* commands with associated MIDI numbers (other metadata such as estimated key was provided, but ignored).

2. INPUT DATA

The input data was transformed into a sequence of *note-on*, *note-off*, and *time-shift* commands, and each input score was represented as a sequence of these commands. Each command takes one argument. For *note-on* and *note-off*, the argument is the integer MIDI number of the note in question (restricted to the range of the 88 notes on the piano). For *time-shift*, the argument is the number of sub-beats to move forward in time, in the range 0–48. Time is quantized to 12 sub-beats per beat, thus the duration can be between 0 and 4 beats, in steps of 1/12 notes (thus allowing for certain triplets as well as quarter, eighth, sixteenth, and thirty-second notes). For example, a performance of an eighth-note "C" followed by a dotted-quarter rest would be represented as the sequence:

[*note-on*(48), *time-shift*(6), *note-on*(48), *time-shift*(18)]

Each command is represented as a binary vector, separated into three components: *command*, *midi*, *duration*. Each of these components is represented as a one-hot vector, with each position in the vector corresponding to one of the possible values of that item. For instance, *command* consists of three possibilities, midi consists of 89 (88 + 1 for n/a), and time-shift consists of 49. For an irrelevant component (for example, midi number for a *time-shift* command), a "0" or "n/a" value of that component is selected. The three one-hot sub-vectors are concatenated together to give a single binary command vector. A score is represented as a time series of these command vectors.

For simplicity and training efficiency, the model was designed to take in a fixed number of input notes (30) and to output a fixed number of output notes (10). The task gives us an indefinite number of input notes and a fixed *duration* of 10 beats of output, so the model doesn't exactly align with the input data. Thus, the input data was sliced up to generate training and evaluation datasets of the required sizes.

For each score in the provided training data, a sliding window of size 40 notes (30 inputs, 10 outputs) was used to generate examples. The window was moved with a hop size of 5 notes to reduce the number of output examples compared with a size-1 sliding window. This resulted in a dataset of over 2.6 million input/output example pairs, of which 80% were used for training and 20% for testing (with care taken to split the dataset at the score level to avoid leakage).

3. MODEL

A sequence-to-sequence (seq2seq) model [4] was trained to model the relationship between the inputs and outputs

in the dataset. The encoder of the model consists of two stacked LSTM layers, each consisting of 1024 nodes. Each training sequence is input to the encoder, and after reading the entire input, the hidden states of the two LSTM nodes (*i.e.*, two vectors of length 1024 each) are fed into the decoder network, which also consists of two LSTM layers. The decoder outputs three separate vectors, each one-hot-encoded, representing the components of the commands. Each output component vector is compared with the desired target output, and a loss is computed via the cross-entropy loss function. Losses for the three components are summed together to give the total network loss. The network is trained via backpropagation-through-time using the Adam optimizer. Gradient clipping is used. The final model consists of ~26.5M parameters.

After training, we run the model in inference mode to generate sequence continuations. As this model requires exactly 30 input notes, if the input is too long is it truncated to the final 30 notes; if too short, the source material is duplicated repeatedly until the desired length is achieved. Sequence continuations were generated by following the same procedure as above of presenting the input data in the required form to the encoder, but then running the decoder one timestep at a time until 10 beats of time have elapsed in the output. After each timestep, the argmax of each component in the output vector of the decoder is computed to give the output command. This command (in one-hot-encoded binary form, as before) is fed back into the decoder network as an input for the next time stamp.

A command-line program was implemented to operate on a batch of input score files (in CSV format) and output the continuations generated by the model.

The model was developed using the Keras library [2] running with the Tensorflow [1] backend.

4. EVALUATION

Evaluation is in progress.

5. DISCUSSION

The model appears to learn to generate pitches in the correct key of the source material. Sometimes the model outputs static pitches (*e.g.*, every note might have the same pitch), but in other cases it outputs more interesting sequences of pitches, which may mirror patterns in the input sequence. Rhythmic outputs are not very interesting, but they do depend on the input material. The model tends to output the same note duration and rest duration over and over.

6. REFERENCES

[1] Abadi, M. et al.: "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015. Software available from tensorflow.org.

[2] Chollet, F. et al.: "Keras," 2015. Software available from http://keras.io.

[3] Simon, I and Oore, S: "Performance RNN: Generating Music with Expressive Timing and Dynamics," Magenta Blog, 2017. https://magenta.tensorflow.org/performance-rnn

[4] Sutskever, I., Vinyals, O., Le, Q.: "Sequence to Sequence Learning with Neural Networks," *Advances in Neural Information Processing Systems.* pp. 3104–3112, 2014.