MAIS Deliverable 3: AI-composed music

Nicholas Connors

Final Training Results

Since the last deliverable, I've finalized what the goal of my project is: to use an LSTM to supply note and chord suggestions in a music editing web-app. More importantly, I've gotten proper results from my LSTM to discuss. The LSTM model was trained on classical music. After 5 epochs, it stopped generating silence and could generate simple sequences such as a single repeated note. After 19 epochs, it could generate relatively pleasing snippets of music. The loss function used was categorical cross-entropy after a softmax output layer. The LSTM was tested with different numbers of hidden layers and hidden layer nodes, and with different learning rates.

Performance

With 5 or less epochs, the LSTM tends to output very simple strings of notes (Fig. 1). As the model improves, it plays a wider array of sounds but still tends to get stuck looping (Fig. 2). Additional hyper-parameter tuning and more training time will likely alleviate this.

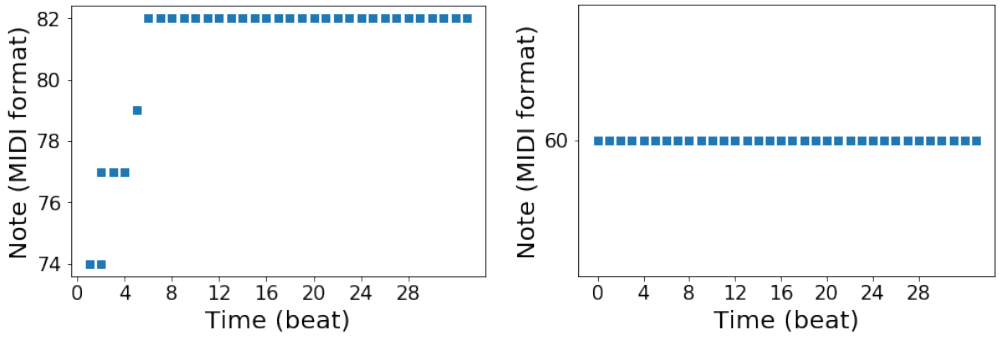

Figure 1: Results from two different models after 5 epochs. LEFT: Two LSTM layers. RIGHT: One LSTM layer. These results were selected from 9 other results for each model. Every other result was just silence (string of repeating time-step-increment tokens).

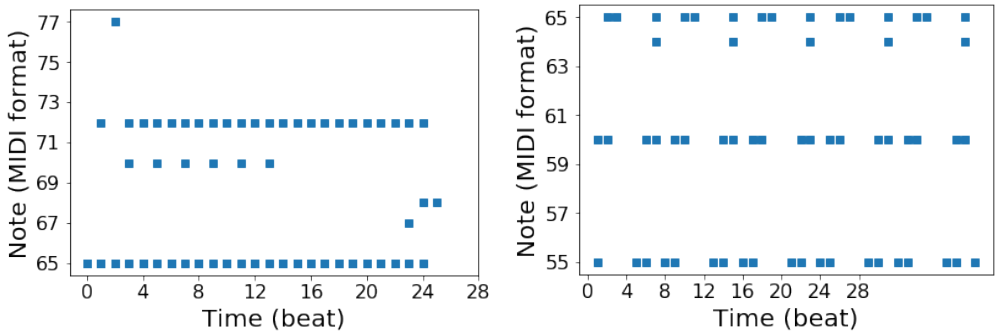

Figure 2: Results from two different models. LEFT: Two LSTM layers (512 and 256 nodes respectively) after 19 epochs. Two notes are repeated with some embellishments. RIGHT: One LSTM layer of 256 nodes after 10 epochs. A two measure sequence is continuously repeated.

Loss function

The model uses a softmax output activation function, and the loss is calculated as categorical cross-entropy loss. This outputs an array of probabilities for the 128 notes (plus the timestep increment token). This will be useful when implementing the AI, as the top three predictions can be output to the user.

Hyper-parameter tuning

In tuning the hyper-parameters, I focused on the size and number of LSTM layers and the learning rate. Due to the long training time for each model, I was not able to tune these parameters as much as I would have liked, nor was I able to train them until their loses converged to a value.

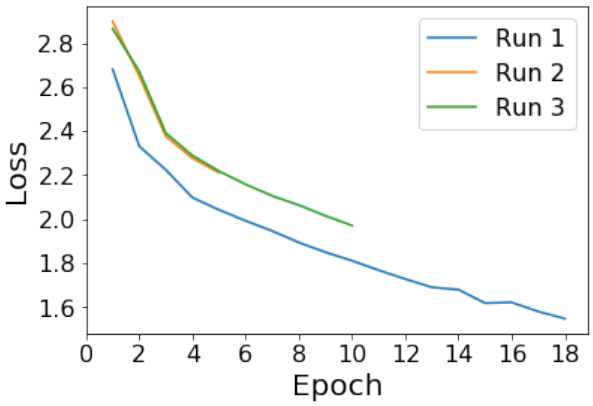

Figure 3: A comparison of models with different LSTM layers. Run 1 is based off a model with two LSTM layers with 512 and 256 nodes; Run 2 has two LSTM layers with 256 and 128 nodes; and Run 1 has a single LSTM node with 256 nodes. We can see that the number of layers seems to have no effect (comparing Runs 2 and 3), and the number of nodes offsets the shape of the curve (comparison between Run 1 and 2).

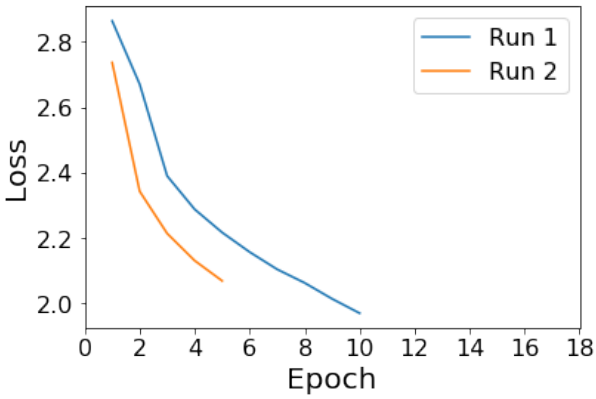

Figure 4: A comparison of models with different learning rates. Both models have a single LSTM layer with 256 nodes. Run 1 has a learning rate of 0.001. Run 2 has a learning rate of 0.005.

Conclusion

The LSTM model outputs satisfactory music snippets. Additionally, since the plan is to implement these as suggestions for a music composing web app, the music outputted by the AI does not necessarily have to stand on its own. In this sense, the current state of the model has accomplished what it needs to. However, improving the quality of the music output by further investigating the best hyper-parameter settings and then training a well designed model for longer and on more data would definitely improve its application, time permitting.

Final demonstration proposal

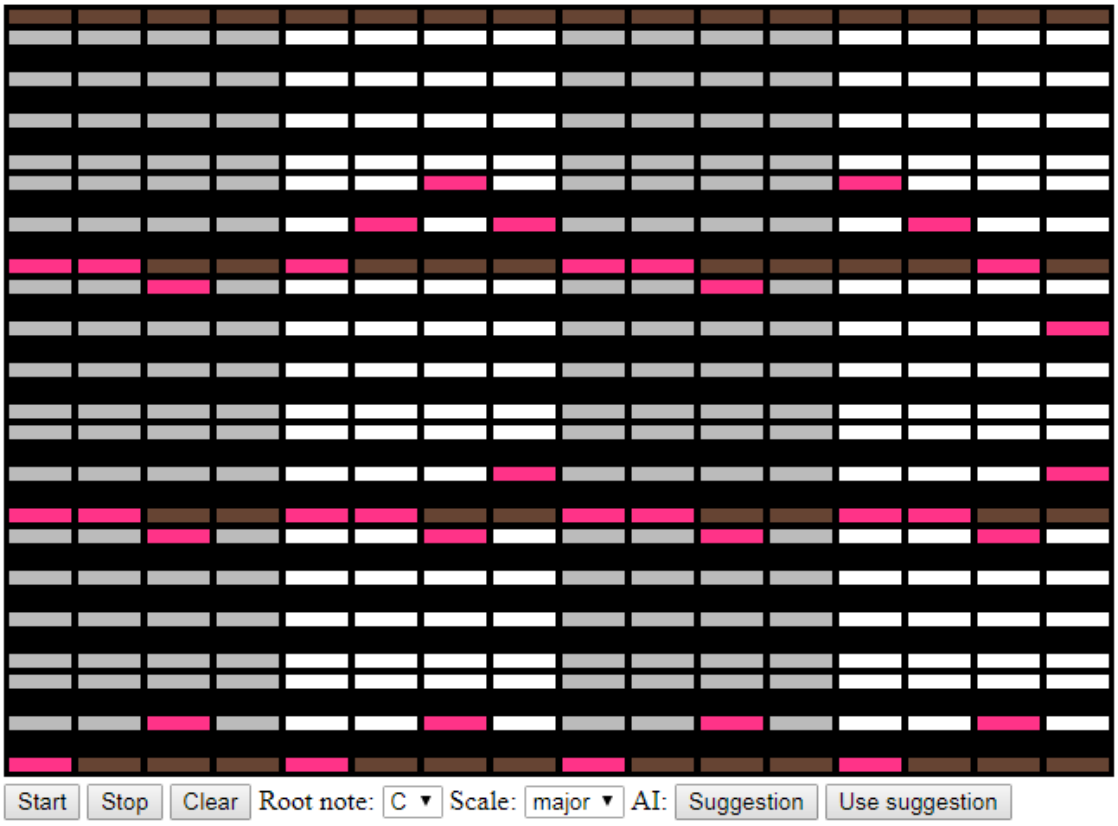

Figure 5: Current set-up for an in browser melody editor written in JS and using an HTML5 canvas. The user is given a range from the 3rd to 6th octaves. The root note of the composition can be chosen (by default it is middle C). A scale can be shown on the screen (current options are minor and major) to help the user play in key. Most importantly, the user can request an AI suggestion (similar to using predictive text). Currently the AI suggestions are not implemented and are given by random number generation.

Web-app

I plan to integrate my LSTM into a music-editing webapp (see above). I am finished making a basic prototype of the application using Tone.js. For now the user can only make a single 4 measure section of a song. I plan to allow the user to create multiple sections at once which can be arranged or repeated to make a song. Aside from that, and the major step of integrating AI predictions into the app, there are a few things I would like to implement to make the app more useful:

- Allow adjustment of song tempo using a slider.
- Add a selection of instruments.
- Allow loading and saving songs in MIDI format.
- Allow exporting songs to .wav or similar format.

Integration

To use the neural network (written in Python) in conjunction with the webapp (written in JS) I plan to use Flask. I have never used Flask before, and as such learning how to use it and ultimately integrating the AI will be the hardest step of what I have left to complete. The integrated AI will need to:

- Load the saved weights.
- Access the last 100 tokens (notes or time-steps) of the user's song before a given point.
- Return a list of 3 predictions.

Additionally, the predictions made by the AI will have to be of variable length, and include notes up until a time-step is predicted. This way the AI will be able to suggest either individual notes or chords.