Magenta Studio: Augmenting Creativity with Deep Learning in Ableton Live

Adam Roberts[1], Jesse Engel[1], Yotam Mann[1], Jon Gillick[2], Claire Kayacik[1]
Signe Nørly[1], Monica Dinculescu[1], Carey Radebaugh[1], Curtis Hawthorne[1], Douglas Eck[1]
[1]Google, Mountain View, CA, USA
[2]University of California, Berkeley, CA, USA

Abstract

The field of Musical Metacreation (MuMe) has produced impressive results for both autonomous and interactive creativity, recently aided by modern deep learning frameworks. However, there are few examples of these systems crossing over to the "mainstream" of music creation and consumption. We tie together existing frameworks (Electron, TensorFlow.js, and Max For Live) to develop a system whose purpose is to bring the promise of interactive MuMe to the realm of professional music creators. Combining compelling applications of deep learning-based music generation with a focus on ease of installation and use in a popular DAW, we hope to expose more musicians and producers to the potential of using such systems in their creative workflows. Our suite of plug-ins for Ableton Live, named Magenta Studio, is available for download at `http://g.co/magenta/studio` along with its open source implementation.

Introduction

While the mainstream discussion on artificial intelligence (AI) has focused on its potential to displace human workers, the concept of Artificial Intelligence Augmentation (AIA) promotes the use of this technology to enhance their productive capacity (Carter and Nielsen, 2017). In this vein, Pasquier et al. (2016) point out that "bringing greater automation and artificial intelligence to creative software is an important goal of computational creativity". Successful examples in the space of generative music include Jam Factory (Zicarelli, 1987), the Continuator (Pachet, 2003), Jnana (Sullivan, 2012), and FlowComposer (Papadopoulos, Roy, and Pachet, 2016).

In recent years, there have been incredible developments in Musical Metacreation (MuMe) thanks in part to advances in deep learning (Hadjeres, Pachet, and Nielsen, 2017; Roberts et al., 2018; Huang et al., 2019). Deep neural networks enable the direct learning of the structure of musical data, requiring fewer hand-tuned features and heuristics. Reducing the reliance on music theory can result in new creative tools using generative models that are less constrained by imposed sets of rules, which may themselves be based on a particular style or genre. Considering the power of deep learning applied to MuMe, it may seem surprising that more of this technology has not made its way into the hands of musicians and producers. One explanation is that the worlds of applied creativity and deep learning research speak different languages, both literally and figuratively. While commonly-used Digital Audio Workstations (DAWs) can interact with Virtual Studio Technology (VST) plug-ins written in C++ or proprietary languages like Max, state-of-the-art generative models for music are often developed in deep learning frameworks such as TensorFlow (Abadi et al., 2015) or PyTorch (Paszke et al., 2017) and typically require specialized hardware accelerators to enable real-time applications.

In this paper we introduce Magenta Studio, a suite of deep learning-based co-creative plug-ins that brings a first-in-class modern machine learning framework (TensorFlow) into a first-in-class modern music production environment (Ableton Live[1]). By creating a bridge between Max For Live[2] and TensorFlow.js (Smilkov et al., 2019), we demonstrate how it is possible to package complex and computationally expensive music generation models into easy-to-use, interactive interfaces for human creators. Furthermore, our open-source JavaScript library, Magenta.js (Roberts, Hawthorne, and Simon, 2018), contains a large collection of TensorFlow.js implementations of state-of-the-art music generation models with a simple API, providing a communal toolbox of components for plug-in developers to use in creating novel interfaces. The combination of Magenta.js and Magenta Studio help fulfill three of the key developments in the MuMe roadmap (Pasquier et al., 2016): *Deployment and Accessibility*, *Standardization and Interoperability*, and *Real-World Applications*.

In the following sections we describe the design and implementation of this framework, the machine learning models used, and the five plug-ins initially launched as part of the suite. We then share the results of an unscientific survey of early adopters and discuss our findings. We close with a discussion of how our approach can be adopted by other researchers to provide an enhanced experience for creators while gaining valuable feedback from real-world users.

[1]`https://ableton.com/live/`
[2]`https://ableton.com/live/max-for-live/`

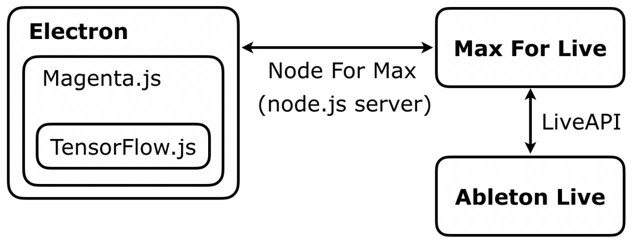

Figure 1: Overview of the Magenta Studio architecture.

Design

We based our design on five major principles, which we arrived at after an extensive design study detailed in Kayacik et al. (2019):

1. Use an environment and UI vocabulary target users were already familiar with.
2. Make it easy to install on operating systems and hardware typically available to our target users.
3. Provide simple interfaces that could be understood without requiring in-depth knowledge of the underlying methods.
4. Do not treat model outputs as sacrosanct. It's ultimately the user's creation, and they should be able to modify or even ignore the output.
5. Provide modular functions that can be used together and with other tools in arbitrary ways to support novel creative workflows.

By building our applications as plug-ins to Ableton Live–one of the most popular DAWs for music professionals–we were guaranteed to meet principles 1 and 2.

While the number of possible interfaces and models are vast (see the wide variety of Magenta.js demos[3]), we focused on 5 simple interactions (see the Plug-ins section) that would be easy to understand and use in order to satisfy principle 3. Each interface takes at most two inputs and has no more than three sliders (Figure 2). The web instructions[4] explain in basic terms the functionality of each, short videos provide examples, and even the animated plug-in titles hint at their purposes.

Limiting ourselves to simple interactions with the models, however, does not inhibit the creative potential of these plug-ins. Ableton provides a modular production workspace in which ideas can be explored by applying effects and modifications to clips, allowing for intermediate outputs to be saved and discarded as needed. We exploit the clip-based approach of Ableton in particular by reading and writing MIDI clips directly from the Session View (see Figure 3), encouraging iterative and playful interaction over more traditional "linear" composition.

[3]http://g.co/magenta/demos

[4]http://g.co/magenta/studio

This Session View also helps satisfy principle 4 by giving us a place to show the user various outputs generated by our models for a given input (due to inherent stochasticity of the models). The user can choose to keep one or more of the output clips, modify them manually as they see fit, or delete them all and try again. The user can also continue modifying the pieces automatically by passing them through the models multiple times or using other plug-ins and functionality provided by Ableton, satisfying principle 5.

Implementation

Ableton, like most DAWs, is closed-source proprietary software with little opportunity for direct integration. VST plug-ins written in C++ are a common form of integration, but they are based on a signal flow paradigm for modifying audio and MIDI in a streaming fashion, not for generating or performing transformations to entire MIDI clips. Also, integrating hardware-accelerated deep learning frameworks such as TensorFlow into VST platforms like JUCE[5] is a significant undertaking. The Live API[6] offers rich interaction features such as modifying and adding MIDI clips within Ableton Live, but it is only well-supported on a few platforms such as Max For Live.

Max For Live contains support for JavaScript, which some generative apps such as Jnana (Sullivan, 2012) have taken advantage of the past. However, JavaScript is too inefficient to run large generative models and Max's implementation lacks support for WebGL[7], which deep learning frameworks like TensorFlow.js use to accelerate computation. We hope Ableton will add support for WebGL in the future, but in the meantime our solution to get the best of cross-platform hardware acceleration and rich feature integration was to build the app using a combination of three frameworks: Electron[8], TensorFlow.js, and Max For Live (see Figure 1).

Electron is a framework that essentially enables a website to be packaged into a native application, providing a few advantages. First, it allows us to use TensorFlow.js, which implements most basic TensorFlow operations in WebGL without requiring users to install any additional tools or GPU drivers. Second, Electron allows us to develop the interface in familiar web technologies, such as HTML and CSS. Finally, Electron binaries can be built for multiple platforms from the same source code, allowing us to support users regardless of their OS.

However, Electron has no way of communicating directly with Ableton Live for reading and writing MIDI clips, so we use Max For Live as an intermediate layer between Live and the Electron plug-ins. Max 8 added the ability to run Node.js[9] with the new Node For Max API[10], which we use to

[5]https://juce.com/

[6]https://docs.cycling74.com/max6/dynamic/c74_docs.html#live_api_overview

[7]http://webgl.org

[8]https://electronjs.org/

[9]https://nodejs.org

[10]https://docs.cycling74.com/nodeformax/api/

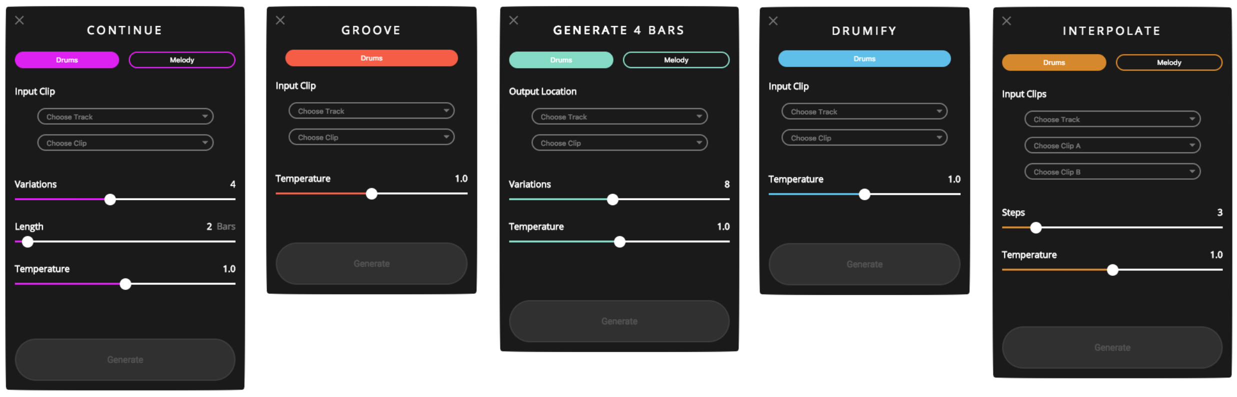

Figure 2: The five initial plug-ins in Magenta Studio. *Continue* extends a MIDI sequence with an LSTM model, *Groove* "humanizes" a drum beat with a GrooVAE, *Generate* samples 4 bars at random from a MusicVAE, *Drumify* maps an arbitrary MIDI sequence into a drum groove using a GrooVAE, and *Interpolate* morphs between a pair of sequences with a MusicVAE.

create a local server. The Electron apps then sends requests over the local network to Max via HTTP, which makes calls to the Live API to execute operations such as finding the notes in a clip or creating a new clip. This approach is somewhat indirect, but because the messages are small and over a local network, it works quickly enough to support real-time interactivity.

Models

The underlying generation in Magenta Studio is provided by the Magenta.js library, a high-level API that implements several state-of-the-art music generation models using TensorFlow.js. This library operates on a MIDI-like representation called a NoteSequence. Below we describe the two model architectures used in the initial Magenta Studio release.

MusicRNN

Since the early 1990s, researchers have experimented with neural network music composition based on language modelling, including simple recurrent neural networks (RNNs) (Todd, 1991), RNNs trained using backpropagation through time (Mozer, 1994), and long short-term memory (LSTM) networks (Eck and Schmidhuber, 2002). See Briot, Hadjeres, and Pachet (2017) for a full overview.

Magenta.js contains a `MusicRNN` class based on a combination of ideas from this body of work, including models for melodies and drum patterns using multi-layer LSTMs with the addition of attention (Waite, 2017) for better capturing long-range dependencies. A typical interaction for this type of model is to extend a priming sequence, similar to what has been done in the past with Markov chains (Brooks et al., 1957; Zicarelli, 1987; Cope, 1996; Pachet, 2003). Since the base models are not conditioned by chord, key, or style, the primer is teacher-forced into the LSTM to set its state before sampling begins. By training on a wide variety of music, the models avoid collapsing to a single style, which enables the user to effectively control the the output by conditioning it with their primer.

We have additionally trained "ImprovRNN" models (Simon, 2016) that accept a chord sequence as a conditioning signal, which is supplied to the LSTM as a side input. This allows the user to to further guide generation based on a chord progression of their choosing. While these chord-conditioned models do not yet appear in Magenta Studio, we plan to increase the amount of user control in future versions with models such as these.

MusicVAE

MusicVAE is a more recently-developed neural network for music generation, with an architecture that is described as a hierarchical recurrent variational autoencoder in Roberts et al. (2018).

Autoencoders support a different set of interactions than RNN language models. A musical sequence can be encoded into latent vector, which is essentially a learned set of qualities that describe the input. The latent vector can then be decoded back into a musical sequence that emphasizes those qualities. These two transformations (encode and decode) are sufficient to enable a rich set of operations including *morphing*–interpolating between the qualities of two musical sequences–and *variation*–adjusting specific qualities with attribute vector arithmetic or latent constraints (Engel, Hoffman, and Roberts, 2018). Because the latent vectors of a variational autoencoder (VAE) are regularized to be similar to a standard normal distribution, it is also possible to *sample* from the distribution of sequences, generating realistic music based on a random combination of qualities.

Furthermore, we can train MusicVAE models to map from

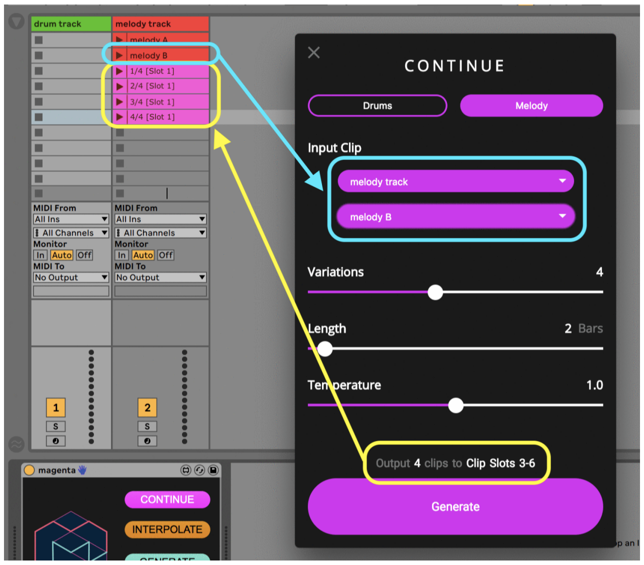

Figure 3: Example usage of the Continue plug-in. The user selects the input clip (annotated in blue) and the plug-in outputs multiple continuations in the same track (annotated in yellow). The *Variations* slider determines the number of output clips, *Length* specifies their length, and *Temperature* allows the user to adjust how "random" the samples will sound.

lossy representations to the full sequences, thus teaching it to essentially "fill in" the missing details. For example, GrooVAE is a variant of MusicVAE (Gillick et al., 2019) that maps lossy drum patterns back to the original human-performed grooves, complete with velocities and microtimings. An example of a lossy function is to quantize the performed inputs and remove the velocities, resulting in a "humanize" model that can add performance characteristics to a programmed beat.

Magenta.js includes a `MusicVAE` class with models for melodies, drum patterns and grooves, and multi-instrument sequences.

Plug-ins

Our initial release contains five plug-ins that exemplify three different creative musical tasks in cooperation with a musician or producer: composition, interpretation, and accompaniment. Examples of outputs generated using these tools can be found at `https://goo.gl/magenta/studio-examples`.

Generate

Generate is a compositional tool that produces a 4-bar phrase with no input necessary. The user chooses where the output should go, the number of variations, a softmax temperature to adjust randomness, and clicks "Generate". This plug-in can be helpful for breaking a creative block or as a source of inspiration for an original composition.

Under the hood, Generate uses a MusicVAE that has been trained on millions of melodies and rhythms to learn a summarized representation of musical qualities (a latent space). It chooses a random combination of these qualities by sampling from the prior distribution and decoding it back to MIDI to produce a new musical phrase.

Since the style and quality of the outputs from this plug-in can vary widely, human curation plays an important role in its use. Users can also provide some control by adjusting the softmax temperature, which effectively determines how random the outputs will be.

Continue

Continue is a compositional tool that can be used to generate notes that are likely to follow an input drum beat or melody. The user provides an input clip, and the plug-in will extend it by up to 32 measures (see Figure 3). This plug-in can help a user add variation to a drum beat or create new material for a melodic track. It typically picks up on things like durations, key signatures, and timing. Users can control how close the continuation will stick to the input pattern by adjusting the temperature used during sampling.

This plug-in is similar to the Continuator (Pachet, 2003), but instead of Markov chains, it uses MusicRNNs to do next-step prediction.

Interpolate

Interpolate is a compositional tool that takes two drum beats or two melodies as inputs. It then generates up to 16 clips which combine the qualities of the original two clips. It's useful for merging musical ideas or creating a smooth morphing between them.

Like Generate, Interpolate uses a MusicVAE. One way to think of the VAE is as a mapping from MIDI to a compressed space in which similar musical patterns are clustered together. Each of the input patterns is represented by a point in the compressed space. Interpolate draws a line between these points and decodes clips at evenly-spaced intervals along it. The number of returned clips is set by the "steps" slider. Because of the use of this compressed space, the resulting morphing is over the characteristics of the endpoints, and not necessarily their raw notes.

Groove

Groove is an interpretive tool that adjusts the timing and velocity of an input drum clip to produce the "feel" of a drummer's performance. This is similar to what a "humanize" plug-in does but is achieved in a totally different way.

We recorded 15 hours of real drummers performing on MIDI drum kits as part of the Groove MIDI Dataset (Gillick et al., 2019). These recordings were quantized, removing all velocity and microtiming, and were used to train a GrooVAE with the original performances as the output to learn a mapping between quantized beats and human-performed grooves. When the user provides a quantized beat, we have the GrooVAE apply its learned transformation to map their beat into a groove that feels as if it were performed by a human.

Drumify

Drumify creates drum grooves based on arbitrary input sequences. It can be used in multiple ways:

- as an accompaniment tool that generates a drum track based on the rhythm of another instrument, or
- as an interpretive tool that creates a full drum track from a monophonic rhythm.

Similar to the Groove plug-in, Drumify uses the Groove MIDI Dataset as its training data. However, in this case we first squashed the drum grooves down to a single rhythm instrument (a "tap") with constant velocity and trained a GrooVAE to learn the mapping from this 1-dimensional signal back to the full grooves. When the user supplies an input sequence (be it a drum pattern, a bass line, or a melody), we squash it to a rhythm in the same manner and have the GrooVAE map it to a new groove.

Evaluation

With our initial public release of Magenta Studio for macOS and Windows, we asked early adopters to fill out a short survey. Of the approximately 2,500 users who downloaded the plug-ins in the first week, 89 responded to this request.

Results

In the first section of the survey, we learned about our of users by asking them to select from a list of descriptions those that matched their background. 71% of the respondents are musicians or producers (31% professional) and 49% are machine learning enthusiasts. 22% label themselves as machine learning researchers and 20% music researchers, with 9% selecting both of these labels.

In the second section of our survey, we sought to determine the ease of installation and usability of the Magenta Studio plug-ins. On a Likert scale, we asked respondents to judge their experience of installation (76% easy, 16% neutral, 8% difficult), using Ableton clips as input (74% easy, 15% neutral, 11% difficult), and using Magenta Studio outputs in their work (66% easy, 24% neutral, 10% difficult). Since the the models powering the plug-ins are fairly large, we also asked about the observed response time in producing outputs. 58% of respondents said they experienced little to no delay and only 6% reported significant delays.

In the third section, we wanted to discover how useful the respondents found the plug-ins to be as part of their creative workflow. While users did not overwhelmingly find it easy to achieve their "desired" musical effect with the plug-ins (41% easy, 40% neutral, 19% difficult), a slight majority found them to perform better than they expected (51% better, 38% neutral, 11% worse), a clear majority of users reported the plug-ins helped them feel more creative (72% more, 20% neutral, 8% less), and 93% said it made them feel more productive in their creative process.

Finally, we allowed users to provide open-ended feedback about Magenta Studio. We report selected responses in the following section.

Discussion

While we cannot draw any firm, scientific conclusions from this self-selected and informal study, the results do provide some evidence for the utility of this project.

First, we note that the responses suggest we reaching our target audience of musicians and producers, with the majority of respondents describing themselves as such (71%).

Second, most respondents found the plug-ins easy to install (76%) and easy to use with Ableton (74% for selecting inputs, 66% for using outputs). Furthermore, very few reported significant latency (6%), confirming that the WebGL acceleration provided by TensorFlow.js results in efficient computation across platforms.

Most importantly, respondents found the plug-ins to be a useful addition to their workflows, helping them be more creative (72%) and productive (93%). An interesting finding is that users' opinions were more mixed on how easy it was for them to achieve their desired results. One interpretation of this apparent discrepancy is that users may have unrealistic expectations of the capabilities of our models–perhaps due to our messaging or general misconceptions about machine learning and AI. However, the users overwhelmingly reported that the plug-ins matched or exceeded their expectations (89%) and open-ended responses suggest a more likely explanation is that while the models did not always output exactly what they wanted, they still did something useful. One user who was able to achieve desired results said "sometimes it just did what I had in my mind," while a more common refrain was that the plug-ins helped users to "explore alternate variations with surprisingly unexpected results", "take a simple idea into a direction [they] never would have considered", and "'make seemingly disparate ideas and sequences that [they] would not normally be as experimental with work." This is a very exciting result that shows these tools do not replace or minimize human creativity, but can instead be used to augment and extend it. While one user did not see things this way ("Music should be hard to make in order to weed out the unenthused."), many more praised the plug-ins as useful for helping them to "break out of a rut" with its ability to generate a"seed to then be inspired from" or to "bring together loops and ideas that I would formerly put in my junkyard, and possibly create another song".

It is also useful to look back at our original design principles and see how these results comport with them. The makeup of our user base and the ease with which they were able to install and use Magenta Studio provides evidence that we succeeded with principles $1-3$. One user specifically pointed out the simple interface (principle 3), writing "The layout is simple and straight to the point, which I like." While we did not directly address principle 4 in our survey, many of the open-ended responses supported this idea without any prompting. For example, multiple users echoed this one: "It helped me come up with with new ideas I could explore or modify." Similarly, we did not address principle 5 directly in the survey but our success in this regard is supported by the numerous workflows described by respondents in the open-ended section, such as "I end up in a feedback loop with Continue. I'll play an idea, it will extend that idea, I riff off of the output, feed it back into Continue. It's super fun.” and “Generate gets juices flowing. Interpolation and Continue allow me explore alternate variation.”

Conclusions and Future Work

We introduced Magenta Studio, a suite of co-creative music generation plug-ins for Ableton Live, powered by deep learning models implemented in TensorFlow.js.

While it is too early to fully measure the success of our approach, we are excited about its potential to reach a large audience of creators, and we are pleased with the response of early adopters.

Magenta Studio is open-source and can be used as a template for future plug-ins that also wish to bridge the gap between Ableton Live and TensorFlow. We have demonstrated its use with Magenta.js, an open-source library that is meant to be a communal repository of music generation models, and we welcome contributions from the community to help it grow. Furthermore, we believe it will be useful to integrate these tools with existing MuMe frameworks such as MuseBots (Eigenfeldt, Bown, and Carey, 2015) to increase interoperability of various developments in the field and help them reach end users.

Our hope is that others will build upon this work to bring their research systems to a professional audience earlier in the process of development, allowing them to gather useful feedback and training data from their target users.

We also see value in using this framework to create personalized models for individual users in real-time with local training by TensorFlow.js, for example applying techniques such as latent constraints (Engel, Hoffman, and Roberts, 2018). This could further empower creators to customize their own models to do things researchers may not have intended or imagined – a truly meta-metacreative experience!

Acknowledgements

The authors wish to thank Nikhil Thorat and Daniel Smilkov for their enormous help with all things related to TensorFlow.js. Thanks to Colin Raffel for his thoughtful review of this paper. Thanks to the valuable feedback from early adopters and user study participants. Magenta Studio is based on work by members of the Google Brain team’s Magenta project along with contributors to the Magenta and Magenta.js libraries.

References

Abadi, M.; Agarwal, A.; Barham, P.; Brevdo, E.; Chen, Z.; Citro, C.; Corrado, G. S.; Davis, A.; Dean, J.; Devin, M.; Ghemawat, S.; Goodfellow, I.; Harp, A.; Irving, G.; Isard, M.; Jia, Y.; Jozefowicz, R.; Kaiser, L.; Kudlur, M.; Levenberg, J.; Mané, D.; Monga, R.; Moore, S.; Murray, D.; Olah, C.; Schuster, M.; Shlens, J.; Steiner, B.; Sutskever, I.; Talwar, K.; Tucker, P.; Vanhoucke, V.; Vasudevan, V.; Viégas, F.; Vinyals, O.; Warden, P.; Wattenberg, M.; Wicke, M.; Yu, Y.; and Zheng, X. 2015. TensorFlow: Large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Briot, J.; Hadjeres, G.; and Pachet, F. 2017. Deep learning techniques for music generation - A survey. *arXiv Preprint*.

Brooks, F. P.; Hopkins, A. L.; Neumann, P. G.; and Wright, W. V. 1957. An experiment in musical composition. *IRE Transactions on Electronic Computers* EC-6(3):175–182.

Carter, S., and Nielsen, M. 2017. Using artificial intelligence to augment human intelligence. *Distill*. `https://distill.pub/2017/aia`.

Cope, D. 1996. *Experiments in Musical Intelligence*. Madison, WI: A-R Editions.

Eck, D., and Schmidhuber, J. 2002. Finding temporal structure in music: blues improvisation with LSTM recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, 747–756.

Eigenfeldt, A.; Bown, O.; and Carey, B. 2015. Collaborative composition with creative systems: Reflections on the first musebot ensemble. In *International Symposium on Electronic Art (ISEA)*, 214–219.

Engel, J.; Hoffman, M.; and Roberts, A. 2018. Latent constraints: Learning to generate conditionally from unconditional generative models. In *International Conference on Learning Representations (ICLR)*.

Gillick, J.; Roberts, A.; Engel, J.; Eck, D.; and Bammam, D. 2019. Learning to groove with inverse sequence transformations. In *Proceedings of the 36th International Conference on Machine Learning (ICML)*.

Hadjeres, G.; Pachet, F.; and Nielsen, F. 2017. DeepBach: a steerable model for Bach chorales generation. In Precup, D., and Teh, Y. W., eds., *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, 1362–1371. International Convention Centre, Sydney, Australia: PMLR.

Huang, C.-Z. A.; Vaswani, A.; Uszkoreit, J.; Simon, I.; Hawthorne, C.; Shazeer, N.; Dai, A. M.; Hoffman, M. D.; Dinculescu, M.; and Eck, D. 2019. Music transformer. In *International Conference on Learning Representations*.

Kayacik, C.; Chen, S.; Nørly, S.; Holbrook, J. S.; Roberts, A.; and Eck, D. 2019. Identifying the intersections: User experience + research scientist collaboration in a generative machine learning interface. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CS09:1–CS09:8.

Mozer, M. C. 1994. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science* 6(2-3):247–280.

Pachet, F. 2003. The continuator: Musical interaction with style. *Journal of New Music Research* 32(3):333–341.

Papadopoulos, A.; Roy, P.; and Pachet, F. 2016. Assisted lead sheet composition using FlowComposer. In *International Conference on Principles and Practice of Constraint Programming*, volume 9892, 769–785.

Pasquier, P.; Eigenfeldt, A.; Bown, O.; and Dubnov, S. 2016. An introduction to musical metacreation. *Computers in Entertainment* 14:2:1–2:14.

Paszke, A.; Gross, S.; Chintala, S.; Chanan, G.; Yang, E.; DeVito, Z.; Lin, Z.; Desmaison, A.; Antiga, L.; and Lerer, A. 2017. Automatic differentiation in PyTorch. In *Autodiff Workshop, NIPS*.

Roberts, A.; Engel, J.; Raffel, C.; Hawthorne, C.; and Eck, D. 2018. A hierarchical latent vector model for learning long-term structure in music. In *Proceedings of the 35th International Conference on Machine Learning (ICML)*, volume 80, 4364–4373.

Roberts, A.; Hawthorne, C.; and Simon, I. 2018. Magenta.js: A JavaScript API for augmenting creativity with deep learning. In *Joint Workshop on Machine Learning for Music (ICML)*.

Simon, I. 2016. Improv rnn. `https://goo.gl/magenta/improvrnn`. Accessed: 2019-5-20.

Smilkov, D.; Thorat, N.; Assogba, Y.; Yuan, A.; Kreeger, N.; Yu, P.; Zhang, K.; Cai, S.; Nielsen, E.; Soergel, D.; Bileschi, S.; Terry, M.; Nicholson, C.; Gupta, S. N.; Sirajuddin, S.; Sculley, D.; Monga, R.; Corrado, G.; Viégas, F. B.; and Wattenberg, M. 2019. TensorFlow.js: Machine learning for the web and beyond. *arxiv Preprint* abs/1901.05350.

Sullivan, C. 2012. Jnana. `https://ccrma.stanford.edu/~colinsul/projects/jnana/`. Accessed: 2019-1-15.

Todd, P. M. 1991. A connectionist approach to algorithmic composition. In Todd, P. M., and Loy, D. G., eds., *Music and connectionism*. Cambridge, MA: MIT Press/Bradford Books. 173–194.

Waite, E. 2017. Generating long-term structure in songs and stories. `https://magenta.tensorflow.org/2016/07/15/lookback-rnn-attention-rnn`. Accessed: 2019-5-20.

Zicarelli, D. 1987. M and Jam Factory. *Computer Music Journal* 11(4):13–29.