Magenta.js: A JavaScript API for Augmenting Creativity with Deep Learning

Adam Roberts [1] **Curtis Hawthorne** [1] **Ian Simon** [1]

Abstract

New methods for modeling musical scores, often based on deep learning, have made it possible to automatically generate ever more convincing compositions. However, these models are often not accessible enough for artists to easily play and experiment with on their own terms. Application developers are already imagining ways to apply their design expertise to this new class of AI technologies but often lack a sufficient background in machine learning. Magenta.js is a new open source library with a simple JavaScript API intended to bridge this gap by abstracting away technical details, making it easier than ever for app developers to create new interfaces to generative models. Furthermore, because it is easily extensible, we hope that Magenta.js can foster a connection between the broader research community and creative developers through contributions from both groups. Finally, Magenta.js will open up the possibility for a new type of compositional tool that adapts to user preferences and behaviors in real-time. Code and documentation are available online at `https://goo.gl/magenta/js`.

1. Introduction

Deep learning has proven to be a powerful technique for modeling complex data distributions, including music and art. While there is some desire to use these models for autonomous generation, a more transformative approach involves using these models to augment–rather than replace–human creativity to develop new processes, new art forms, and new types of artists (Carter & Nielsen, 2017).

In order to invent the next generation of smart instruments and compositional tools, interface developers and musicians must be able to experiment with generative models in an environment they are comfortable with. While there are notable examples of simple tools for creating instrument interfaces with machine learning (Fiebrink & Cook, 2010), using deep learning for generation has thus far relied on frameworks targeted at researchers and backend developers with a high level of machine learning expertise.

The recent advent of WebGL-accelerated frameworks, including TensorFlow.js[1], makes it possible to do efficient inference *and* training within a desktop or mobile web browser using common frontend languages like JavaScript. The web provides a flexible, universal portal with an immense base of existing developers and users. WebAudio and WebMIDI supply an interface to professional music production tools, making the web an ideal testbed for developing smart musical applications that are accessible to creators of all levels. A high-level API that implements music generation models while hiding unnecessary complexities from developers would remove the need for machine learning expertise, creating an environment that is rife with opportunity to revolutionize music creation.

The purpose of Magenta.js is to to provide this higher-level API for creative developers, utilizing TensorFlow.js to be fast and extensible. While the goal of Magenta.js is broader than just music, the first package of the suite, `@magenta/music`, implements several state-of-the-art models for music generation. We hope it will grow to include many more as researchers adopt it as a conduit for engaging developers and end users.

2. API

The `@magenta/music` package contains a JavaScript API for interacting with music generation models. Here we discuss three major components of the API: data processing, model interfaces, and accessing pre-trained models.

2.1. Data Processing

To provide a universal interface, a shared representation of musical scores is required. We chose to use the `NoteSequence` protocol buffer originally developed for the Magenta Python library[2]. This data representation stores fundamental aspects of note sequences (timing, pitches, in-

[1]Google Brain, Mountain View, CA, USA. Correspondence to: Adam Roberts <adarob@google.com>.

Proceedings of the 35^{th} International Conference on Machine Learning, Stockholm, Sweden, PMLR 80, 2018.

[1]`https://js.tensorflow.org`

[2]`https://goo.gl/magenta/py`

struments, etc.) as well additional metadata (section labels, chord information, etc.). It is serializable and can be accessed and modified via multiple languages.

The package also contains a TypeScript re-implementation of a subset of the Magenta Python `NoteSequence` library, including functionality for converting to and from MIDI and synthesizing audio for playback. There is also a set of `DataConverter` classes to convert between `NoteSequences` and the tensors that are used as input and output to the neural networks. These converters mirror those used in the Python library, enabling inference on models trained with TensorFlow.

2.2. Model Interfaces

We are initially supplying two model classes: `MusicRNN` and `MusicVAE`.

2.2.1. MusicRNN

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

The `MusicRNN` class implements a combination of ideas from this body of work, including models for melodies, drum patterns, and polyphonic human piano performances. A typical interaction for this type of model is to extend a priming sequence, similar to "auto-complete" in text.

In our interface, this interaction is achieved by calling `continueSequence` and providing a priming `NoteSequence`, which may be empty for pure sampling. Additionally, the method accepts an optional chord sequence for models that support this type of conditioning.

2.2.2. MusicVAE

The `MusicVAE` class implements several variants of the more recent hierarchical recurrent variational autoencoder described in (Roberts et al., 2018), including models for melodies, drum patterns, and multi-instrument sequences.

Autoencoders support a different set of interactions than language models. The implemented methods include `encode` and `decode`, to encode a `NoteSequence` into a latent vector and decode a latent vector into a `NoteSequence`, respectively. These two methods are sufficient to enable a rich set of latent space operations including sampling, interpolation, and attribute vector arithmetic. We provide `interpolate` and `sample` methods for simplicity.

As with `MusicRNN`, these methods accept optional arguments for chord conditioning when supported by the model.

2.3. Loading Models

A key to making it trivial for all developers to use machine learning in their applications is to provide pre-trained models. Our approach is to host weights along with configurations in JSON format. The configuration contains all parameters of a model that cannot be inferred from names and shapes of the weights. A JSON-formatted list of hosted models along with their descriptions and URLs is maintained at `https://goo.gl/magenta/js-checkpoints`.

To use a particular model, a developer simply instantiates the correct model class with a URL for its configuration and weights (see Figure 1, line 6).

So that others may host their own trained models, we also provide a script for extracting weights from TensorFlow checkpoints along with specifications for the JSON-formatted configuration file.

3. Examples

Figure 1 implements a simple web application built with Magenta.js. When a user clicks the "Play Trio" button, the model is loaded and a sample is played back. A number of more interesting apps have already been made by others[3].

```
1  <html>
2    <head>
3      <script src="https://cdn.jsdelivr.net/npm/@magenta/
           music@1.0.0"></script>
4      <script>
5        // Instantiate model by loading desired config.
6        const model = new mm.MusicVAE(
7          'https://storage.googleapis.com/magentadata/js/
               checkpoints/music_vae/trio_4bar');
8        const player = new mm.Player();
9
10       function play() {
11         mm.Player.tone.context.resume();  // enable audio
12         model.sample(1)
13           .then((samples) => player.start(samples[0]));
14       }
15     </script>
16   </head>
17   <body><button onclick="play()">Play Trio</button></body>
18 </html>
```

Figure 1. A minimal web application that samples and plays back a 4-bar "trio" from a MusicVAE model. This application is hosted at `https://goo.gl/magenta/simpletrio`.

4. Conclusion

With Magenta.js, we provide an API to help bridge the gap between researchers, developers, and creators. We hope our open source approach will lead to contributions from all three communities as new models and creative tools are developed. Magenta.js also lays the groundwork for the in-browser refinement and personalization of models based on user preferences and behavior (Jaques et al., 2018), which we plan to explore in future work.

[3] `https://g.co/magenta/demos`

References

Briot, Jean-Pierre, Hadjeres, Gaëtan, and Pachet, François. Deep learning techniques for music generation - A survey. *arXiv Preprint*, 2017. URL http://arxiv.org/abs/1709.01620.

Carter, Shan and Nielsen, Michael. Using Artificial Intelligence to Augment Human Intelligence. *Distill*, 2017. URL http://distill.pub/2017/aia.

Eck, Douglas and Schmidhuber, Jürgen. Finding temporal structure in music: blues improvisation with lstm recurrent networks. In *Proceedings of the 12th IEEE Workshop on Neural Networks for Signal Processing*, pp. 747–756, Sept 2002. doi: 10.1109/NNSP.2002.1030094.

Fiebrink, Rebecca and Cook, Perry R. The wekinator: a system for real-time, interactive machine learning in music. In *Proceedings of The Eleventh International Society for Music Information Retrieval Conference (ISMIR 2010)(Utrecht)*, 2010.

Jaques, Natasha, Engel, Jesse, Ha, David, Bertsch, Fred, Picard, Rosalind, and Eck, Douglas. Learning via social awareness: improving sketch representations with facial feedback. In *International Conference on Learning Representations*, 2018. URL https://arxiv.org/abs/1802.04877. Workshop Track.

Mozer, Michael C. Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multi-scale processing. *Connection Science*, 6(2-3):247–280, 1994. URL https://doi.org/10.1080/09540099408915726.

Roberts, Adam, Engel, Jesse, Raffel, Colin, Hawthorne, Curtis, and Eck, Doug. A hierarchical latent vector model for learning long-term structure in music. *arXiv preprint arXiv:1803.05428*, 2018.

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