RoboJam: A Musical Mixture Density Network for Collaborative Touchscreen Interaction

Charles P. Martin* and Jim Torresen†

November 22, 2017

Abstract

RoboJam is a machine-learning system for generating music that assists users of a touchscreen music app by performing responses to their short improvisations. This system uses a recurrent artificial neural network to generate sequences of touchscreen interactions and absolute timings, rather than high-level musical notes. To accomplish this, RoboJam's network uses a mixture density layer to predict appropriate touch interaction locations in space and time. In this paper, we describe the design and implementation of RoboJam's network and how it has been integrated into a touchscreen music app. A preliminary evaluation analyses the system in terms of training, musical generation and user interaction.

Keywords: New Interfaces for Musical Expression (NIME); Artificial Neural Networks; Musical Artificial Intelligence; Mobile Human-Computer Interaction; Collaboration.

1 Introduction

New interfaces for musical expression (NIMEs) are often aimed at casual or novice musicians, this is especially the case for touchscreen instruments that can be deployed on popular mobile devices. While these interfaces often emphasise a solo musical production paradigm, the concept of ensemble, or collaborative, performance has often been under-explored. That mobile devices are typically used while alone, in transit, or in public, significantly limits opportunities for touchscreen musicians to jam with others. This limitation restricts the user's ability to gain feedback, respond to others' ideas, and to refine their own music-making through collaboration. In this research, we propose a machine-learning system for generating music that can be used to assist the user by composing responses in near real-time, thus emulating the experience of collaborating with another user.

Our system, called *RoboJam*, has been integrated into an existing touchscreen music app. In this app, the user is able to compose short musical pieces using the touchscreen, share them with friends, or collaborate by "replying" to other performances. The user taps, swipes and swirls in a free-form manner on the touchscreen and these interactions are translated into musical sounds.

*Department of Informatics, University of Oslo
†Department of Informatics, University of Oslo

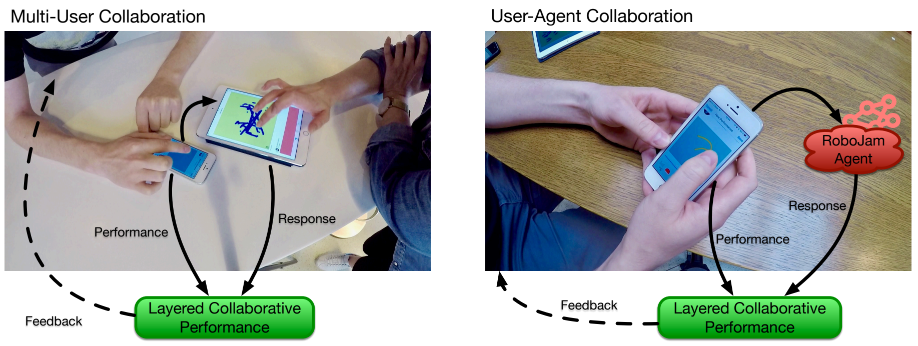


Figure 1: RoboJam takes the role of a remote collaborator, providing musical responses to a user's touchscreen improvisations on demand. In this way, the user can gain feedback from a simulated collaboration, while waiting for other users of to respond.

Several sound schemes are possible, with the touches mapped to different kinds of synth and instrumental sounds. Recordings, which are limited to 5 seconds in length, are uploaded automatically to a cloud server where they can be played back by other users. Users can collaborate by replying to each others' performances, forming more complex pieces where replies are composed as sonic layers with the original performance. In this way, the collaborative interaction mirrors a call-and-response style of performance; the first performer plays the call, and subsequent performers reply with responses.

This process of musical collaboration can potentially offer important feedback to the original performer. Hearing their own performances layered with other responses might change how they approach further improvisations and they might gain new ideas from others' responses. In a co-located situation, as shown in the left hand side of Figure 1, this feedback loop could be tight and allow rapid improvement and increased engagement with the touchscreen interface. RoboJam is an agent designed to emulate this call-and-response interaction in situation when other users aren't present or available, this process is shown in the right side of Figure 1. The user can call on RoboJam to provide a response to their performances whenever–and however many times–it is required. RoboJam is designed to predict what a touchscreen performer might do next, having heard the first performance. The advantage of this design is that the user can hear their own performance with multiple accompanying replies, thus the user can practice and refine their own performance ideas in context.

The machine-learning system behind RoboJam uses an artificial neural network (ANN) to generate responses on demand. Similar to other music generation systems, RoboJam uses a recurrent neural network (RNN) with long short-term memory (LSTM) cells to predict a temporal sequence of discrete events. In this sequence-predicting configuration, the input to the network is the current event, and the output is the next predicted event; thus, the network can predict a sequence one event at a time.

When used to predict responses, our RoboJam network first "listens" to the user's performance, i.e., the performance is propagated through the network (ignoring predictions) to condition the LSTM memory state. A new performance, 5 seconds in length, is then predicted by the network to form the response. A key point of difference for RoboJam is that it models the input data, a stream of touch-screen events, rather than the musical data, frequently MIDI-note pitches, modelled by other ANNs for generating music. This is made possible by the novel application of a mixture density network (MDN) to creative touchscreen interactions in RoboJam. In this research we show how this system models musical control data, and discuss evaluations from the perspectives of model validation, performance generation and user experience. The results support RoboJam's ability to generate responses that are related to the call performance and improve their sound as well as to enhance the human performer's experience.

The structure of this paper is as follows: In Section 2 we will discuss related work in the use of ANNs to generate music and in co-creative musical interfaces. In Section 3 we will discuss the neural network design and training for RoboJam. Section 4 will describe how this network is integrated into the touchscreen app and its interaction design. Evaluations will be discussed in Section 5.

2 Related Work

2.1 RNN Music Generation

ANNs have long been used to model and generate music. RNNs, able to learn temporal information in between computations, are particularly applicable to modelling musical sequences, where upcoming events strongly depend on those that have previously occurred. In general, these networks generate musical event sequences in a one-by-one manner; the input is the present note, and the output predicts the next note in the sequence. Mozer's CONCERT system [17] was an early attempt to compose music using an RNN; this work emphasised the advantage in learning long range dependencies in music without handling extremely large transition tables in, for example, a Markov model. Eck and Schmidhuber contributed further work in this area, using an RNN to generate (potentially) endless blues music [5]. This project notably used long short-term memory (LSTM) units [9] to help alleviate the problem of vanishing or exploding gradients when training RNNs.

In recent times, the use of GPU computation and large datasets have enabled a variety of creative applications for RNNs including the popular Char-RNN model [12] as well as in music generation. Sturm et al. focused on a textual representation of music in their FolkRNN project [19]. This system was trained on a large dataset of folk melodies available online in the plain-text "ABC" format. Hadjeres et al. created an RNN generator of J.S. Bach-styled chorales that can be steered towards particular melodies [8]. Colombo et al. used an RNN to predict pitch and rhythm separately [4]. Malik and Ek used an RNN trained on MIDI-recordings of piano performances to augment existing MIDI compositions with dynamic (volume) instructions [13]. Hutchings and McCormack used an RNN to generate harmonic sequences in an agent-based jazz improvisation system [10]. Google's Magenta project[1] have released various trained RNN models for music generation and made steps towards integrating them with popular music production software.

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

2.2 Mixture Density Networks

A commonality of the above models is that they predict sequences of events from a finite set of symbols (e.g., one of the 128 integer MIDI pitches). Mapping the output of an RNN to a prediction from a finite number of classes can be effectively managed using the softmax function. In our application, we wish to predict locations and timings of interactions with a touchscreen, which have real—not finite–values, so a softmax output cannot be used without a significant loss of precision. A similar task, generating simple line drawings, was recently tackled by the Magenta group resulting in SketchRNN [7]. This system is able to generate drawings which, unlike pixel representations of images, are constructed from sequences of pen movements in a 2D plane. The approach used here, and in previous experiments in generating handwriting [6] was to replace the categorical softmax model at the outputs of the network with a mixture model, which provide more flexible predictions.

Mixture density networks (MDNs), where the output of a neural network is used as the parameters of a mixture model, was first explored by Bishop as a way of learning to predict multimodal problems that do not fit a normal distribution and are poorly modelled by a least-squares approach to training the output of an ANN [2]. The probability density function (PDF) of a mixture of normal distributions can be calculated by taking the linear combination of a number of normal distributions with weighting coefficients. Bishop observed that an ANN could be trained to produce these coefficients (π_i), and the centres (μ_i) and variances (σ_i) for each component of the mixture. Thus a loss function for the network could be given by the negative logarithm of the following likelihood function, given m mixtures, of target $\mathbf{t}$ occurring in the distribution generated by input data $\mathbf{x}$:

$$\mathcal{L} = \sum_{i=1}^{m} \pi_i(\mathbf{x}) \mathcal{N}\big(\mu_i(\mathbf{x}), \sigma_i^2(\mathbf{x}); \mathbf{t}\big) \tag{1}$$

After training, the mixture model can then be sampled to generate output data. While Bishop used an MDN with a non-recurrent network, it has also been applied with RNNs as in Graves' experiments [6], and SketchRNN [7] mentioned above. Such a configuration, where mixture density layers follow an RNN, could be termed an MDRNN. While MDNs have advantages, they also suffer from difficulties in training [3], and are not widely used. To our knowledge, an MDN design has not previously been applied to a musical task.

2.3 Predictive Musical Interfaces

There are many examples of generative music systems that are designed to collaborate with human performers. These are often used in a soloist-ensemble configuration where a human performer is "accompanied" by artificially generated musicians. GenJam [1] is a generative backing band that knows how to play jazz standards. The Reflexive Looper [15] can arrange short recordings of the performer's own sounds into parts fitting a given song structure.

Systems that *learn* some aspect of the user's style and can interact with them in performance are more relevant to this work. The *Continuator* [18] did this very effectively using Markov models, and defined an interaction model of continuing to play when the user stops. More recently, it has become possible to apply RNN music models in a real-time system as shown in Magenta AI Duet [14]. In this instance, the model does not learn from the user in the sense of updating ANN weights; rather, the user's performance can be used to condition the LSTM cells' memory state which then governs the style of generated notes. Due to this conditioning and generation process, AI Duet can be directed towards different styles after having learned a more general model of music.

3 Neural Network Design

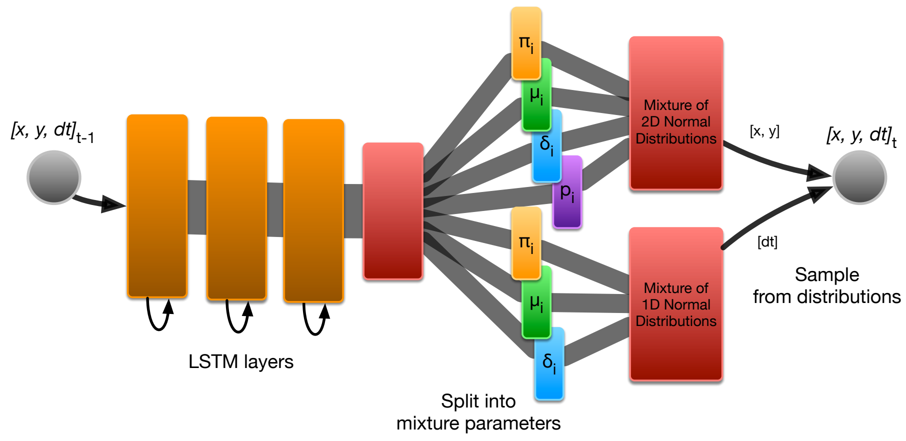
RoboJam's artificial neural network is a model of musical touchscreen interactions. These interactions consist of touch points in a 2D plane as well as the location of these interactions in time. In our touchscreen music app, the user interacts with an area of the touchscreen in a free-form manner to perform music, and is not constrained by UI elements such as virtual faders or buttons. This means that touchscreen interaction data can be considered as a record of the user's control gestures: interactions with the touchscreen that produce a musical result [11]. Thus, by modelling and generating this control data, RoboJam can produce new performances.

The focus on control gesture data and temporal locations means that this model differs from most music generation RNNs. FolkRNN [19] (among others) predicts symbols from a finite dictionary defined before training. This means that the output of the network can be generated by sampling from a categorical distribution given by a softmax layer. As our network must predict real-valued locations in 2D space, a softmax layer cannot be used without quantising the output to an unacceptably low resolution. Other musical generation systems such as DeepBach [8] generate notes on a predefined semiquaver pulse. Our network predicts temporal locations for each touch interaction, again as a positive, real-valued number of seconds. Generating real-valued control data provides precision, but has a cost in terms of the amount of high-level information (e.g., harmony) that can be learned. This approach would not be appropriate in all musical systems, but allows RoboJam to focus on free-form touch expression and means that it can be used with many different musical mappings, and even in different creative apps.

3.1 Dataset

Training data for the network consisted of musical touchscreen improvisations. This data was derived from a corpus of performances by touchscreen ensembles [16]. The dataset included 163 collaborative sessions corresponding to 20 hours of performance and 4.3M individual touch interaction events–the musical control data for these performances. It should be noted that this dataset was collected in a variety of apps. Each of these apps mapped free-form touchscreen interactions directly to sound so this corpus is an appropriate analogue for the host app for RoboJam.

The dataset was prepared first by extracting each performer's contribution from each session. The touch locations of each record were transformed from the original recording resolution to be in $[0, 1] \times [0, 1]$. Absolute times were changed to time deltas with a maximum value of 5 seconds. This means that



$[x, y, dt]_{t-1}$
LSTM layers
π_i
μ_i
δ_i
p_i
π_i
μ_i
δ_i
Split into mixture parameters
Mixture of 2D Normal Distributions
Mixture of 1D Normal Distributions
[x, y]
[dt]
$[x, y, dt]_t$
Sample from distributions

Figure 2: RoboJam uses the output of an RNN to parameterise mixture models that handle touchscreen location and the timing of events. Two models are used: a mixture of 2D normal distributions to predict location of touches, and a mixture of 1D normals to predict time deltas in between events.

each touch event became a vector $(x, y, dt) \in [0, 1] \times [0, 1] \times [0, 5]$. For training, the performances were placed end-to-end to form one long sequence of touch events. Overlapping examples of 256 events in length were extracted from this sequence resulting in almost 4.3M training examples.

3.2 Implementation

To model this data, we use an MDRNN design, inspired by SketchRNN [7] and Graves' handwriting generation network [6] (discussed in Section 2.2). Both of these systems are intended to generate sequences of 2D surface interactions, very similar to our touchscreen data; however, sketches and handwriting do not have a significant temporal component except for the ordering of strokes. Both of these previous ANNs used the outputs of LSTM cells as the parameters of a mixture of probability models. In both cases, a mixture of bi-dimensional normal distributions were used to predict the location of the next pen movement in 2D space, the number of normal distributions in a mixture was a hyperparameter in training these networks. In our case, location of touches is accompanied by a positive time value, indicating the number of seconds in the future that the interaction should occur. The same 2D mixture distribution as in SketchRNN is used to predict spatial location of touches while a separate mixture of 1D normal distributions is used to predict the temporal location of the touch.

Our MDRNN is implemented in TensorFlow, and the source code is available online[2]. An overview of our MDRNN design is shown in Figure 2. Input data are vectors $[x, y, dt] \in \mathbb{R}^3$ corresponding to absolute location in the plane $[0, 1] \times [0, 1]$ and value in $[0, 5]$ indicating the number of seconds since the last interaction. The input vector flows through three fully-connected layers of LSTM cells. The output of the last LSTM layer is projected using a fully connected layer to a

[2]RoboJam's source code is available at `https://doi.org/10.5281/zenodo.1064014`

vector p that is used to generate parameters for our two mixture models that predict time and space values respectively.

The time model is a mixture of M 1D normal distributions, parameterised as given in Section 2.2 by a mean (μ_t), standard deviation (σ_t), and a coefficient π_t for each mixture component. The space model is a mixture of M 2D normal distributions, each having two means (μ_x, μ_y), standard deviations (σ_x, σ_y), one correlation (ρ) and coefficient (π_{xy}). So, p has size $3M + 6M$.

The components of p are transformed as recommended by Brando [3], Graves [6] and Bishop [2]. This ensures that the standard deviations are greater than zero, correlations are in $(-1, 1)$, and that the coefficients for each mixture sum to one. In operation, a sample is drawn from these two mixture models to generate the next vector: $[x_{t+1}, y_{t+1}, dt_{t+1}]$. In training, we measure the likelihood of the next (known) vector occurring in the generated mixture models. The loss function has two components that are added — the average negative log likelihood over the batch for the time model and for the space model:

$$\mathcal{L}_{space} = -\frac{1}{N}\sum_{i=1}^{N}\log\sum_{j=1}^{M}\big(\pi_{xy,j}\mathcal{N}_{2D}(\mu_{x,j}, \mu_{y,j}, \sigma_{x,j}, \sigma_{y,j}, \rho_j; x_i, y_i)\big) \tag{2}$$

$$\mathcal{L}_{time} = -\frac{1}{N}\sum_{i=1}^{N}\log\sum_{j=1}^{M}\pi_{dt,j}\mathcal{N}_{1D}(\mu_{dt,j}, \sigma_{dt,j}; dt_i) \tag{3}$$

$$\mathcal{L}_{tot} = \mathcal{L}_{space} + \mathcal{L}_{time} \tag{4}$$

Where $\mathcal{N}_{2D}$ is as given in equations 24 and 25 of [6], and $\mathcal{N}_{1D}$ is as given in equation 23 of [2]. Code for $\mathcal{N}_{2D}$ follows Ha and Eck's work [7] and a similar TensorFlow implementation was developed for $\mathcal{N}_{1D}$.

3.3 Training

As discussed above in Section 3.1, the training dataset consisted of 4.3M overlapping examples of 256-event performance excerpts. These examples were shuffled before training and divided into 33579 batches of 128 examples. The Adam optimiser was used for gradient descent with an initial learning rate of 1×10^{-4}. RoboJam's network was configured with 3 layers of LSTM units and $M = 16$ mixtures. Models were trained up to five epochs with LSTM layer sizes of 64, 128, 256, and 512 units. The results of this training is discussed below in Section 5.1.

Numerical stability in training is a significant challenge for MDNs. We follow advice from Bishop [2], Brando [3] and others to transform and clip the mixture parameters, and to apply gradient clipping. Nevertheless, avoiding division by zero when calculating $\mathcal{L}_{tot}$ continued to be a challenge in this research and is a topic for future investigation.

3.4 Model Limitations

Our implementation separates space and time predictions into two mixture models, both connected to the output of one RNN. It may be possible to represent these in a single mixture of 3D normal distributions, but the PDF for a 3D normal is more complex. Our decision here was pragmatic in that we were able

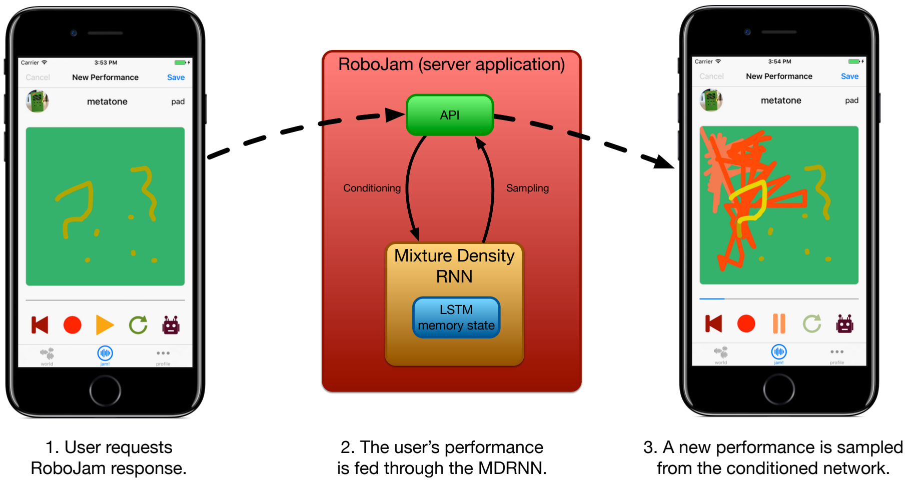

Carrier
3:53 PM
Cancel
New Performance
Save
metatone
pad
RoboJam (server application)
API
Conditioning
Sampling
Mixture Density RNN
LSTM memory state
Carrier
3:54 PM
Cancel
New Performance
Save
metatone
pad
1. User requests RoboJam response.
2. The user's performance is fed through the MDRNN.
3. A new performance is sampled from the conditioned network.

Figure 3: RoboJam is a web-application that generates responses to touchscreen performances. User performances are sent to the server, used to condition the memory state of an MDRNN, this can then generate a new performance that is returned to the user as layer of accompaniment.

to extend previously effective MDN designs with parameters for space and time predictions linked via the fully-connected LSTM and projection layers. Future work could determine whether a 3D model, or indeed three 1D models would be more suitable. Our model does not explicitly consider the state of a touch event (whether it is a new, or moving touch), even though this data is available in the corpus. Given the temporal dimension, this aspect of the data is quite predictable so we used a heuristic threshold of $dt > 0.1$ to determine a touch as new. Potentially this data could be predicted by the MDRNN as in earlier handwriting [6] and sketch [7] models.

4 Interaction Design

RoboJam is a web-application that uses the MDRNN described above to predict appropriate responses to user-created touchscreen performances. The interaction paradigm mirrors a call-and-response improvisation; the performer creates a short improvisation, RoboJam "listens" to this performance and creates a response, then both contributions can be played back at the same time[3].

The system architecture is illustrated in Figure 3: First, the user performs a short piece of music with the touchscreen interface; the app limits these performances to 5 seconds in length. The user can then request a RoboJam response by tapping a button in the app's graphical interface. The performance, consisting of a sequence of touch interaction events, is encoded in JSON format and sent to the server via a REST API. When the server receives a performance, it feeds each touch event in sequence through the MDRNN in order to condition the neural net's memory state. When this is complete, it generates a new performance from the MDRNN by sampling a new sequence of touch-events

[3]This process is illustrated in the video figure: `https://vimeo.com/242251501`

Training Step (1000s)	64 units		256 units		512 units	
	Training	Valid'n	Training	Valid'n	Training	Valid'n
2	-3.72	-4.21	-3.98	-4.39	-3.56	-3.81
34	-6.73	-7.58	-7.30	-8.58	-6.96	-7.92
68	-7.61	-8.21	-8.42	-8.98	-8.86	-9.50
100	-7.67	-8.39	-8.14	-8.85	-9.14	-9.88
132	-7.93	-8.61	-8.19	-9.09	-9.47	-10.00
166	-8.07	-8.76	-8.51	-8.84	-9.42	-10.31

Table 1: Training and validation loss for RoboJam's MDRNN at throughout training and with different RNN-layer sizes. (Lower is better.)

until 5 seconds of interactions have been saved. This sampled performance is sent back to the user's device and displayed as a second layer of performance on the screen. When the user hits "play", both performances are played back simultaneously as separate parts.

In this research, RoboJam has been used with a prototype touchscreen music app for iOS devices. This app allows touch interactions to be interpreted as different instrumental sounds such as drums, strings, bass, or different synthesiser sounds. In general, these mappings allow the user continuous control of pitch on the x-axis and tone or effects on the y-axis. The mappings are available to the user as selectable presets for their performances. Response performances from RoboJam's MDRNN can be performed by any one of these mappings, but our implementation assigns one randomly to each response such that the mapping is different than that currently selected by the user.

RoboJam is implemented in Python using the Flask web development framework. This design allowed RoboJam's TensorFlow-based MDRNN (also defined in Python) to be accessed easily. Future implementations of RoboJam could be integrated directly into a touchscreen app using TensorFlow, or manufacturer-specific deep learning frameworks such as Apple's CoreML. In the short-term, however, the simplicity of using the MDRNN in a client/server architecture outweighed the potential benefits of on-device predictions.

5 Evaluation

Evaluation of RoboJam, as with many co-creative interfaces, presents challenges. We have chosen to split evaluation into three stages: validation, generation, and interaction. These stages respectively address the following questions: Has the networked learned anything? Does the network generate realistic performances? Is the interactive system useful for its purpose of enhancing the users' musical experience?

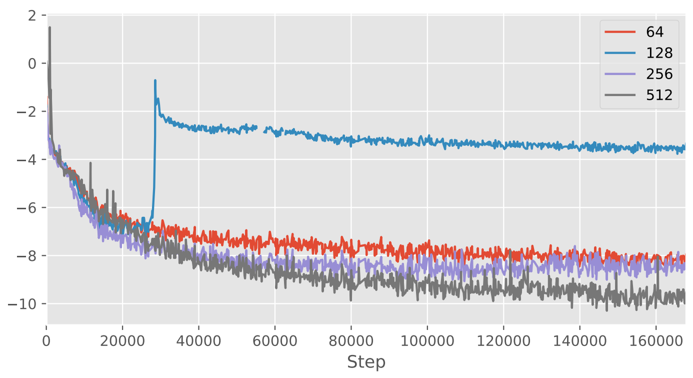


Figure 4: Training loss for RoboJam networks with 64, 128, 256, and 512-unit LSTM layers. Training and validation loss for the 512 unit network was lowest. The 128-unit network failed to train due to numerical errors.

5.1 Validation

Four versions of the model were trained up to five epochs (166000 batches) using 64, 128, 256, and 512-unit LSTM layers respectively. Training loss for these models can be seen in 4; notably, the 128-unit network failed to train due to numerical errors.

Validation was performed by calculating loss values when the network predicts excerpts of touchscreen performances from RoboJam's host touchscreen app. This validation dataset was not used in training and consisted of 1047 performances with a total of 168039 touch interactions; these data were processed similarly to the training set. A validation experiment was performed on the three successfully trained models with training and validation loss recorded at model checkpoints near the end of each epoch. The results in Table 1 show that both training loss and validation loss decrease during training with the lowest scores occurring with the 512-unit network after five epochs. This model was used for the two evaluations below.

It is notable that the loss on the validation set is consistently lower than on the training set – an unusual situation. This suggests that the validation data may actually be more predictable than the training data. Given that the training data was collected from a variety of different apps, it may contain more unpredictable or arbitrary interactions than we thought. While validation shows that the network's predictive ability improved during training, it may be better in future to focus training on a dataset that has been restricted to the most relevant interactions.

5.2 Performance Generation

The output of RoboJam's MDRNN are performances that should be experienced in time; however, as a quick overview, we can view plots of the touches from these performance over their whole duration. Figure 5 shows five-second performances generated without conditioning the network's memory state. More relevant to the application, Figure 6 shows examples conditioned performances, the input

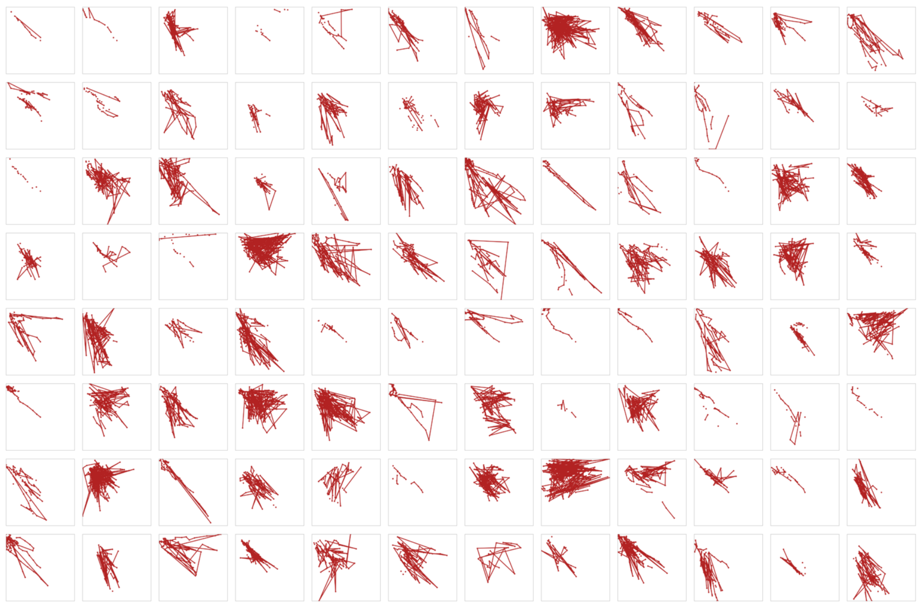


Figure 5: RoboJam's network generating 5-second performances in "unconditional" mode, i.e., with an empty memory state and starting from the centre of the performance area.

performance is shown in blue and RoboJam's response is shown in red. In both figures, moving touches are connected while individual touches are not. Examples of call-and-response RoboJam performances can be found in a video figure for this paper which is available online[4].

Figures 5 and 6 show a variety of touchscreen interaction behaviours, but reveal limitations in terms of generating realistic performances. Compared with example touchscreen performances (shown in blue in Figure 6), the network generated performances do not have as many smooth paths, and have many more large jumps across the screen with very short time delays (resulting in long connected paths). While the unconditional performance (Figure 5) show uncontrolled behaviour, many of the conditioned performances 6 bear some relation to the "style" of the input performance, in terms of location of touch points, direction and shape of motion. Some rhythmic relationships can also be seen, input performances with sparse disconnected paths indicate a rhythmic performance, these seem to lead to more rhythmic variation in the generated response.

Frequently in the generated performances, the touch point tends to move to the upper left corner of the touch area, corresponding to location $(0, 0)$ in the 2D plane. This tendency is as confusing as it is worrying, given that the training data is much more evenly spread across the touch field. Exploring the reasons and possible solutions for this issue is a topic to be addressed in future research.

[4]Video figure: `https://vimeo.com/242251501`

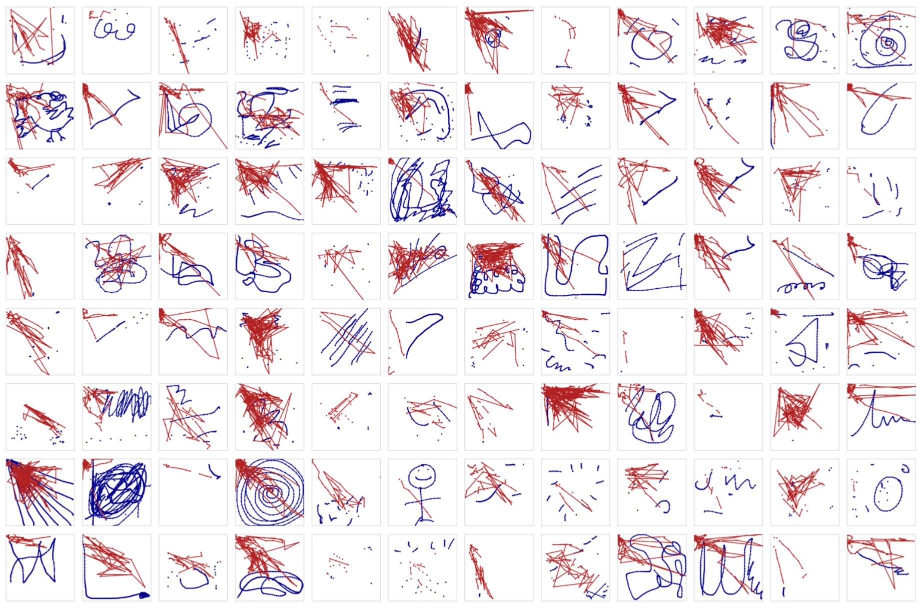


Figure 6: Examples of 5-second responses to performances from the validation set. The input performance is shown in blue and RoboJam's response is shown in red.

5.3 Interaction

A preliminary evaluation of interaction was performed with a small group of users with RoboJam connected to a touchscreen music application. The aim of the study was to gain a rough picture of how users viewed the quality of RoboJam's responses and the experience of interacting with this call-and-response agent. Twelve participants were included in the study: students, researchers, and teachers of computer science and music with a mix of musical experience. The study procedure was as follows: Each participant was given a short tutorial and practice session (around 5 minutes) in creating performances in the app. They were then asked to create several performances and generate as many responses from RoboJam as they liked, listening to each resulting layered performance. After the test session, the participants responded to five Likert-style statements on 5-point agreement scales (*strongly disagree, disagree, neutral, agree, strongly agree*). The questions were:

1. The responses were related to my performance. (*related*)
2. The responses had a high musical quality. (*quality*)
3. The responses showed musical creativity. (*creativity*)
4. The response layer made my performance sound better. (*improvement*)
5. Interacting with RoboJam enhanced my experience. (*experience*)

The results of this study are shown in Figure 7. Almost all the users felt that the responses were related to their performance and that interacting with

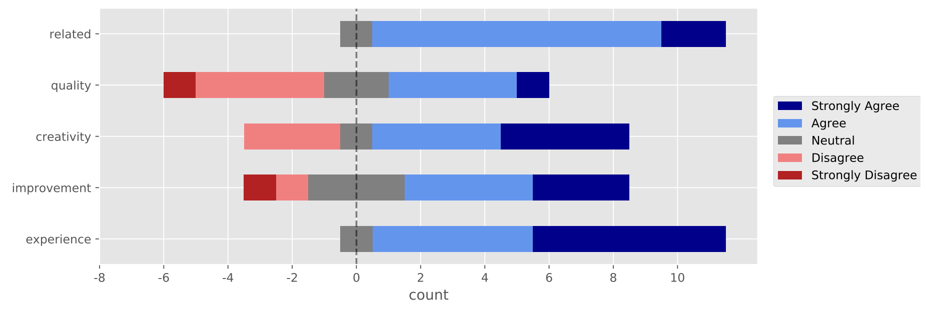


Figure 7: Distribution of responses to each question in the preliminary user study of RoboJam. Most users agreed that the responses were related to their performances and that interacting with RoboJam enhanced their experience.

RoboJam enhanced their experience. Most participants agreed that RoboJam's responses showed creativity and made their performances sound better; however, they were more uncertain about the overall quality of the responses. These results tell us that interacting with a call-and-response agent is a potentially useful addition to this app. In some ways, it would appear that this feature is useful even when the agent produces responses of uneven quality. The participants appear to have appreciated hearing their performances in context with a response and felt that this improved the sound of their own contributions. They were able to naturally cherry-pick responses until they found one that appealed to them, this involved critically engaging with their own performances as well as RoboJam's.

These results encourage us that RoboJam could be a useful and significant addition to our touchscreen music app. A more in depth user study could compare RoboJam responses with other agents that generate responses via alternative generative music processes.

6 Conclusion

This work has described RoboJam, an ANN-based musical sequence generator connected to an interactive touchscreen music app as a agent for call-and-response style improvisation. This system has the potential to enhance the user's experience, particularly when they are not able to collaborate with other users. Our agent, RoboJam, uses a novel application of MDN and RNN to model musical control data. Not only does this network model the location of interactions on a touchscreen, but it also the *rhythm* of these interactions in absolute time. This configuration distinguishes RoboJam from other typical approaches in ANN music generation; our network learns from data at the control gesture level, rather than at the note level, it also learns to perform in absolute time, rather than at preset rhythmic subdivisions. In the context of an interactive digital musical instrument, this configuration allows RoboJam to perform music in exactly the same way as users. Rather than learning to compose music, RoboJam is actually learning to perform on a touchscreen instrument.

This design choice has several implications in our results. Learning from low-level interaction data seems to be a harder task than learning from higher-level musical notes. From a musical perspective, RoboJam's performances seem underwhelming compared with high-level ANN music generators. However, there are advantages from learning to *perform* rather than to *compose*. Since a touchscreen interface can be used to control different kinds of instruments, RoboJam's output can be mapped to different kinds of synthesis processes as in our touchscreen app. Furthermore, RoboJam's responses are related to the body movements of the app user, an embodied approach that is appropriate in our application where free-form gestural exploration is emphasised.

In this research, we examined the results of evaluations of RoboJam in terms of model validation, generative power, and user experience. These studies have demonstrated that this system can generate responses that are related in movements and rhythm to the call performance. Our preliminary human-centred evaluation has shown that users felt the responses improved their performances and enhanced their experiences. This was the case even when response quality was not always highly rated.

Our results are encouraging, but they are tempered by the difficulty of training an MDRNN. Our musical data contains a wider and more abstract variety of interactions than, for example, handwritten letters of the alphabet. This may explain some of our difficulties with training. Future improvements to RoboJam could explore more curated training data and alternative mixture model designs.

The generation of music and other creative data is an exciting topic in deep learning. While it is clear that computer-generated art spurs the imagination, it is less clear how music generators can be integrated into human-centred creative processes. We have focused on modelling musical touchscreen control data and on developing a call-and-response interaction between user and agent. As a result, our system can easily be integrated into mobile musical interfaces and appears to enhance the musical experience of users in our study. Future research could expand on how models of musical control data and call-and-response agents could be used to accompany, modulate, or assess human musical performances.

References

[1] John A. Biles. Improvizing with genetic algorithms: Genjam. In Eduardo Reck Miranda and John Al Biles, editors, *Evolutionary Computer Music*, pages 137–169. Springer London, London, 2007. `doi:10.1007/978-1-84628-600-1_7`.

[2] Christopher M. Bishop. Mixture density networks. Technical Report NCRG/97/004, Neural Computing Research Group, Aston University, 1994.

[3] Axel Brando. Mixture density networks (mdn) for distribution and uncertainty estimation. Master's thesis, Universitat Politècnica de Catalunya, 2017. URL: `https://github.com/axelbrando/Mixture-Density-Networks-for-distribution-and-uncertainty-estimation/`.

[4] Florian Colombo, Alexander Seeholzer, and Wulfram Gerstner. Deep artificial composer: A creative neural network model for automated melody generation. In João Correia, Vic Ciesielski, and Antonios Liapis, editors, *Computational Intelligence in Music, Sound, Art and Design: 6th International Conference, EvoMUSART 2017 Proceedings*, pages 81–96. Springer International Publishing, Cham, 2017. `doi:10.1007/978-3-319-55750-2_6`.

[5] Douglas Eck and Jürgen Schmidhuber. A first look at music composition using lstm recurrent neural networks. Technical Report IDSIA-07-02, Instituto Dalle Molle di studi sull' intelligenza artificiale, Manno, Switzerland, 2007.

[6] A. Graves. Generating Sequences With Recurrent Neural Networks. *ArXiv e-prints*, August 2013. URL: `https://arxiv.org/abs/1308.0850`.

[7] D. Ha and D. Eck. A Neural Representation of Sketch Drawings. *ArXiv e-prints*, April 2017. URL: `https://arxiv.org/abs/1704.03477`.

[8] Gaëtan Hadjeres, François Pachet, and Frank Nielsen. DeepBach: a steerable model for Bach chorales generation. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1362–1371, International Convention Centre, Sydney, Australia, 06–11 Aug 2017. PMLR. URL: `http://proceedings.mlr.press/v70/hadjeres17a.html`.

[9] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8):1735–1780, 1997. `doi:10.1162/neco.1997.9.8.1735`.

[10] Patrick Hutchings and Jon McCormack. Using autonomous agents to improvise music compositions in real-time. In *EvoMUSART 2017*, volume 10198 of *LNCS*. Springer International Publishing, 2017. `doi:10.1007/978-3-319-55750-2_8`.

[11] Alexander Refsum Jensenius, Marcelo M. Wanderley, Rolf Inge Godøy, and Marc Leman. Musical gestures: Concepts and methods in research. In *Musical Gestures: Sound, Movement, and Meaning.* Routledge, 2010.

[12] Andrej Karpathy. The unreasonable effectiveness of recurrent neural networks. Published on Andrej Karpathy's blog, May 2015. URL: `http://karpathy.github.io/2015/05/21/rnn-effectiveness/`.

[13] I. Malik and C. H. Ek. Neural translation of musical style. *ArXiv e-prints*, August 2017. URL: `https://arxiv.org/abs/1708.03535`.

[14] Yotam Mann. AI duet. Git Repository, 2016. URL: `https://github.com/tensorflow/magenta-demos/tree/master/ai-jam-js`.

[15] Marco Marchini, François Pachet, and Benoît Carré. Rethinking reflexive looper for structured pop music. In *Proceedings of the International Conference on New Interfaces for Musical Expression*, pages 139–144, Copenhagen, 2017. Aalborg University Copenhagen. URL: `http://www.nime.org/proceedings/2017/nime2017_paper0027.pdf`.

[16] Charles Martin, Ben Swift, and Henry Gardner. anucc/metatone-analysis: Touchscreen data corpus, 2017. URL: `https://doi.org/10.5281/zenodo.1020166`, `doi:10.5281/zenodo.1020166`.

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

[18] François Pachet. The continuator: Musical interaction with style. *Journal of New Music Research*, 32(3):333–341, 2003. `doi:10.1076/jnmr.32.3.333.16861`.

[19] Bob L. Sturm, João Felipe Santos, Oded Ben-Tal, and Iryna Korshunova. Music transcription modelling and composition using deep learning. In *Proceedings of the 1st Conference on Computer Simulation of Musical Creativity*, 2016.