Towards a Machine Learning Based Control of Musical Synthesizers in Real-Time Live Performance

Nathan Sommer and Anca Ralescu
EECS Department
University of Cincinnati, ML0030
Cincinnati, OH 45221-0030, USA
sommernw@mail.uc.edu, anca.ralescu@uc.edu

Abstract

Musicians who play synthesizers often adjust synthesis parameters during live performance to achieve a more expressive sound. Training a computer to make automatic parameter adjustments based on examples provided by the performer frees the performer from this responsibility while maintaining an expressive sound in line with the performer's desired aesthetic. This paper is an overview of ongoing research to explore the effectiveness of using *Long Short-Term Memory* (LSTM) recurrent neural networks to accomplish this task.

Introduction

Electronic sound synthesizers have been used as musical instruments for more than a century, and musicians and researchers continue to explore new ways to synthesize interesting and expressive sounds. Approaches to allow humans to control synthesizers include woodwind style controllers, guitar and other stringed instrument style controllers, and controllers that map gestures to musical events. However, the most popular synthesizer controller continues to be the the piano-style keyboard.

The keyboard remains an attractive controller because it is familiar to many musicians, and because it is a natural way to tell a synthesizer to start or stop playing a sound. When a key on a keyboard is depressed, a message is sent to start playing a sound at a certain frequency. When the key is released, a message is sent to tell the synthesizer to stop.

What's missing from this is a way to control the quality of the sound once a key has been depressed. Wind instruments allow the musician to alter the quality of the sound through breath and mouth control, and bowed string instruments allow for different sounds through different bowing techniques. To allow for similar expressive sound adjustments, most synthesizers have a number of parameters that can be adjusted via knobs, sliders, wheels, pedals, and other methods. This allows for a great deal of sound control, but the number of parameters that can be controlled simultaneously is limited by the number of hands and feet the performer has, and often the performer would like to use both hands simultaneously to play the keyboard.

One way to allow for a more expressive sound during performance without requiring the human performer to directly control synthesis parameters is to use a computer to control the parameters. Many modern synthesizers are hardware or software modules that are not directly connected to a keyboard. Multiple protocols exist to control these synthesizers, such as MIDI and OSC. Keyboards, other types of controllers, and computers can send messages to these synthesizers telling them to start or stop playing a sound, or to change a synthesis parameter value. With a such a setup, a computer program can be used to monitor messages from a keyboard controller and alter the synthesis parameters in real time based on what the human is playing. This paper proposes a synthesizer control system, the *Middleman*, which implements such a setup, and is illustrated in Figure 1.

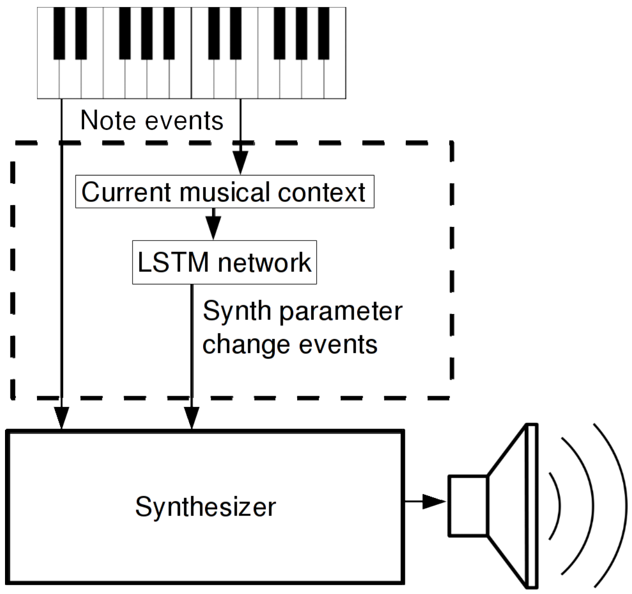

Figure 1: Basic functionality of the proposed system. Note events are received by the system and allow it to continually update the current musical context which is continually fed through the LSTM network. Note events are passed through to the synthesizer along with generated synthesizer parameter change events.

Because different musicians have different desired aesthetics, there is no one correct way to shape synthesis parameters over time during a performance. Ideally a performer would be able to teach a machine to learn to control the parameters in a way that is *consistent* with the performer's desired aesthetic. This paper explores the extent to which machine learning techniques, specifically a class of neural networks, can be used to achieve such control in a manner comparable to that of a human musician.

Expressive Music Performance

Expressive music performance has been of particular interest to researchers in the last decade, and much work has been done to attempt to model expressive human performance with machines (Kirke and Miranda, 2009). These models can be used to generate expressive musical performances by machine alone, and also for collaborative performances between human and machine.

When human musicians perform a piece of music from a written score, they inject their own expression and aesthetic into the music by varying the following musical parameters, or *performance actions* (Kirke and Miranda, 2009):

- *tempo*, the speed of the music
- *dynamics*, how loud the notes are played
- *articulation*, the transitions between notes
- *intonation*, pitch accuracy
- *timbre*, the quality of the sound

A musician may only be able to vary a subset of these actions depending on the instrument played. For example, a saxophone player can vary the timbre and intonation of a note as it is being played by altering the tightness of the mouth on the mouthpiece of the instrument, but a piano player cannot achieve this at all.

Traditionally, music composers and arrangers provide information about how pieces of music are intended to be performed through *scores*. Scores contain notation that tells musicians how loud to play, when to speed up and slow down, what articulation to use for what notes, etc. The musician ultimately decides how to interpret this information, and adds expressive subtlety to a piece of music during performance that cannot be conveyed in a score alone.

Expressive Computer Music Performance

Similarly to a human performer, a computer can perform a piece of music on an electronic instrument as it is written in a score. However, it is difficult for a computer to perform a piece of music with the same expressive subtlety as a human performer. Computers are good at following rules, but the rules for musical expression are difficult to define. Different styles of music have different rules of expression, and often what makes a particular performance of a piece of music interesting is how the musician plays with the listener's expectations of such rules.

One way researchers have tried to get computers to perform music in an expressive way is by *learning expression rules* from human performances (Widmer, 2001). Performance examples are used which include a musical score along with a recorded human performance, and algorithms are employed to explicitly or implicitly extract performance rules that a computer can use to perform unseen scores.

One of the easiest instruments with which to accomplish this task is the piano. Unlike many wind and string instruments, the piano does not allow for subtle control of dynamics, timbre, and intonation once a note has been struck. Because the piano can only play discrete notes there is no control over intonation, and the timbre can only be controlled in the same manner as the dynamics, in how hard the key is struck.

Due to this, a piano performance can easily be described with key and pedal events rather than recorded audio, and mechanical pianos can be controlled by a computer. There is an annual piano performance rendering contest called Rencon (`http://renconmusic.org/`) which evaluates performance rendering systems' abilities at performing unseen musical scores on the piano which continues to push progress in this area (Widmer, Flossmann, and Grachten, 2009). An interesting aspect of this competition is that while the contestants are computer systems, the judges are human and therefore the evaluations are highly subjective.

Human Computer Collaborative Performance

Other research explores ways in which humans and computers can collaboratively contribute to expressive performances. The OMax system (Assayag et al., 2006) allows for improvisational collaboration in real time by listening to the human musician, learning features of the musician's style, and playing along interactively. Music Plus One (Raphael, 2010) is an automatic accompaniment system which plays a scored piece along with a soloist, *following the soloist's tempo*. In addition to the piano performance rending contest, Rencon also has a "semi-automatic and interactive" category, which in 2013 was won by VirtualPhilharmony, a system that allows a human to conduct a virtual orchestra.

All of these examples are exciting works that showcase the extent to which humans and computers can work together to make music, but none of the collaborative systems developed so far address the problem put forth in this paper – *allowing a human performer to control the pitch, tempo, dynamics, and articulation of a performance, while a computer controls timbre and intonation by varying sound synthesis parameters*.

We hypothesize that a machine learning approach to this problem can be successful. Allowing musicians to train the system with example performances created by the musicians themselves will result in performances that are unique and adhere to the performers' visions. This problem also presents unique challenges from a machine learning perspective, which will be discussed in later sections.

Creating Performance Examples

Often when creating recordings in a studio, synthesizer parts are initially recorded as events rather than as audio. This allows musicians to separate the recording of note events from the recording of synthesizer parameter change events. After a keyboard performance has been recorded, parameter changes can be recorded to achieve the desired sound over the course of the recording. In this way, musicians can create interesting and expressive synthesizer recordings that could not be performed live by a single musician.

These studio event recordings can be used as training examples for our system. If the system can learn to reproduce the desired parameter changes while a human is performing, temporally changing sounds that were previously only attainable in a studio can be brought to life during live performances. This method of creating training examples is natural because musicians are already accustomed to recording this way, and allows them to use the synthesizers they are already familiar with.

Learning with Time Series Data

Many tasks which have been tackled with machine learning approaches involve time series data. Some of these tasks, such as speech recognition, involve finding patterns in time series data. Other tasks, such as numerous types of forecasting, involve predicting what will happen in the future based on what has happened in the past.

For this particular problem we are concerned with predicting the immediate future values of synthesizer parameters. These values must be predicted based on two things:

- past parameter values
- the current *musical context* of the piece being played.

The important aspects of the musical context that affect the parameter levels are defined implicitly by a musician through training examples, and so the learning system employed must be able to discover those aspects. The current musical context at any time step during a musical performance is dependent on events that have happened at previous time steps, and so the system must have an awareness of the past. Which past events are important and how long they remain important will differ for each set of training examples, and so the system must be flexible in that regard. The following sections discuss some techniques that have been used to achieve this goal in other problem domains.

Recurrent Neural Networks

Artificial neural networks have long been useful tools in machine learning due to their ability to approximate non-linear functions. While the standard artificial neural network can be useful for some time series data learning tasks, there are limitations to the model when applied to time series data.

After a standard neural network is trained, data is input via input nodes. In each subsequent layer, node activations are calculated by applying activation functions to weighted sums of the previous layer's activation values. No information about previous activation values is stored, and so the network is in the same state before each forward pass. Therefore, in order for such a network to accept time series data as input it must receive a *window* of data, or all the data after time step $t - n$ up to time step t, where t is the latest time step under consideration and n is the size of the window.

This sliding window approach has limitations because events relevant to time step t could have occurred at or before time step $t-n$, yet they will not be taken into consideration by the network because they are outside of the window. An alternative to the window approach is to use a Recurrent Neural Network (RNN) (Jordan, 1986; Elman, 1990). RNNs contain nodes which retain activation values from the previous time step, and contain *recurrent connections* from those nodes to other nodes in the network. In this manner, data can be fed into the network one time step per forward pass, and the network will learn to take into account information from past time steps when calculating output for the current time step.

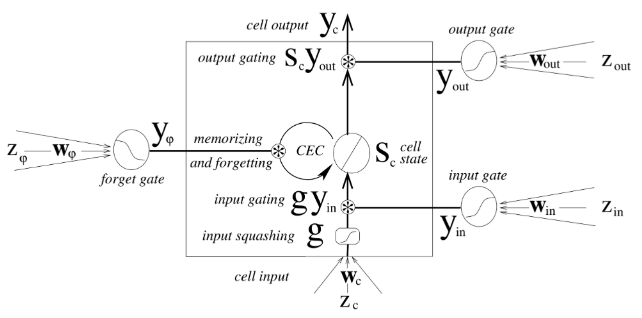

Figure 2: LSTM block with a single memory cell, taken from Gers, Schraudolph, and Schmidhuber (2003). The gates and cell input activations are calculated by passing the weighted sum of incoming connections through an activation function, as with a standard artificial neural network node. Input to the cell is scaled by the input gate's activation, output from the cell is scaled by the output gate's activation, and the cell's state is scaled by the forget gate's activation.

Simple RNNs as described above are generally trained using Backpropagation Through Time (BPTT). Using this method, errors at the current time step flow backwards through previous time steps in order to calculate changes in the network's weights. However, these errors either vanish or blow up as they travel backwards in time. As a result, simple RNNs cannot learn well when relevant events happen more than 5-10 time steps in the past (Gers, Schmidhuber, and Cummins, 2000).

LSTM Networks

Long Short-Term Memory (LSTM) RNNs overcome this limitation (Gers, Schmidhuber, and Cummins, 2000). LSTM networks contain *memory cells* which retain values between forward passes through the network. The networks also contain three types of specialized activation units called *gates*. A group of memory cells and their associated gates are organized into *blocks*. *Input gates* control write access to the memory cells by scaling values that are to be added to the memory cells; *output gates* control read access from the memory cells by scaling the values that are output by the cells; *forget gates* allow the memory cell to periodically reset by scaling the cell's current value. The gates have weighted connections from the input as well as recurrent connections from the other gates and memory cells. During training the gates learn to open and close so that the memory cells can store and accumulate values, keep them for arbitrary periods of time, and use the cells' values to affect the output as needed.

LSTM networks have been used successfully in both recognizing patterns from sequences and in generating sequences. Most notably they have proven highly effective in handwriting recognition and generation (Graves, 2013). There have been musical applications of LSTM: LSTM networks have been used to generate musical chord progressions (Eck and Schmidhuber, 2002), and for learning musical structure from scores (Eck and Lapalme, 2008), but these applications operate on the level of notes and chords and do not operate in real time. Our project explores how effectively LSTM networks can be trained to *implicitly extract* relevant high level musical structures from low level time series input, and use that information to control sound synthesis parameters.

Specific Challenges

In order for this system to be successful, several challenges must be overcome. The learning system is to be given the musical context at each time step, and so the most effective manner of encoding the current musical context must be determined. The system must run in real time, thus care must be taken to ensure it can continually predict parameter levels quickly enough. Generalization is an issue with any machine learning task, and here one must be careful not to overfit to the provided training examples so that the system can generalize well. Finally, suitable metrics must be devised to evaluate the success of the system.

Capturing the Musical Context

When this system is used, it will be given a regularly updated state of the current musical context. This information must be determined from the stream of *note on* and *note off* events received from the keyboard and should provide enough context to the learning system so that it can effectively predict the synthesizer parameter levels.

One very simple way to capture the context is to have a single input value which represents whether or not a note is currently being played. When a note on event is sent from a controller, it has two values: *pitch* and *velocity*. The pitch value indicates which key was depressed on the keyboard, and the velocity value indicates how hard the key was struck. Given the normalized velocity v of the last note played, where $0 < v \leq 1$, a very simple single input x_0 looks like this:

$$x_0 = \begin{cases} 0, & \text{if no note is depressed} \\ v, & \text{otherwise} \end{cases}$$

This input scheme can capture the timing and dynamics of the piece being played, but the pitch is ignored and it cannot take into account *polyphonic* playing, where the performer depresses more than one key at a time. If a musician merely wants parameter changes based on the timing and dynamics of *monophonic* playing however, this input scheme might be a good choice.

Another option is to have one input for each key on the keyboard. For each key i on the keyboard, the input vector element x_i looks like this:

$$x_i = \begin{cases} 0, & \text{if key } i \text{ is not depressed} \\ v_i, & \text{otherwise} \end{cases}$$

This input scheme captures pitch, timing, and dynamics, and hypothetically provides enough information for a properly trained network to be able to extract any musical context that might be relevant. However, it remains to be seen if such training is feasible. Having one input value for each key on the keyboard might require significantly larger networks which introduce longer training times and additional processing latency.

A third option is to explicitly provide the network with higher level musical features. *Melodic* information can be determined based on the intervals between consecutive notes. Expressive computer music performance systems (Widmer, Flossmann, and Grachten, 2009; Arcos, De Mántaras, and Serra, 1998) have had success using the Implication-Realization model (Narmour, 1990), which can be used to classify small local melodic structures. Such information can be determined from a monophonic performance, but becomes difficult if the performance is highly polyphonic.

Harmonic information can be explicitly determined if multiple notes are being played at once. The same harmonic intervals can mean different things depending on the *key* of the piece. Therefore, to use this information as input, the system must be trained for a specific key or the training data must be *transposed* to different keys during training to achieve generalization across keys.

Real Time Performance

Musicians using this system will be playing their keyboards, or whatever controllers they prefer. Every time a note is played, the system must predict the values of the synthesizer parameters and set them before the note event is sent to the synthesizer. This ensures that the parameters are set before the note is played so that the synthesizer creates the desired sound. Thus the system must be able to run fast enough that the total latency from the time at which the note is depressed to the time at which the note is played on the synthesizer is within the musician's acceptable latency tolerance.

In preliminary testing, relatively small LSTM networks have been able to complete a forward pass in approximately 50 microseconds on a mid-range laptop. For most musicians, added latency does not become an issue until it reaches several milliseconds. As this research progresses the networks are sure to grow in size, increasing the time required to complete a forward pass. Because forward pass time is the primary contributor to added latency, the networks cannot grow past a certain size before use will not be satisfactory to the musician. This limit will be continually evaluated as development of the system progresses.

Generalization

Any good machine learning system must be able to generalize well. Generalization becomes an issue here because it is impossible for human musicians to play something on an instrument exactly the same way twice, and often musicians will play the same musical phrase with slightly different tempo, dynamics, and articulation each time for variety. This system must be able to make satisfying parameter adjustments when musicians play phrases that are similar to the example phrases used to train the system, but not exactly the same.

Normally it is ideal to train a learning system with a large data set. Providing as many training examples as possible increases the generalizing power of the learned model. There are several possible ways to increase the number of training examples for this particular system. One is for the musician to play numerous examples of similar musical phrases and create parameter curves for each. However, this is a rather cumbersome task and the musician still might not be able to produce enough examples for satisfactory generalization.

Another way is for the system to alter the data during training in the same ways that a human might alter the playing of a musical phrase during performance. This involves changing the duration and velocity of notes, the tempo at which the notes are played, and even the notes themselves.

Altering the examples during training is similar to generating expressive computer music performances based on example human performances. The difference here is that these altered examples will never be heard, and thus do not need to sound like authentic human performances. There is generally a consistency to the way human performers make changes to the timing and dynamics of a musical phrase to achieve an expressive performance. For example, if a performer is playing an upward moving phrase, he or she might increase the tempo and dynamics as the phrase moves upwards. It could sound musically awkward to speed up and slow down multiple times during the same upward moving phrase, and as such is something that an expressive performance generating system would want to avoid. However, if one is only concerned with creating an altered training example for the sake of generalization it is not a concern if the example does not sound musically correct as a whole. It is only important that the individual changes within the phrase are consistent with what might happen note to note during a performance.

Evaluation and Measuring Success

It is important to establish metrics to determine the level of success of this approach. Here there are both subjective and objective measures of quality to consider when evaluating performance.

To subjectively evaluate such a system, it needs to be put in the hands of a variety of musicians. Different musicians will have different ideas of how to use it, and different ways of determining if it lives up to their expectations. After collecting initial feedback and seeing how musicians use the system it will be easier to determine more specific subjective evaluation criteria.

As mentioned before, the system must be able to generate satisfactory output when presented with musical phrases that are similar to the phrases used to train the system. In some cases, what the system sees as similar and what the musician sees as similar might not agree, and what might be seen as a failure of the system might be due to training examples that do not properly capture the desired behavior.

Objective metrics are easier to define. As with any supervised learning task, the general goal is to minimize training error, and to do it in as few training epochs as possible. Different methods for capturing the musical context and different network topologies and parameters can be objectively compared based on the level of error minimization achieved during training and, in the case of similar results, the amount of time taken to train. These objective metrics can be compared with subjective evaluations of performance to ensure that optimizing the objective metrics correlates with improved subjective evaluation.

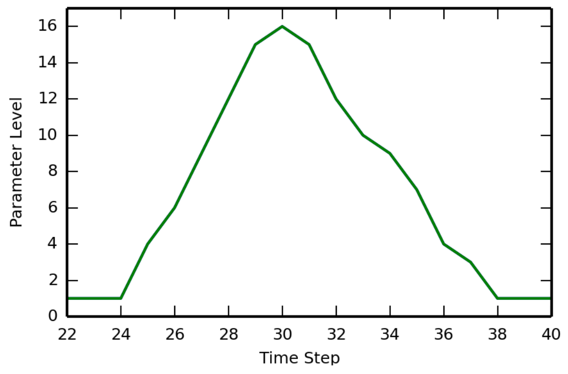

Figure 3: A simple triangle-shaped parameter modulation over 15 time steps. The network was able to learn to output this shape perfectly after $27,370$ training epochs. It is merely an approximation of a triangle due to the 16 discrete output levels.

Preliminary Results

Much of the work so far has been on developing a custom LSTM implementation and conducting training experiments to ensure that LSTM is a suitable learning algorithm for this problem. Two simple experiments are presented here. The first experiment demonstrates that the LSTM implementation presented here can learn to output a basic triangle-shaped temporal parameter change on demand. The second experiment shows that LSTM is capable of learning to detect higher level articulation patterns from an input stream and output different values based on the temporal position within a pattern.

Simple Temporal Modulation Experiment

This experiment was devised to determine how well an LSTM network can learn to output a simple triangle-shaped parameter modulation whenever a key is pressed. This shape is shown in Figure 3.

Rather than outputting a single continuous parameter value, the network outputs a vector y which contains 16 values representing discrete parameter levels. The predicted parameter level is selected by finding the maximum element in y. Each element of each target vector is set to 0 except for the element representing the desired parameter level which is set to 1. Training follows the backpropagation algorithm described in Gers, Schmidhuber, and Cummins (2000).

Because the output is discrete rather than continuous, the output pattern is merely an approximation of a triangle shape.

A single network input x_0 represents the current state of the controller at the current time step, and has two possible values:

$$x_0 = \begin{cases} 0, & \text{if no note is depressed} \\ 1, & \text{otherwise} \end{cases}$$

Each training sequence consists of 25 subsequences, distributed as follows:

- One subsequence containing the triangle-shaped modulation. Input for each time step is 1. The output starts at level 1, rises to level 16, and falls back to 1 over 15 time steps. This results in output that is a discrete approximation of the triangle shape.
- Ten subsequences representing silence. Input for all time steps is 0 and output for all time steps is level 1. The lengths of the silent sequences range from 9 to 11 time steps. Prior to each training epoch and each validation pass, these subsequences are shuffled so that the non-silent section is in a different position within the sequence for each pass.

The topology of the LSTM hidden layer consists of seven memory cells, each with input, output, and forget gates. All units in the hidden layer receive a weighted connection from the input, as well as recurrent connections from all other units in the hidden layer. The memory cells are fully connected to the output units. Gates and output units utilize a sigmoid activation function.

Networks with this topology are able to learn to output the desired shape perfectly over time steps with an input equal to 1.

Articulation Experiment

For this experiment, training data is generated to simulate notes being played with different articulation. Notes with two types of articulation are generated: *staccato* notes, which here last for a duration of 8 to 12 time steps followed by 8 to 12 time steps of silence, and *normal* notes, which last for a duration of 16 to 20 time steps followed by 1 to 4 time steps of silence.

At each time step the network is to output one of three parameter levels as follows: Level 1 is the normal level which is set while normal notes are being played and during extended periods of silence. If a staccato note is played, the parameter level should be increased to level 2 at the onset of the next note. If the next note is also staccato, then the level should be increased to level 3 on the onset of the note after that. The parameter level will stay at level 3 as long as staccato notes continue to be played. If a normal note is played after a series of 3 or more staccato notes, the level should be decreased to level 2 after the note has been sustained for 14 time steps, and then decreased to level 1 after 3 more time steps. This behavior is illustrated in Figure 4.

It is worth noting that it would be impossible for a network to learn to raise the parameter level on the onset of the first staccato note in a series of staccato notes, because at that point it is impossible to determine for how long the note will be sustained. This is a limitation of operating in real time, and must be kept in mind when creating future training examples.

Each training sequence consists of 25 subsequences, distributed as follows

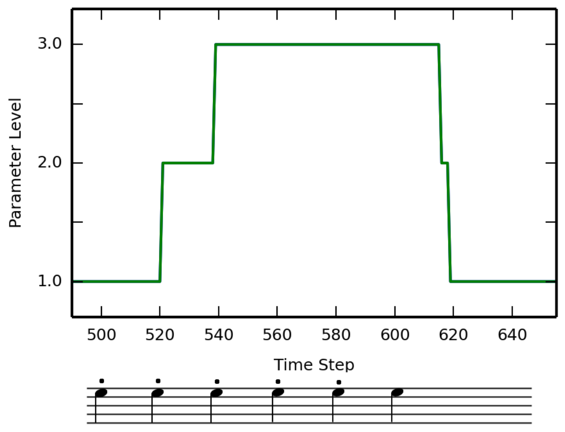

Figure 4: Illustration of the articulation experiment. Shown are 5 consecutive staccato notes followed by one normal note. This subsequence starts at time step 500 of a validation run after 90, 010 training epochs. The parameter level stays at 1 at the onset of the first note, raises to 2 at the onset of the second note, and raises to 3 at the onset of the third. The parameter level remains at 3 until most of the way through the normal note, at which point it falls back down to 2 and then 1.

- Ten subsequences of silence, each with a random duration of 1 to 50 time steps. Output is always level 1.
- Ten subsequences each consisting of a single normal note with a random duration of 16 to 20 time steps followed by 1 to 4 time steps of silence. Output is always level 1.
- Five subsequences of 3 to 5 staccato notes followed by 1 normal note. Output starts at level 1, increases to level 2 at the onset of the second note, increases to level 3 at the onset of the third note, decreases to level 2 15 time steps into the normal note, and then decreases back to level 1 3 time steps later.

Prior to each training epoch and each validation pass, these subsequences are shuffled to create a unique sequence which still contains the desired properties.

As in the previous experiment, the state of the controller is passed to the network as a single input x_0 which is 0 or 1 depending on whether or not a key is depressed. The output vector again represents discrete parameter levels, but in this case only 3 levels are used.

The input vector at time step t also contains the parameter level from time step $t - 1$. During training the target vector from time step $t - 1$ is used. During validation all elements of the output vector from time step $t - 1$ are set to 0 except for the maximum element, which is set to 1. Feeding the output from $t - 1$ into the network along with the current controller state improves training accuracy dramatically.

Perfect output was achieved in this experiment as well, using the same network topology and parameters as in the previous experiment.

Future Work

It has been shown that LSTM networks are capable of learning to output simple temporal changes in the presence of a stimulating input, and that LSTM networks can be trained to recognize articulation patterns and to output parameter levels based on provided examples. Future experimentation will focus on training networks to achieve tasks that depend on pitch and dynamics as well as articulation.

Once satisfactory performance has been established using generated data, the system will be tested by various musicians using their own training examples which will further expose the strengths and limitations of the system.

Conclusions

Employing a computer system to automatically control sound synthesizer parameters during human performance is an unexplored problem that warrants continued investigation. Results from initial experimentation suggest that LSTM networks have great potential for use in solving this problem. Successful application here will hopefully aid others in applying LSTM to other problems that involve continuous real time sequences.

Adopting a machine learning approach to this problem allows for parameter control that is consistent with performers' desired aesthetics, and allows such a system to be used by musicians that do not possess computer programming skills. Machine learning applications usually learn from large aggregations of data sampled from many individuals. This project puts the teaching power directly in the hands of individuals to allow them to fully realize their visions.

References

Arcos, J. L.; De Mántaras, R. L.; and Serra, X. 1998. Saxex: A case-based reasoning system for generating expressive musical performances*. *Journal of New Music Research* 27(3):194–210.

Assayag, G.; Bloch, G.; Chemillier, M.; Cont, A.; and Dubnov, S. 2006. Omax brothers: A dynamic topology of agents for improvization learning. In *Proceedings of the 1st ACM Workshop on Audio and Music Computing Multimedia*, AMCMM '06, 125–132. New York, NY, USA: ACM.

Eck, D., and Lapalme, J. 2008. Learning musical structure directly from sequences of music. *University of Montreal, Department of Computer Science, CP* 6128.

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

Elman, J. L. 1990. Finding structure in time. *Cognitive Science* 14:179–211.

Gers, F. A.; Schmidhuber, J. A.; and Cummins, F. A. 2000. Learning to forget: Continual prediction with lstm. *Neural Computation* 12(10):2451–2471.

Gers, F. A.; Schraudolph, N. N.; and Schmidhuber, J. 2003. Learning precise timing with lstm recurrent networks. *The Journal of Machine Learning Research* 3:115–143.

Graves, A. 2013. Generating sequences with recurrent neural networks. *CoRR* abs/1308.0850.

Jordan, M. I. 1986. Serial order: A parallel distributed processing approach. Technical Report ICS Report 8604, Institute for Cognitive Science, University of California, San Diego.

Kirke, A., and Miranda, E. R. 2009. A survey of computer systems for expressive music performance. *ACM Computing Surveys* 42(1):3:1–3:41.

Narmour, E. 1990. *The Analysis and Cognition of Basic Melodic Structures: The Implication-Realization Model*. University of Chicago Press.

Raphael, C. 2010. Music plus one and machine learning. In *Proceedings of the 27th International Conference on Machine Learning*, ICML 10.

Widmer, G.; Flossmann, S.; and Grachten, M. 2009. Yqx plays chopin. *AI Magazine* 30(3):35.

Widmer, G. 2001. Discovering simple rules in complex data: A meta-learning algorithm and some surprising musical discoveries. *Artificial Intelligence* 146:129–148.