Neural Networks for Simultaneous Classification and Parameter Estimation in Musical Instrument Control

Michael Lee
Adrian Freed
David Wessel

Center for New Music and Audio Technologies (CNMAT)
University of California at Berkeley
1750 Arch Street
Berkeley, CA 94709
(510) 643–9990
lee, adrian, wessel@cnmat.cnmat.berkeley.edu

ABSTRACT

In this report we present our tools for prototyping adaptive user interfaces in the context of real–time musical instrument control. Characteristic of most human communication is the simultaneous use of classified events and estimated parameters. We have integrated a neural network object into the MAX language to explore adaptive user interfaces that considers these facets of human communication. By placing the neural processing in the context of a flexible real–time musical programming environment, we can rapidly prototype experiments on applications of adaptive interfaces and learning systems to musical problems. We have trained networks to recognize gestures from a Mathews radio baton, Nintendo Power Glove™, and MIDI keyboard gestural input devices. In one experiment, a network successfully extracted classification and attribute data from gestural contours transduced by a continuous space controller, suggesting their application in the interpretation of conducting gestures and musical instrument control. We discuss network architectures, low–level features extracted for the networks to operate on, training methods, and musical applications of adaptive techniques.

1. INTRODUCTION

Human communication takes place through many different media such as audio, visual, or touch. In order for a someone to communicate an intention, they must channel it through the available media. Consider a conductor and an orchestra. During a musical performance, the only practical medium of communication is visual (a screaming conductor would detract from the performance). When a conductor wants the orchestra to play louder they might wave an open hand in the air and the orchestra plays louder according to the speed of his hand movement. Immediately following, the conductor makes a tight fist and moves it up and down and the orchestra changes the tempo of the music to match the speed of the conductor's moving fist. In this example, communication has taken place in the visual medium and intention is successfully translated in to and out of physical gestures.

Characteristic of most human communication is the simultaneous use of classification information and parameter estimation. In the conductor/orchestra example, the orchestra members were able to classify the conductor's gestures and then extract information about them. Note that the classification needs to take place before the gesture can be fully interpreted. For example, the orchestra members cannot use the speed of the conductors moving hand information until they know what context it should be interpreted in.

Key issues in human interfaces are determining how mappings between intention and gesture develop and determining what degree of adaptability to expect from each party. The problem can be simplified by first

 0-8194-0871-9/92/$4.00

assuming that both communicating parties know the vocabulary of gestures (bootstrapping could be complicated). Each party initially has some internal representation for these symbols. Learning the translations is a process in which each party critiques the other's representation and either changes them accordingly or builds an additional interface layer. In our conductor/orchestra example, the conductor and orchestra both know that a volume and tempo gesture exist. In addition, each has a prior notion of how to represent them in the available communication media. As the conductor and orchestra rehearse together, both adapt their representations until they reach an agreement (note that an insistent conductor may adapt less than the orchestra).

In the remainder of the report we present a brief background on computers and music, discuss the MAX programming environment and the neural network simulator MAXNet. A description of our hardware environment, experiments, and a conclusion follows.

2. MUSIC and COMPUTERS

Musical applications of computers range from computer generated music to computer aided control of tone generators. In this report we focus on using computers as real–time performance tools to control musical instruments.

The control interface of acoustical musical instruments is usually determined by the physical nature of the instrument. Many times, the predetermined interface is not physically intuitive to the musician or doesn't match the musician's physique (i.e., it is more difficult for pianists with small hands to play one handed tenth intervals than a pianist with large hands). Instruments may be adapted to a musician's physique, but this is expensive, and may result in poorer sound than a standard or mass produced instrument.

Computers and electronic instruments have allowed approaches to the instrument control problem. It is now possible to decouple the physical gestures used to control an instrument from the control interface. In addition, electronic instruments are no longer constrained to the one gesture/one sound basis of most acoustic instruments [Wess91]. Computer instrumentation allows us to use one gesture to represent a sequence of notes. This opens up the opportunity to modulate a sequence of notes after it has been launched. The central issue in musical applications is not just gesture recognition, but gesture understanding. The attribute information of a musical gesture is equally as important as its classification information. If we can build adaptive systems that learn the mapping from arbitrary physical gestures to control gestures of the device, the result would be an instrument that responds in a rich, personal and intimate way.

3. MAX

MAX is a commercially available object oriented visual programming environment, which features a real–time scheduler[PuZi90]. MAX runs on Macintosh computers and was developed by Miller Puckette of IRCAM and David Zicarelli of OpCode Systems. Users can schedule events with one millisecond resolution. The scheduler operates in either non–preemptive or preemptive mode (preemptive is also called Overdrive in MAX terminology). In non–preemptive mode, once a thread is started it does not give up control of the processor until it agrees to do so. In Overdrive, the midi port and internal timer is given priority and can interrupt the currently executing thread and take control of the processor. The interrupted thread is allowed to continue when the interrupting thread completes.

MAX programs, also referred to as patches, are written by connecting objects in a workspace called a patcher. Objects are instantiated by selecting a small icon from a palette of available object classes and placing them in the patcher. Instances of objects are usually represented by boxes or icons which have

input ports at the top or output ports on the bottom. Communication channels are created by connecting a line between objects using click and drag techniques. MAX provides many built in objects to generate and manipulate numerical, textual, mouse, keyboard, and input and output MIDI data. MAX's graphical editor follows the standard Macintosh graphic manipulation conventions allowing new users to become rapidly productive.

A major strength of the MAX environment is the ability to extend the language by adding external objects written in C [Zica90]. Extending the language gives the user the power to create objects which perform computations more efficiently than networks of MAX objects alone. The message interface between the MAX and external objects is well defined so that new external objects transparently integrate into the system. One object we have written for MAX is an external object called maxFace that accesses a data acquisition system consisting of multichannel A/D converters. This device gives us a direct link from the outside world into the MAX environment. With this link, we can explore hardware issues of alternate controllers and user interfaces.

We have also added a neural network simulator to the MAX environment called MAXNet. By placing neural computations in the context of a real–time scheduler, we can rapidly prototype experiments on applications of adaptive user interfaces and learning systems to musical problems. The combination of real–time scheduling and extensibility makes MAX a flexible environment well suited for prototyping real–time adaptive systems.

4. MAXNet

4.1 Architectural Parameters

MAXNet is an external MAX object that simulates multilayered feed forward networks and Jordan and Elman style recurrent networks [Elma88][Jord86]. The user can specify several architectural parameters: the number of input and output neurons, the number of neurons per hidden layer, the number of hidden layers, and the function each neuron computes. Each layer is fully interconnected to adjacent layers.

Jordan and Elman recurrent networks are similar to strictly feed forward neural networks with the addition of context neurons that get their input from either a hidden (Elman) or output (Jordan) neuron. The extra context units are like input units, which obtain their inputs from other neurons instead of an external stimulus. This can be achieved by configuring a network with the number of input neurons equal to the number of external stimuli sources plus the number of output or hidden neurons (depending on recurrent style). The modified input patterns will be a vector consisting of the normal stimulus vector prepended to a vector describing the feedback paths. Another external control adjusts the amount of feedback from a context neuron to itself (this gives the neuron some memory). [Todd89] describes musical applications of recurrent neural networks.

Each layer of neurons computes either linear or sigmoidal functions. We have given both of these functions an additional slope parameter which effectively multiplies the net input value to the neurons. Setting the parameter to less than 1 decreases the slope of a sigmoid and makes the function behave more linearly. A slope greater than 1 results in a function that behaves more like a threshold function. Equivalent results are obtained if the training algorithm simultaneously multiplies all incoming weights by the same number, but learning this parameter speeds convergence.

MAXNet optionally graphically displays the state of the network in real–time. MAXNet uses color to represent the weight values and the size of the hole in the middle of a neuron to represent its activation

level (see Figure 1). Connecting MAX user interface objects, such as sliders, to MAXNet allows for exploration of how the activation levels of input neurons affect the neurons in other layers.

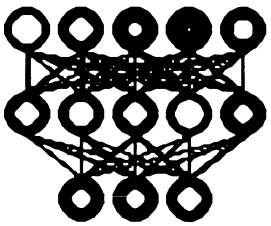

Figure 1. MAXNet Graphic Display

4.2 Training Parameters

MAXNet uses the classic back–propagation training algorithm to find the network parameters [RuMc86]. In addition to the usual learning rate and momentum parameters, MAXNet has parameters associated with learning the slope functions of the neurons and a parameter for weight decay [WeRH91]. For Jordan recurrent nets, MAXNet uses teacher forcing to speed convergence. In teacher forcing, the forward and backward pass are done as normal back–propagation technique, however on the next time step, the desired context is loaded into the context neurons instead of the actual context that results from the forward pass of the non–fully trained network.

MAXNet can optionally add hidden units according to some error criteria. This means that the number of hidden units does not need to be determined *apriori*. A new node is added if the error has not decreased more than some percentage over the last n epochs [HiYH91]. User parameters are the maximum number of hidden units (to terminate learning), percentage of error decrease, and the number of epochs to train before applying grow criteria.

MAXNet has a companion program, MACNet, which runs on the Macintosh and UNIX machines to facilitate training. Both code objects have identical capabilities. The most efficient way to use MAXNet is to setup a data acquisition harness in MAX and collect the data into a training file. Then use MACNet to learn the input/output mapping and save the weights file. Finally, attach MAXNet to the acquisition harness, read in the weights, and then compute. Both simulators use compatible text files, which eases examining and editing weights.

4.3 Performance

Training MAXNet is computationally intensive, but once the weights have been found, running MAXNet feed forward requires less computing resources. There is a multiply/add for each weight and a function evaluation for each hidden and output neuron. For large nets, the number of connections exceeds the number of neurons so we can assume that most of the time will be spent computing weighted sums. On a 16MHz 68020 without a math coprocessor, we can achieve at least 200K connections/Sec. This translates to one forward pass through a three layer net with 10 neurons on each layer in every millisecond.

4.4 MAXNet in MAX

Because the messaging interface in the MAX environment is well defined, any object in the MAX environment can have direct access to either the input or output of MAXNet. Embedding a neural net

computation in a MAX patch is done by simply connecting objects to it in the graphical editor. Multiple MAXNet objects together can be tied together to produce networks made up of subnets.

5. DATA ACQUISITION

5.1 Hardware

Our hardware data acquisition environment consists of a Macintosh IIci equipped with a GreenSprings SpringBoard 16MHz 68020 coprocessor card and a 12–Bit A/D Converter (ADC). The SpringBoard features dual ported RAM that is accessible by both the local 68020 and the host processor via the nuBus. The SpringBoard is able to supervise the I/O independently while the host processor handles other tasks such as the graphics interface.

The ADC is a separate hardware I/O module that plugs into a connector on the SpringBoard. It can be configured to read up to 20 independent single ended input channels, 8 differential input channels, or any combination in between. The analog signal passes through a programmable gain amplifier and channel selection circuitry before it reaches the converter. A gain dependant settling time must be observed when either switching channels or gain settings before initiating a conversion. Settling time is 14 microseconds for unity gain and a conversion competes in 15 microseconds . In certain configurations, over 60,000 conversions per second is achievable.

5.2 Software

Software for the SpringBoard may be developed using native macintosh development tools such as MPW. After compiling and linking the code under MPW, it is converted into a form that can be downloaded and run on the SpringBoard. The software that we have written for the SpringBoard is responsible for configuring and periodically reading the ADC, and storing the readings in local memory. On the Macintosh side, we have written an external object for the MAX environment, maxFace, which simply reads the SpringBoard's local memory through the nuBus and brings it into the MAX environment. The nuBus read is intiated in MAX by sending a maxFace object a read message and can be initiated on a periodic basis by using a simple MAX clocking mechanism such as a metronome.

5.3 Sensors

5.3.1 Mathews Radio Baton

The Mathews Radio Baton was developed by Max Mathews of CCRMA[Math91]. The drum consists of frequency multiplexed transmitting antennas and an array of receiving antennas. The antennas can be made out of inexpensive aluminum foil and can be configured in many different geometries. The received signal passes through a phase sensitive detector circuit and a low pass filter and outputs a voltage between 0 and 5. The oscillators that drive the transmitters are also used as a reference signal for the receivers in order to reduce noise effects. The outputs of the detector circuits are brought into MAX through the SpringBoard ADC and the maxFace MAX object.

The principle behind the drum is to take advantage of the near–field effects of radio systems. If the receiver is in the near field of the transmitter, the signal strength is proportional to the capacitance between the transmitter and receiver. The capacitance is, in turn, a function of the distance between the two antennas.

5.3.2 Nintendo™ Power Glove

The Nintendo™ Power Glove by Mattel was designed for use with the popular Nintendo video games. It uses a combination of sonic range finding and resistive strips to measure information about the position of the hand and fingers. In our applications, we have removed the resistive strips from the Power Glove and mounted them in our own glove. A simple circuit was connected to each of the resistive strips to output a voltage between 0 and 5. MAX programs measure these values through the maxFace MAX object and the SpringBoard ADC.

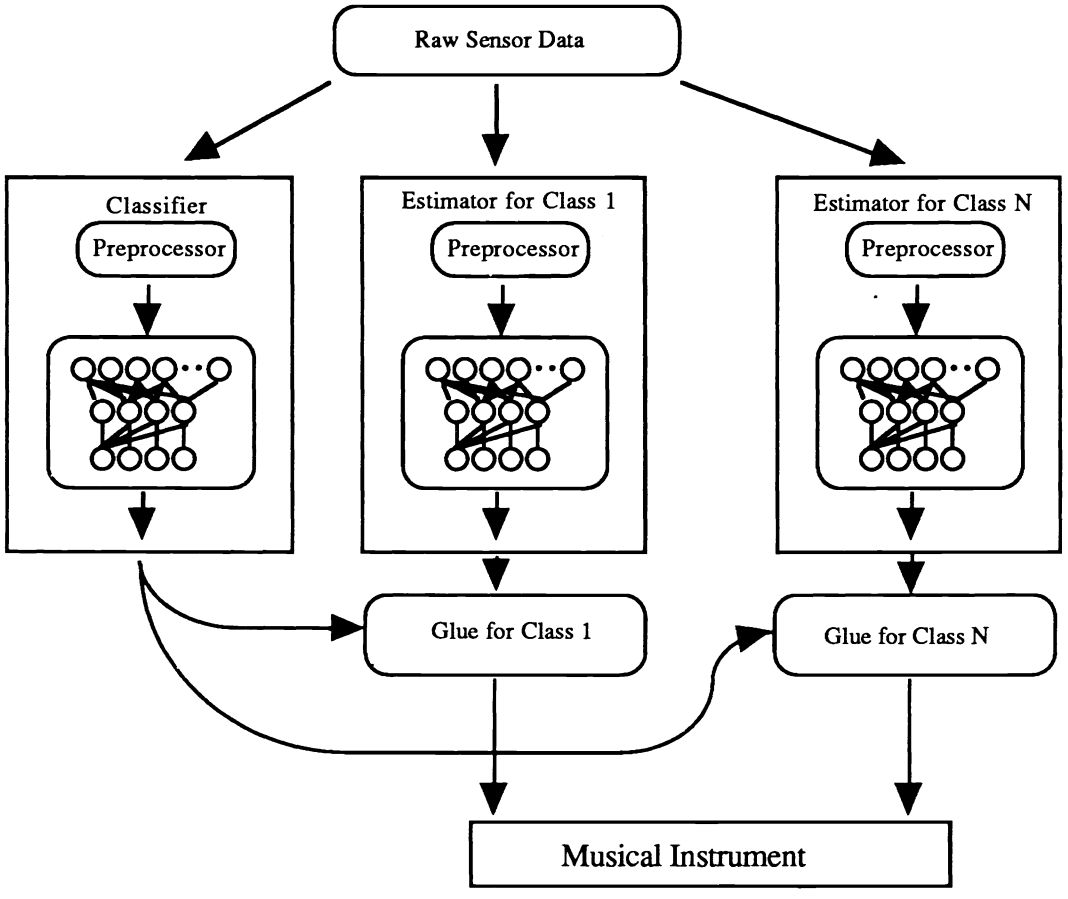

Figure 2. System Architecture

5.3.3 MIDI Keyboard

The Musical Instrument Digital Interface (MIDI) is an industry standard protocol that was developed to allow different electronic musical devices to communicate with each other. MIDI is an event based serial interface, which runs at 31.2Kbits/sec. Typical events are 'note on' and 'note off' which describe whether a change in note state has occurred. For example, to ask a tone generator to sound a note for 1 second, we send it a 'note on' message and follow it by a 'note off' message 1 second later. A variety of MIDI control generators, or MIDI controllers, exist with the keyboard and percussion being among the most popular.

The MIDI Keyboard used in our experiments is a commercial available instrument with piano keys. Each key is capable of sending pitch, velocity, and after touch information. The keyboard has several sliders, buttons, and inputs for various foot pedals and several. An inexpensive MIDI interface connected to the serial port of our Macintosh provided MIDI I/O. Built in MAX MIDI communication objects provided MIDI I/O within MAX.

6. EXPERIMENTAL RESULTS

We have designed a few experiments to show neural networks applied to musical instrument control and demonstrate our adaptive user interface prototyping capabilities. Other applications can be found in [LeFW91]. The devices incorporated into these experiments are the Mathews radio baton, Nintendo Power Glove™ and a MIDI keyboard gestural input devices. We present the low level features, training methods, and results for these applications below.

6.1 System and Network Architectures

Our architectures for simultaneously producing classification and parameter estimate information are highly modular and can be divided into three major pieces; a classification module, a group of parameter estimators, and some glue to combine the output of the classifier and estimator (see Figure 2).

The classifier module consists of a preprocessor section and a feed forward neural network classifier. Input to this module is raw data and the output is a vector of length equal to the number of classes. The preprocessor is designed on a per application basis to assist the network in discriminating between the classes. It takes the raw data and produces a feature vector for presentation to the network. Any prior information about a problem is injected at this point.

The classifier module is then trained by back–propagation using a training set that reflects the prior probability distribution of classes. If the added constraint that all the output values sum to one is included into the cost function, the output vector contains the Bayesian posterior probabilities of class membership for a given feature vector [BoMW90a][BoMW90b].

In parallel with the classifier, a group of parameter estimation modules runs for each class. These estimation modules process data in the same manner as the classification module; raw input passes through a preprocessor and then flows through a neural network. The difference between the two types of modules is that estimator uses the neural network as a multidimensional function approximator and the output of the module as a control vector. Note that the feature vectors of the estimator modules may partially or completely overlap or overlap with the feature vector of the classifier.

After the gesture has been segmented (the end of the gesture is marked), the glue logic combines the control vectors with the corresponding classification probability and then routes the modified control vector to the device. This architecture can be theoretically compacted into a single network and trained altogether. However the number of system parameters increases and makes training more difficult.

Training is terminated in our networks when a cross validation error is no longer decreasing.

6.2 Controlling an Electronic Instrumentalist

6.2.1 Description

This experiment realizes the simple example used throughout the first part of the report; a conductor controlling an instrumentalist that only understands how to change volume and tempo. The instrumentalist

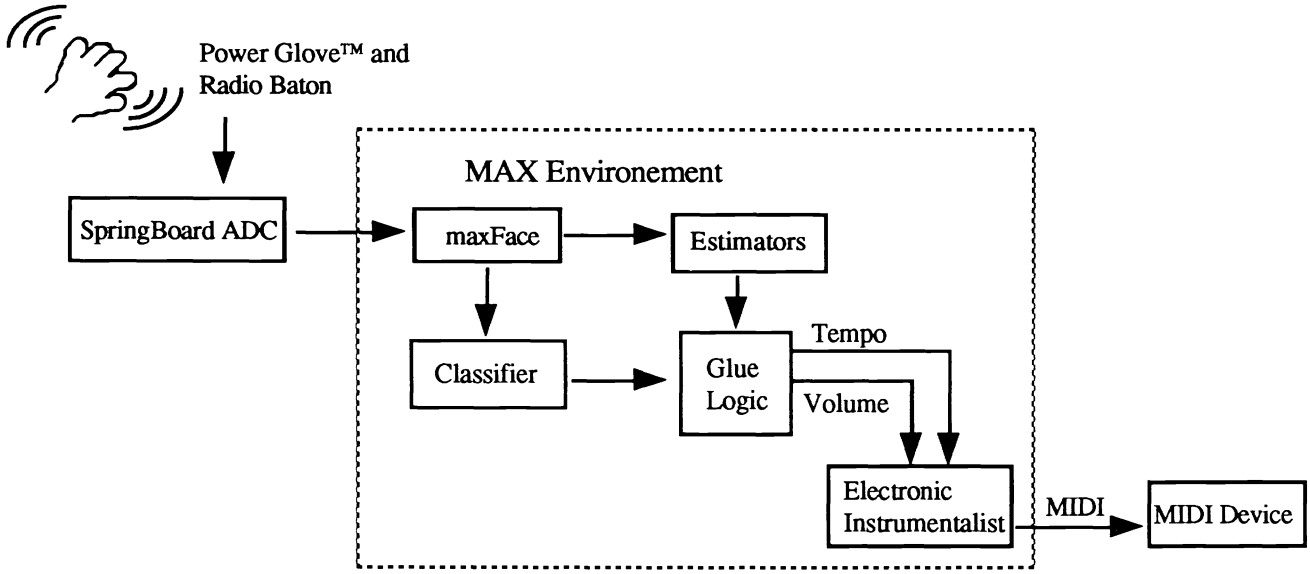

Figure 3. The Electronic Instrumentalist

is a simple note sequence player that has information on what notes to play, how to phrase them (inter–note temporal spacing and volume), and initial tempo and volume. The knowledge of initial tempo and volume corresponds to a human instrumentalist taking notes during rehearsal. Volume and tempo are the two control parameters of the instrumentalist. The network used to control the instrumentalist has been constructed to adjust only one parameter at a time.

6.2.2 Low–Level Features and Training Harness

The two classes of gestures we wanted to discriminate were the tempo and volume gestures. To implement this system, we used a combination of the Mathews radio baton and the Power Glove™. The speed was derived from the positional data provided by the radio baton and was a simple measure of the time to change linear direction. The power glove provided a hand shape representation in finger resistor space.

We collected training data for the classifier by asking the user to replicate hand shapes for the two gestures and sampling the finger resistor values. Data to train the estimator networks were obtained by first setting limits on the range of valid tempi and volume. We then sampled this range and asked the user to wave his hand at the rates corresponding to each of the sampled points.

6.2.3 Results

We used an open hand gesture to denote the volume class and the closed fist to denote the tempo class. The two classes of gestures were quite distinct and only required a 4x4x3 network (one output unit was

used as a "don't know" class). Both our training and test set consisted of 20 samples of each gesture. The network was able to achieve an average pattern error of 0.001 in 1000 epochs using a learning rate of 0.2 and momentum of 0.5.

In this experiment, we constructed a simple mapping from hand velocity to tempo or volume. The training and test set both had 10 samples of speed–velocity or tempo pairs. Both functions turned out to be monotonic and only required a 1x3x1 network. They reached a sufficiently low error measure in about 600 epochs using a learning rate of 0.2 and momentum of 0.5.

The system worked reasonably well, however the segmentation strategy was not reliable. The glue code consisted of thresholding the classifier outputs and gating the output of the estimators. Through this logic, the glue code was responsible for controlling the focus of interface. By replacing the glue code with another neural network, we might obtain more reliable results.

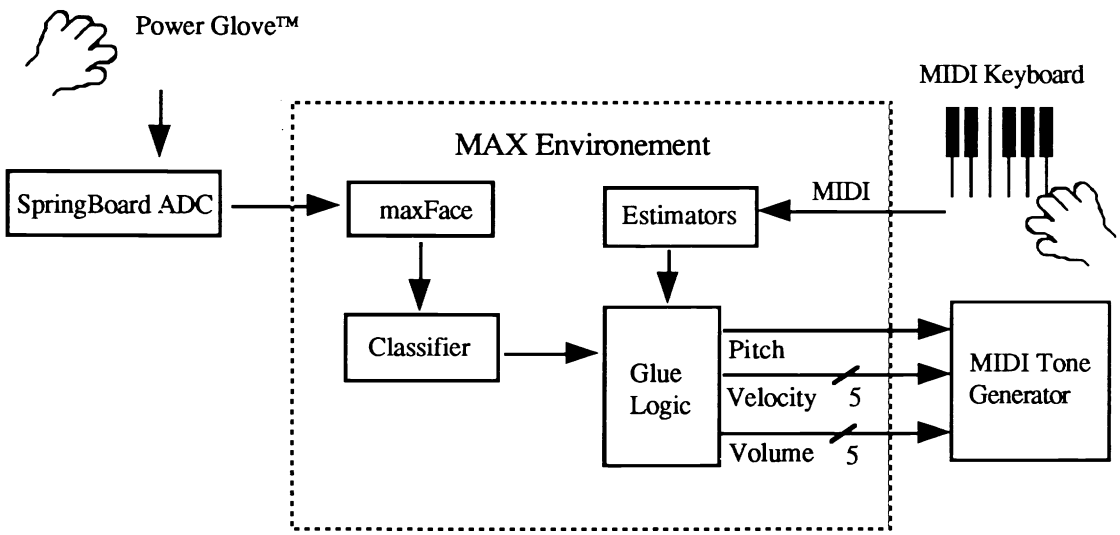

Figure 4. Tone Bank Control

6.3 Controlling a Bank of Tone Generators

6.3.1 Description

This experiment demonstrates the potential for neural networks to dynamically control the mixture of a bank of tone generators. The hand shape controlled a mixture of several timbres. Control parameters of the device were the individual levels of each timbre, pitch, and velocity. We used the MIDI keyboard as the segmentor (segmentation is resolved when pitch and note onset are determined) and to provide pitch and a velocity measure. Because each timbre had different velocity response curves, we used estimator modules to match and rescale the volume of the bank to the liking of the user. The user controlled the composite timbre of the sound with one hand and the triggered the note with the other.

6.3.2 Low-Level Features and Training Harness

In this experiment, we mixed five guitar timbres together. The control devices we used were the power glove and MIDI keyboard. The MIDI keyboard provided pitch and velocity data while the glove controlled the mix. The timbres were a muted, clean, overdriven, distorted, and feedback guitar. The

corresponding hand shapes were an open hand, thumb touching forefinger, first three fingers touching, a fist, and thumb and small finger extended respectively. The values of the finger resisters were fed directly into the classification network. The velocity was fed directly into the estimator networks.

Training data was gathered in the same manner as the above example.

6.3.3 Results

The classifier was able to discriminate among the five gestures using a 4x5x5 network. There were 20 samples of each gesture in both the training and the test set. The network achieved an average pattern error of 0.02 after 1000 epochs using a learning rate of 0.2 and momentum of 0.5.

The functions the estimator networks were required to perform were also quite monotonic and each required only a 1x3x1 network. The experience training these networks were similar to those of the orchestra example.

The system had reasonable response in real–time. Although training data was taken to fit most of the system parameters, several iterations had to be performed for the velocity scaling. This was due to the inconsistencies between physical reality and mental expectations of the user.

Our experiment used the MIDI keyboard as the segmentor and spatial locator. The radio baton is an alternative for applications such as playing drum timbres. For this application, we can use the information about the trajectory to control other parameters. One extension is to feed the finger resistor values into the estimator networks in addition to the velocity measure. Access to the gestural trajectory opens up possibilities of learning to associate arbitrary gestural trajectories with control trajectories.

7. CONCLUSIONS

The real–time scheduling, graphical programming interface, and language extensibility features of MAX combine to make it an environment well suited for prototyping real–time adaptive user interfaces. Neural computations are integrated into MAX programs by graphically connecting the MAXNet neural network simulator object to other MAX objects.

We have demonstrated two musical instrument control examples where the neural networks were used to make simultaneous use of both classification data and parameter estimates. In these examples, arbitrary gestures were transduced and mapped into device control parameters.

Neural networks are well suited to problems in musical instrument control because they can help obtain the subtle, complex, fluid control required. However, intelligently preprocessing the inputs remains paramount. Certain experiments did not work well until we changed the input representation to the networks. Any system that attempts to recognize continuous gestures must deal with segmentation and temporal problems. The experiments that we performed handled these problems in an ad hoc way. This illustrates the benefits of being able to embed all the knowledge available in the adaptive system, i.e. we need rich environments like MAX in addition to neural networks.

8. FURTHER EXTENSIONS

There are several adaptive system issues that our work has brought to our attention, such as training systems with multiple adaptive elements. None of our current experiments admits the fact that the human is an adaptive element too. For example, how do we place the human in the loop. Should we let the human be the error function? In our experiments, the training did not occur in real–time. Thus feedback

on how the human performed a gesture was not immediately available. We need to explore other learning strategies, or even other adaptive architectures that addresses this problem.

Other issues to be addressed are plasticity and scalebility of neural networks. In our applications, increasing the vocabulary is a much needed feature. Simply adding new patterns to the training set increases the training time. Training only on new patterns may damage the patterns already stored in the network.

The examples included in our report could have been approached using classical methods, however we feel that there are significant advantages to using neural networks as controllers when we experiment with more complex synthesis models and modes of control. For example, associating gestural trajectories with control trajectories in control spaces of with excess degrees of freedom. This class of problems seem well suited for the forward modelling techniques by [JoRu90].

9. ACKNOWLEDGMENTS

The authors would like to thank Guy E. Garnett and Leslie Delehanty for their diligence and support.

10. REFERENCES

[BoMW90a] Bourlard, H., Morgan, N., Wellekens, C.J., "Statistical Inference in Multilayer Perceptrons and Hidden Markov Models with Applications in Continuous Speech Recognition." *Neurocomputing*, Fogelman, F., Herault, J., eds., NATO ASI Series, Vol. F68.

[BoMW90b] Bourlard, H., Wellekens, C.J., "Links Between Markov Models and Multilayer Perceptrons." IEEE Transactions on Pattern Analysis and Machine Intelligence, Vol. 12, No. 12, 1990.

[Elma88] Elman, J.L., "Finding Structure in Time." Technical Report 8801: La Jolla: University of California at San Diego, Center for Research in Language.

[GHAL91] Guyon, I., Henderson, D., Albrecht, P., Le Cun, Y., Denker, J., "Writer Independent and Writer Adaptive Neural Network for On–line Character Recognition." To appear in IWFHR–2, 1991.

[HiYH91] Hirose, Y., Yamashita, K., Hijiya, S., "Back Propagation Algorithm Which Varies the Number of Hidden Units." *Neural Networks*, Vol. 4, 1991.

[HoSW89] Hornik, K., Stinchcombe, M., White, H., "Multilayer Feedforward Networks are Universal Approximators." *Neural Networks*, Vol. 4, 1989.

[JoRu90] Jordan, M.I., Rumelhart, D., "Forward models: Supervised learning with a distal teacher," submitted to *Cognitive Science*, 1990.

[Jord86] Jordan, M.I. "Serial Order: A Parallel Distributed Processing Approach." Technical Report ICS–8604. La Jolla: University of California at San Diego, Institute for Cognitive Science, 1986.

[LeFW91] Lee, M., Freed, A., Wessel, D., "Real–Time Neural Network Processing of Gestural and Acoustic Signals," Proceedings of the International Computer Music Conference, Montreal, 1991.

[Math91] Mathews, M., "The Radio Baton and Conductor Program, or: Pitch, the Most Important and Least Expressive Part of Music," Computer Music Journal, Vol. 15, No. 4, 1991.

[MaSc89] Mathews, M., Schloss, A., "The Radio Drum as a Synthesizer Controller," Proceedings of the ICMC, 1989.

[MiSW90] Miller, W.T., Sutton, R.S., Werbos, P.J., eds., *Neural Networks for Control*, MIT Press, Cambridge, MA, 1990.

[Pao89] Pao, Y.H. *Adaptive Pattern Recognition and Neural Networks*, Addison–Wesley, Reading, MA, 1989.

[PuZi90] Puckette, M., Zicarelli, D., *MAX – An Interactive Graphic Programming Environment*, Opcode Systems, Menlo Park, CA, 1990.

[Rubi91a] Rubine, D., "Criteria for Gesture Recognition Technologies," Position Statement for Workshop on Pattern Recognition and Neural Networks in Human–Computer Interaction, CHI '91, 1991.

[Rubi91b] Rubine, D., "Specifying Gestures by Example." Proc. SIGGRAPH 91. ACM, 1991.

[RuMc86] Rumelhart, D.E., McClelland, J.L. *Parallel Distributed Processing: Explorations in the Microstructure of Cognition*, Vols. 1 and 2, MIT Press, Cambridge, MA, 1986.

[Todd89] Todd, P., "A Connectionist Approach to Algorithmic Composition," Computer Music Journal, Vol. 13, no. 4, 1989.

[Verc84] Vercoe, B., "The Synthetic Performer in the Context of Live Performance" Proc. Int'l Computer Music Conf., Computer Music Association San Francisco, 1984, pp. 199–200.

[WeRH91] Weigend, A.S., Rumelhart, D.E., Huberman, B.A., "Generalization by Weight–Elimination with Application to Forecasting," *Neural Information Processing Systems 3*, Morgan Kaufman: San Mateo, 1991.

[Wess91] Wessel, D., "Instruments That Learn, Refined Controllers, and Source Model Loudspeakers," Computer Music Journal, Vol. 15, no. 4, 1991.

[Zica90] Zicarelli, D. "Writing External Objects for MAX," Opcode Systems, Menlo Park, CA, 1990.