Predicting Reinforcement of Pitch Sequences via LSTM and TD

Judy A. Franklin
Computer Science Department, Smith College
jfranklin@cs.smith.edu

Abstract

We examine the use of a recurrent neural network called Long Short-Term Memory (LSTM) with a prediction algorithm called temporal difference (TD) to predict the outcome of a music pitch sequence, while the sequence is being played. This is part of a larger system that will use this prediction in order to choose pitches to play. We describe our previous results using the LSTM network for musical tasks and then show its ability to predict a positive or negative outcome for a short musical task of chromatic lead-in to a chord tone. Then we describe its ability to predict positive outcomes when certain chord tones are played on the last beat of each bar of ii-V-I chord progressions.

1 Introduction

Reinforcement Learning is a method of machine learning that takes action, such as playing a note or sequence of notes, and receives a reward value that indicates the effects of the action: how good or bad it is. When sequences are to be learned, it is necessary to predict the effects of an action, well before the reinforcement value is delivered.

Reinforcement learning has generally been used as a tabular method, where each state has a corresponding entry in a table and the predicted value is learned for each state. There has been some work in combining reinforcement learning with function approximators such as neural networks (most notably is Tesauro's champion-level backgammon player (Tesauro 1994; Sutton and Barto 1998). The approximator learns a nonlinear function of each state. Th eoutput of the approximator is the value of the state. This enables faster learning for similar states, and generalization for new states. A reinforcement learning algorithm chooses the state with the maximal value. This paper decribes experiments in predicting the reinforcement value of pitch sequences. The nonlinear approximator is a recurrent neural network combined with a temporal difference (TD) prediction algorithm.

Nonlinear neural networks consist of two or more layers of small processing units that are connected to the next layer by weighted connections.

The output of each layer is fed forward through these connections to the next layer, until the output layer is reached. Each processing unit sums the weighted inputs it receives and passes the sum through a nonlinear function, usually the sigmoid or hyperbolic tangent. In an iterative training procedure, example inputs and the target outputs are presented to the network repeatedly. The network can learn a nonlinear function of the inputs by adjusting the weights (on the weighted connections) on each iteration. Such networks are useful for pattern matching and classification, and have been explored within the computer music community to classify chords(Laden and Keefe 1991), to detect musical styles (Dannenberg, Thom, and Watson 1997), and other tasks (Todd and Loy 1991; Griffith and Todd 1999).

A recurrent neural network includes the past output of some of its processing units as part of its state, in a feedback configuration. This enables it to learn sequences. Todd (1991) used a Jordan recurrent network (Jordan 1986) where the output of the network is fed back to the input layer. The actual input is a decaying average of the most recent output values, providing a decaying memory of the melody being played by the network. This system can learn melodies with 20 or more notes with durations that are multiples of sixteenth notes, and can use the network to generate new tunes.

Mozer (1994) developed a system called CONCERT. CONCERT uses the back-propagation through time (BPTT) algorithm (Williams and Zipser 1988; Rumelhart, Hinton, and Williams 1986; Campolucci 1998) and is a fully connected network; each processing unit receives, in addition to the set of external inputs, $x_j(n)$, the output of all other processing units, including itself, at the last step $n-1$. CONCERT uses a novel representation of pitch, duration, and chord for input and output that has a phychological, musical basis. Mozer's careful analysis of the behavior of the network for each presented task includes comparisons showing the network is more general and concise than second and third-order probabilistic transition table approaches. CONCERT can learn and compose waltzes including the harmonic chord sequences and multiple phrases.

Our past work uses Todd's design as a basis for a two-phase learning system called CHIME (Franklin 2000; Franklin 2001) that, in phase 1, learns three 12-bar jazz melodies. Pitches are represented in the same type of localized represen-

tation as Todd used (1 bit dedicated to each possible pitch). An additional set of 12 inputs represents the current chord of the song. The 12 bits correspond to 12 chromatic pitches, 4 of which are 1 and 8 of which are 0. The 4 "on" pitches are the chord tones. In the second phase the output units are further trained to improvise jazz. A reinforcement value indicates, numerically, how good or bad the output is, as determined by a set of rules for local-in-time improvisation. This network learned to increase the reinforcement value over time, and an analysis of its improvisation shows that it not only generally heeds the improvisation rules but also employs parts of the original melodies learned in the first phase.

Our earlier work provided the basis and motivation for our current in-depth study of recurrent networks and their use with reinforcement learning for music. In the next section we describe the recurrent networks that we are using and the modifications that enable it to learn according to the temporal difference algorithm. Following that we describe its performance on two prediction problems. We follow this with a discussion of the next step, that of employing reinforcement learning for pitch generation.

2 Long Short-Term Memory (LSTM) with Temporal Difference

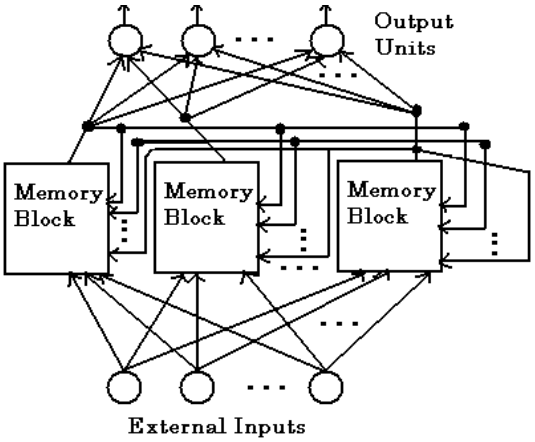

Figure 1: An LSTM network with recurrent memory blocks between the input layer and the output layer.

A Long Short-Term Memory or LSTM network (Hochreiter and Schmidhuber 1997, Gers et al. 2000) is a recurrent network that employs an internal layer of recurrent memory blocks that can be thought of as complex processing units as shown in Figure 1. Rather than being one typical unit that sums its weighted inputs and passes them through a nonlinear sigmoid function, each memory block contains several units used in different ways.

Eck and Schmidhuber (2002) use LSTM networks to learn and compose blues music. They successfully trained an LSTM network to learn a standard 12-bar sequence of blues chords. Similarly to Todd, they split time into eighth note increments, with one network iteration per eighth note time slice. The network must be able to output a chord value for as many as 8 time increments (for a whole note chord) and then output the next chord in the sequence. Each chord has a duration of either 8 or 4 time steps (whole note or half note durations). Chords are represented as sets of 3 or 4 (triads or triads plus the seventh) simultaneous note values. They use a second LSTM network to learn concatenations of 1-bar melodies over these learned chords.

Figure 2 shows a detailed view of memory block j with n self-recurrent linear memory cells. Each block also contains three gating units that are typical sigmoid units, but are used in the unusual way of controlling access to the memory cells. The output gate y^{out_j} learns to control when the cells' outputs are passed on, the input gate y^{in_j} learns to control when inputs are allowed to pass in to the cells, and the forget gate y^{ϕ_j} learns when to reset the memory cells. LSTM's designers were driven to design a network that could overcome the vanishing gradient problem (Hochreiter et al. 2001). Over time, as gradient information is passed backward to update weights whose values affect later outputs, the error/gradient information is continually decreased by weight update values that are typically less than one. Because of this, the gradient vanishes. Yet the presence of an input value way back in time may be the best predictor of a value far forward in time. LSTM's design overcomes these limitations.

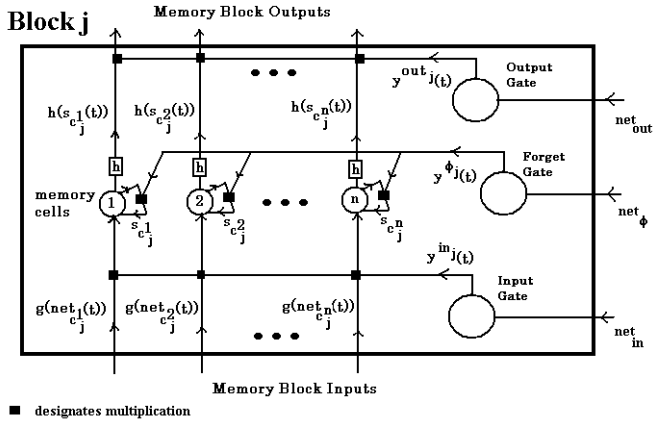

Figure 2: An LSTM memory block showing n memory cells and 3 gates. The memory block outputs (at top) are fed into the output layer and are also fed back recurrently to all of the memory blocks.

Referring again to Figure 2 and using the notation of Gers et al. (2000), c_j^v refers to the v^{th} cell of memory block j. The inputs to the blocks are multiplied by weights that belong to

the v^{th} cell, and then are summed, to form the $net_{c_j^v(t)}$ that are then passed through sigmoid function g, as shown at the bottom of Figure 2. The v^{th} memory cell's output is

$$s_{c_j^v}(t) = y^{\phi_j}(t) s_{c_j^v}(t-1) + y^{in_j}(t) g(net_{c_j^v(t)}) \quad (1)$$

where $s_{c_j^v}(0) = 0$. By its role as multiplier, the input gate $y^{in_j}(t)$ is gating the entrance of new inputs, $g(net_{c_j^v(t)})$ into the cell. With a sigmoid output, its value can swing between 0 and 1, allowing no access or complete access. Furthermore, the block's forget gate's output $y^{\phi_j}(t)$ is gating the cell's own access to itself, learning to reset the cell when information it is storing is no longer needed.

The cell's output $s_{c_j^v}(t)$ is passed through a sigmoid function, h, with range [-1,1], and then it may be passed on as an output of the memory block according to

$$y^{c_j^v}(t) = y^{out_j}(t) h(s_{c_j^v}(t)) \quad (2)$$

where we see that the output gate may allow $h(s_{c_j^v}(t))$ to pass out of the memory block, or it may inhibit it, by multiplying by 0. $y^{out_j}(t)$ is a sigmoid function of a weighted sum of inputs $net_{out_j}(t)$. I.e. $y^{out_j}(t) = f(net_{out_j}(t))$. The inputs are received via recurrent links from the memory blocks and from the external inputs to the network. Note, $y^{\phi_j}(t) = f(net_{\phi_j}(t))$ and $y^{in_j}(t) = f(net_{in_j}(t))$.

The weight updates for the LSTM network are complex because of the use of the memory cells and the three gates within each block. Each output unit has a set of weights used to multiply the values coming from the memory blocks. Each gate has a set of weights that it uses to multiply its inputs. Each cell has its own set of weights $w_{c_j^v m}$ used to calculate $net_{c_j^v}(t)$. In the usual neural network scenario, the network learns to minimize a function $E(t)$ that is the sum of the squared output unit errors. Each output unit error is $e^k(t) = T^k(t) - y^k(t)$ where $T^k(t)$ is the target of output unit k at time t and $y^k(t)$ is the actual output. i.e. $E(t) = \sum_k ((T^k(t) - y^k(t))^2)$. Then any weight w_{lm} is changed at each iteration by

$$\Delta w_{lm}(t) = -\alpha \frac{\partial E(t)}{\partial w_{lm}(t)} \quad (3)$$

moving the weight in the direction of a minimal value in the error surface. If there is just one output unit with output y,

$$\Delta w_{lm}(t) = \alpha e(t) \frac{\partial y}{\partial w_{lm}} \quad (4)$$

In temporal difference (TD) learning the one output at each time step is a prediction $V(t)$. The reinforcement value it is predicting may not be known until the end of the whole sequence of pitches. TD approximates the error $e(t)$ by first getting the next state $s(t+1)$ and using a temporal difference

$$e(t) = (R + \gamma V(s(t+1)) - V(s(t)). \quad (5)$$

In our case R is very sparse, having the value 0 until the end of the pitch sequence, when it is finally either 0 or 1, indicating "bad" or "good". γ is a discount factor, $0 < \gamma < 1$.

TD also uses an eligibility trace that "correlates" a weight's recent values with its effect on the output V, determining how "eligible" it is for weight updating. We follow Bakker's derivation of adding eligibility traces to an LSTM network (Bakker 2002). Bakker uses LSTM with advantage learning, a type of reinforcement learning. Advantage learning incorporates the TD eligibility traces in exactly the same way as TD.

Consider $w_{lm}(t)$ is the weight that connects input m to unit l, whatever that may be (cell, gate, output unit, etc.). Each change to weight w_{lm} is

$$\Delta w_{lm} = \alpha \frac{\partial E(t)}{\partial V(s(t)t} elig_{lm}(t) \quad (6)$$

where $\frac{\partial E(t)}{\partial V(s(t))}$ is the TD "error" $e(t)$ as given in equation (5), and $elig_{lm}(t)$ is the eligibility trace that indicates how eligible weight w_{lm} is to be updated. Its eligibility depends on how much it affects the output $V(s(t))$ and its recent cumulative effects on $V(s(t))$, with accumulation length determined by $\lambda, 0 < \lambda < 1$:

$$elig_{lm}(t) = \gamma \lambda elig_{lm}(t-1) + \frac{\partial V(s(t))}{\partial w_{lm}} \quad (7)$$

So, for every weight w_{lm} in the network, $\frac{\partial V(s(t))}{\partial w_{lm}}$ must be calculated.

The network's output is $V(s((t)) = f(net_O(t))$, where $net_O(t)$ is the weighted sum and f is the sigmoid function. The weights connecting memory block outputs to the network output, w_{Om}, are updated using equation (6):

$$\Delta w_{Om}(t) = \alpha\, e(t) elig_{Om}(t) \quad (8)$$

where

$$elig_{Om}(t) = \gamma \lambda elig_{Om}(t-1) + \frac{\partial V(s(t))}{\partial w_{Om}} \quad (9)$$

and $\frac{\partial V(s(t))}{\partial w_{Om}} = V'(s(t)) h_m(t) y^{out_j}(t)$. Each $h_m(t)$ is equal to $h(s_{c_j^v}(t))$ for some block j and some cell v in that block and $y^{out_j}(t)$ is the output gate's output for the same block j (refer again to Figure 2).

In each j^{th} block. the output of the output gate, $y^{out_j}(t) = f_{out_j}(net_{out_j}(t))$, multiplies every $h(s_{c_j^v}(t))$. For the output gate itself,

$$\Delta w_{out_j m} = \alpha \frac{\partial E(t)}{\partial V(t)} e_{out_j m}(t) \quad (10)$$

where

$$e_{out_j m}(t) = \gamma \lambda e_{out_j m}(t-1) + \frac{\partial V(t)}{\partial w_{out_j m}} \quad (11)$$

For our network configuration

$$\frac{\partial V(s(t))}{\partial w_{out_j m}} = \sum_{v=1}^{S_j} h(s_{c_j^v})(w_{Oc_j^v} V^{'}(s(t))) * f^{'}_{out_j}(net_{out_j}(t)) y^m(t-1)$$

where $x_m(t)$ is the m^{th} input to the output gate, and $w_{Oc_j^v}$ connects the cell output $h(s_{c_j^v}(t))$ to the TD output unit that predicts V. In other words the network output errors are propagated back to the j^{th} output gate, from the network output unit through the weights $w_{Oc_j^v}$ connecting all the cell outputs of block j to the output unit.

The errors for the input gate, forget gate, and cell input weights must all be propagated back through the weights $w_{Oc_j^v}$ as well, but also further back, through the memory cell. For this we calculate first three recurrent gradients:

$$\frac{\partial s_{c_j^v}(t)}{\partial w_{c_j^v m}} = \frac{\partial s_{c_j^v}(t-1)}{\partial w_{c_j^v m}} y^{\phi_j}(t) + g^{'}(net_{c_j^v}) y^{in_j}(t) x_m(t), \quad (12)$$

$$\frac{\partial s_{c_j^v}(t)}{\partial w_{\phi_j m}} = \frac{\partial s_{c_j^v}(t-1)}{\partial w_{\phi_j m}} y^{\phi_j}(t) + s_{c_j^v}(t-1) f^{'}_{\phi_j}(net_{\phi_j}(t)) x_m(t). \quad (13)$$

and

$$\frac{\partial s_{c_j^v}(t)}{\partial w_{in_j m}} = \frac{\partial s_{c_j^v}(t-1)}{\partial w_{in_j m}} y^{\phi_j}(t) + g(net_{c_j^v}(t)) f^{'}_{in_j}(net_{in_j}(t)) x_m(t).$$

Notice they all have the form

$$\frac{\partial s_{c_j^v}(t)}{\partial w_{lm}} = \frac{\partial s_{c_j^v}(t-1)}{\partial w_{lm}} y^{\phi_j}(t) + \delta_l(t) x_m(t). \quad (14)$$

The only recursive gradient equations are those involving the gradients of the cell outputs $s_{c_j^v}$. The crucial element that leads to this network's succcess is the ability of the memory cell to "cache" error/gradient information for later use, as can be seen in equations (1), (12), (13), and (14).

In order to update the weights on the inputs to the cells, $w_{c_j^v m}$, and the weights into the forget gate, $w_{\phi_j m}$, as well as the weights on the inputs to the input gate, $w_{in_j m}$, these gradients are used, along with the eligibility traces from the temporal difference method. Given n as the number of cells in block j:

$$\Delta w_{c_j^v m} = \alpha e(t) elig_{c_j^v m}(t) \quad (15)$$

for

$$elig_{c_j^v m}(t) = \gamma\lambda elig_{c_j^v m}(t-1) + w_{Oc_j^v} V^{'}(s(t)) y_{out_j}(t) h^{'}(s_{c_j^v}(t)) \frac{\partial S_{c_j^v}(t)}{\partial w_{c_j^v m}}$$

and for the input gate,

$$\Delta w_{in_j m} = \alpha e(t) elig_{in_j m}(t) \quad (16)$$

for

$$elig_{in_j m}(t) = \gamma\lambda elig_{in_j m}(t-1) + \sum_{v=1}^{n} w_{Oc_j^v} V^{'}(s(t)) y_{out_j} h^{'}(s_{c_j^v}(t)) \frac{\partial S_{c_j^v}(t)}{\partial w_{in_j m}}.$$

Finally, for the forget gates, the update equations are very similar to the input gate equations:

$$\Delta w_{\phi_j m} = \alpha e(t) elig_{\phi_j m}(t) \quad (17)$$

for

$$elig_{\phi_j m}(t) = \gamma\lambda elig_{\phi_j m}(t-1) + \sum_{v=1}^{n} w_{Oc_j^v} V^{'}(s(t)) y_{out_j} h^{'}(s_{c_j^v}(t)) \frac{\partial S_{c_j^v}(t)}{\partial w_{\phi_j m}}.$$

The reader is referred to Gers, Schmidhuber, and Cummins (2000) and Bakker (2002), for even more explicit detail, and to Sutton and Barto (1998) for their treatise on TD and value function learning.

3 Experiments

In the past, we have run experiments on jazz-oriented musical tasks. After trying several kinds of recurrent networks for this purpose, we found that LSTM networks provide the best results. We derived a new pitch representation called Circles of Thirds that works well for these tasks (Franklin 2004b). Pitches are represented as 7 bits, determined by their membership in the 4 circles of major thirds and the 3 circles of minor thirds. We also used LSTM networks to learn long songs, using the Circles of Thirds pitch representation and a new duration representation that enables learning of MIDI-based durations, recorded from human playing, rather than requiring exact score-based durations that do not vary (Franklin 2004a).

3.1 Predicting Success for Chromatic Lead-in

As Berg (1990) points out, one effective technique of improvisation is to use chord tones in creating a melody, and to lead-in to a chord tone with chromatic pitches just below or just above the chord tone. In our first prediction experiment, the network is given a set of 7 pairs of sequences, one sequence at a time. Each sequence contains five pitches. In the first of each sequence pair, the third note is a chromatic tie between the second note and the fourth note (except sequence pair 3 where it appears one step earlier). Both the fourth and

fifth notes are chord tones. The positive sequences, from Berg all occur over the Cma7 chord (with the 6th and 9th included as chord tones). There is no chord input however. The 7 pairs of sequences, each sequence labeled with its correct final target (1 or 0), are below, with '-' indicating a flat note:

g,a,a-,g,g	1	g,a,a,g,g	0
g,a,a#,b,b	1	g,a,a,b,b	0
d,d-,c,b,b	1	d,c,c,b,b	0
b,d,d#,e,e	1	b,d,d,e,e	0
e,f#,f,e,e	1	e,f#,f#,e,e	0
e,e,e#,f#,f#	1	e,e,e,f#,f#	0
a,b,b-,a,a	1	a,b,b,a,a	0

In an earlier paper we showed that sequences like these can be classified correctly using LSTM (Franklin 2004b). In classification, the network output is 0 at each time step, except the last, when the output is 1 if there is a chromatic lead-in, and 0 otherwise.

The next question is: Can the target value be predicted at the time when the significant chromatic note occurs? We employed the combination of temporal difference and LSTM recurrent network algorithm as described above, and used our Circles of Thirds pitch representation. In the best of the experiments, the LSTM network contains 15 blocks with 2 memory cells each. Learning occurs over 20000 epochs, with an epoch being one presentation of all of the sequences. $\lambda = 0.65$ and $\gamma = .95$. The learning rate for the output unit is $\alpha = .5$ and it is lower for the rest of the units, $\alpha = .2$. The network is as depicted in Figure 1 except that there is also a direct connection from the inputs, the state $s(t)$ that is the current pitch, to the single TD output unit that outputs the prediction $V(s(t))$ of the target. The LSTM bias factor, that determines how long before each block is used for learning (Gers, Schmidhuber, and Cummins 2000) is -.1.

Here are the predictions for each sequence:

1)	g,a,a-,g,g	.02,.08,.01,0,.99
	g,a,a,g,g	.02,.08,.01,0,0
2)	g,a,a#,b,b	.02,.08,.01,.95, 1.0
	g,a,a,b,b	.02,.08,.01,0, 0
3)	d,d-,c,b,b	0,.61,.86,.1.0,1.0
	d,c,c,b,b	0,0,0,0,0
4)	b,d,d#,e,e	0,0,0,.88,.99
	b,d,d,e,e	0,0,0,0,0
5)	e,f#,f,e,e	.05,.1,0,.11,.91
	e,f#,f#,e,e	.05,.1,0,0,0
6)	e,e,e#,f#,f#	.05,.03,.05,.95,1
	e,e,e,f#,f#	.05,.03,.02,.03,0
7)	a,b,b-,a,a	.01,0,0,1,1
	a,b,b,a,a	.01,0,0,0,0

In sequences 2, 4, 6, and 7, the prediction was clear at the fourth and fifth steps. All sequences were correctly predicted by the fifth step. When two pairs of sequences are similar (sequence pairs 1 and 2, and 5 and 6), positive prediction is delayed. In sequence pair 3 the chromatic change occurs one step earlier than in the others and the prediction reflects that; in the third step, the prediction was .86, then 1, then 1 on the fourth and fifth steps, after the d- to c.

3.2 ii-V7-I

In the second set of experiments we refer to the target as the reinforcement value, anticipating future experiments in using the prediction for reinforcement learning of pitch sequences. In these experiments, the network is presented with example riffs that are longer, carried out over a three-bar ii-V7-I sequence. The riffs are 12 pitches in total, all of the same duration. The examples come in pairs. The first one is a positive example, with the final reinforcement value, R, being 1 (and all 11 reinforcement values prior to that are 0). It is a positive example because the fourth, eighth, and twelfth notes in the pitch sequence are the tonic note of the final chord. All the other notes in the sequence are chosen from a scale that can be played over the current chord. The second sequence of the pair is the same sequence except the fourth, eighth, and twelfth notes are any of the scale notes except the tonic or fifth of the last chord.

Inputs are the current pitch, plus the current chord of the possible three, from a ii-V7-I sequence, held for four pitches each. The network also has two inputs that are beat inputs. One input is 1 at the first beat of each of the three bars. The second input is 1 on each of the two off-beats of the three bars.

For all examples, the number of epochs is 10000. 20 blocks with 2 cells in each block are used in the LSTM network. There is one output unit, a TD unit. The TD learning rate α is reduced to .1 in the ii-V7-I experiments and for all other units, α is .5. Again, the TD discounting factor $\gamma = .95$ and the bias factor for LSTM is -.1.

We retain the direct connection from inputs to output unit (experiments were not successful otherwise). In the original LSTM, there was no forget gate. It was introduced later by Gers, Schmidhuber, and Cummins (2000) for use when there is no clear demarcation of sequences. Wondering if it might interfere with the TD mechanism, we eliminated it by setting $y^{\phi_j}(t) = 1$ for all t. Suprisingly, the forget gate seems necessary for TD to work with LSTM. Without the forget gate, 0 was predicted for nearly all sequence steps.

We found that the value of λ made quite a difference in prediction values, with even a .05 change in its value producing significant results. We present two tables of results, for the values of $\lambda = .75$, and $.8$. We refer to these tables in the discussion following them.

For $\lambda = .75$

#	R	Tonic	Chord	Pitches				
1	1	g	A-7	a, f#, g, g	0	.8	0	0
			D7	c, a, c, g	.09	0	.05	.61
			G	g, e, f#, g	.81	1	.95	.98
	0	a	A-7	a, f#, g, c	0	.82	0	0
			D7	c, a, c, f#	.1	0	.04	.65
			G	g, e, f#, f#	0	0	0	0
2	1	d#	F-	g#, f, g#, d#	0	0	0	.01
			Bb7	f, c, g, d#	.01	.5	0	.41
			Eb	d, d#, a#, d#	.94	.02	.96	1
	0	d#	F-	g#, f, g#, g#	0	0	0	.01
			Bb7	f, c, g, g#	0	0	0	0
			Eb	d, d#, a#, d	.04	0	0	.07
3	1	b	C#-	f#, a#, e, b	.03	.28	.22	.23
			F#7	a#, a#, a#, b	.84	.07	.76	1
			B	e, b, e, b	.84	.54	.87	.85
	0	b	C#-	f#, a#, e, g#	.03	.34	.29	0
			F#7	a#, a#, a#, c#	.05	0	0	0
			B	e, b, e, c#	.87	0	0	0
4	1	c	D	c, e, g, c	0	.01	0	0
			G7	b, a, d, c	.02	0	.02	.05
			C	f, b, d, c	.01	.35	.93	1
	0	c	D	c, e, g, f	0	.01	0	0
			G7	b, a, d, a	.01	0	0	0
			C	f, b, d, e	0	.01	.05	.01
5	1	a#	C-	d, c, d, a#	0	0	0	0
			F7	g, d#, a#, a#	0	0	0	.19
			Bb	c, d#, c, a#	.19	.18	.22	.99
	0	a#	C-	d, c, d, c	0	0	0	0
			F7	g, d#, a#, d	0	0	.03	0
			Bb	c, d#, c, d#	.18	.16	.23	.18
6	1	g#	Bb-	f, c#, g#, g#	0	0	0	.05
			Eb7	g#, c#, c#, g#	.01	0	0	0
			Ab	c#, f, c#, g#	.01	.01	.85	1
	0	g#	Bb-	f, c#, g#, a#	0	0	0	.01
			Eb7	g#, c#, c#, c#	.01	0	0	0
			Ab	c#, f, c#, a#	0	0	0	0

For $\lambda = .8$

#	R	Tonic	Chord	Pitches				
1	1	g	A-7	c, d, f#, g	.99	.84	.86	.04
			D7	g, d, d, g	0	.25	.81	.99
			G	e, e, b, g	1	1	.85	.97
	0	g	A-7	c, d, f#, a	.99	.84	.86	.07
			D7	g, d, d, b	0	.02	.97	.9
			G	e, e, b, c	.88	.99	.92	.1
2	1	d#	F-	g, c, a#, d#	.99	1	.8	.22
			Bb7	d, g#, d, d#	.95	.91	.98	.92
			Eb	g#, g, c, d#	.97	.97	.92	.99
	0	d#	F-	g, c, a#, g	.99	1	.81	.09
			Bb7	d, g#, d, g#	.95	.9	.97	.99
			Eb	g#, g, c, g	1	.9	.93	.06
3	1	b	C#-	c#, c#, c#, b	.96	.95	.54	.24
			F#7	a#, f#, f#, b	.94	.95	.92	.89
			B	f#, a#, a#, b	1	1	.96	.98
	0	b	C#-	c#, c#, c#, d#	.96	.95	.5	.02
			F#7	a#, f#, f#, e	.95	.95	.93	1
			B	f#, a#, a#, c#	.94	.93	.91	0
4	1	c	D	b, a, f, c	1	.9	1	1
			G7	b, c, e, c	.98	.99	.93	.9
			C	b, d, g, c	.97	.78	.96	.98
	0	c	D	b, a, f, b	1	.9	1	.92
			G7	b, c, e, a	.98	1	.92	.47
			C	b, d, g, f	.98	.29	.07	.03
5	1	a#	C-	a#, d#, c, a#	.93	.91	.96	.01
			F7	g, d#, a, a#	.93	.68	.95	.89
			Bb	a#, g, d#, a#	1	.99	.93	.96
	0	a#	C-	a#, d#, c, d#	.93	.91	.96	.03
			F7	g, d#, a, d#	.97	.97	1	.89
			Bb	a#, g, d#, d#	1	.99	.87	.02
6	1	g#	Bb-	f, a#, f, g#	1	.9	.99	.91
			Eb7	c, c#, c#, g#	1	.73	.76	.16
			Ab	a#, c, g, g#	.18	.64	.19	.89
	0	g#	Bb-	f, a#, f, c	1	.91	.99	.99
			Eb7	c, c#, c#, a#	1	.81	.71	.01
			Ab	a#, c, g, c	.22	.54	.17	.01

Sequence pairs are numbered in the first column. The final target reinforcement value R for each sequence appears in the second column. The tonic of the third chord is shown in the third column. The current chord of the progression is in the fourth column, and the four pitches played over it are in the fifth column. The fourth pitch will always be the tonic of the third column, when R=1.

When $\lambda = .75$ (table above) sequence pairs 2, and 3 show the desired behavior of predicting low reinforcement from time steps 4 onward for the sequences without the tonics (although there is an anomaly of .87 in the R=0 sequence of pair number 3). The other pairs show good prediction several steps from the end of the sequence, to varying degrees. When $\lambda = .85$ (table not shown), all predictions are high except the very last which is always correct. So it produces a classifier, rather than a predictor. A table for $\lambda = .8$ is shown next:

For $\lambda = .8$ most notes in both sequences are judged "good" and nicely, the fourth, eighth, and twelfth notes receive different kinds of predictions than the rest. The twelfth (last) note is always correct, but the network does not always make the right prediction on the fourth and eighth notes.

3.3 Discussion

In its most successful behavior this combined TD-LSTM network seems to pick out the notes that produce the final value of R. And at the end of the sequence, as we saw in the chromatic lead-in experiments, the last several values correctly predict the final reinforcement. One overall impression we have of the LSTM network is that it is very adept at learning particular sequences, but may find it hard to ignore values in a sequence. Comparing the two tables with only a .05 difference in λ (.75 vs. .8), the prediction values are much higher, earlier on in the sequences for the higher value of λ,

whether the value of R is 0 or 1. In the second table, in many cases the prediction on the fourth or eighth step is 0 or 1, a sign that the network has traced back to that step as important in determining reward. But it is not always consistent. Also, there is obvious interference with a chord tonic appearing in other sequences and affecting the predictions there. Before we move on to the next step, we will try using three smaller LSTM networks for this task, one each for the three chords in the progression. It seems obvious that these networks could easily learn to classify the pitches over an individual chord. A fourth network could learn to use these classifications to predict the final reward.

4 Next Step

After refining the prediction behavior of the LSTM-TD network(s), the next question to ask is: Can a reinforcement learning algorithm learn to use this prediction to choose valid pitches? First note that Bakker (2002) combined a form of reinforcement learning called advantage learning with an LSTM network. The LSTMRL network learned to solve 2 classic non-Markov tasks (ball balancing and T-maze following). One network was used, with one output per possible action. The network is given the state as input, and its output is the value of the corresponding action, in the given state. The action is chosen by finding the output with the highest value.

There are two problems with this configuration for musical tasks. First, we have shown that for three tasks, the LSTM network learns faster, and better, with a state (pitch or duration) representation that has more than one value "on" at a time. For example, C is represented in the Circles of Thirds representation as 1000100. And in our duration representation, not only are there many more allowable actions, but the representation does not even have a fixed number of "on" bits.

To solve these problems we hark back to early ideas and architectures that have not remained popular currently. In particular, we will use the RL architecture of Jordan and Jacobs (1986). In this and other architectures, the value and action function are learned separately. The large number of epochs required to learn prediction make make it inffeasible to use these networks in interactive learning in real-time with humans. However, the networks could be trained first and then courld learn further through a second phase that does involve interaction (Franklin 2001).

References

Bakker, B. (2002). Reinforcement learning with long short-term memory. In T. Dietterich, S. Becker, and Z. Ghahramani (Eds.), *Neural Information Processing Systems, 14*. MIT Press, Cambridge, MA.

Berg, S. (1990). *Jazz Improvisation: the Goal-Note Method, Second Ed.* Delevan, NY: Kendor Music, Inc.

Campolucci, P. (1998). *A Circuit Theory Approach to Recurrent Neural Network Architectures and Learning Methods*. Universita Degli Studi di Bologna: Dottorato di Ricerca in Ingegneria Elettrotecnica.

Dannenberg, R., B. Thom, and D. Watson (1997). A machine learning approach to musical style recognition. In *Proceedings of International Computer Music Conference*. ICMA.

Eck, D. and J. Schmidhuber (2002). Learning the long-term structure of the blues. In *Proceedings of International Conference on Artificial Neural Networks*, pp. 284–289. ICANN.

Franklin, J. A. (2000). Multi-phase learning for jazz improvisation and interaction. In *Proc. 8th Biennial Connecticut College Symposium on Arts and Technology*. CT College, New London CT.

Franklin, J. A. (2001). Learning and improvisation. In T. Dietterich, S. Becker, and Z. Ghahramani (Eds.), *Neural Information Processing Systems 14*. MIT Press, Cambridge, MA.

Franklin, J. A. (2004a). Computational models for learning pitch and duration using lstm recurrent neural networks. In *Proc. of the International Conference on Musical Perception and Cognition 8*.

Franklin, J. A. (2004b). Recurrent neural networks and pitch representations for music tasks. In *Proc. of the Florida Artificial Intelligence Research Symposium (FLAIRS) 2004*.

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

Griffith, N. G. and P. M. Todd (1999). *Musical Networks: Parallel Distributed Perception and Performance*. Cambridge, Massachusetts: The MIT Press.

Jordan, M. (1986). Attractor dynamics and parallelism in a connectionist sequential machine. In *Proc. Eighth Annual Conf. of the Cognitive Science Society, Amherst, MA*.

Laden, B. and D. H. Keefe (1991). The representation of pitch in a neural net model of chord classification. In P. Todd and E. Loy (Eds.), *Music and Connectionism*, Cambridge, Massachusetts. The MIT Press.

Mozer, M. C. (1994). Neural network music composition by prediction: Exploring the benefits of psychophysical constraints and multiscale processing. *Connection Science 6*, 247–280.

Rumelhart, D., G. Hinton, and R. Williams (1986). Learning internal representations by error propagation. *Parallel Distributed Processing 1*, 318–362.

Sutton, R. S. and A. G. Barto (1998). *Reinforcement Learning*. Cambridge, Massachusetts: The MIT Press.

Tesauro, G. J. (1994). Td-gammon, a self-teaching backgammon program, achieves master-level play. *Neural Computation 6*(2), 215–219.

Todd, P. M. (1991). A connectionist approach to algorithmic composition. *Music and Connectionism*.

Todd, P. M. and E. D. Loy (1991). *Music and Connectionism*. Cambridge, Massachusetts: The MIT Press.

Williams, R. J. and D. Zipser (1988). *A learning algorithm for continually running fully recurrent networks*. San Diego, La Jolla, CA: Tech Report ICS-8805, Univ of Calif.