Learning to Create Jazz Melodies Using Deep Belief Nets

Greg Bickerman[1], Sam Bosley[2], Peter Swire[3], Robert M. Keller[1]

[1] Harvey Mudd College, Claremont, CA, USA
[2] Stanford University, Stanford, CA, USA
[3] Brandeis University, Waltham, MA, USA
gbickerman@hmc.edu, sbosley@stanford.edu,
swirepe@brandeis.edu, keller@cs.hmc.edu

Abstract. We describe an unsupervised learning technique to facilitate automated creation of jazz melodic improvisation over chord sequences. Specifically we demonstrate training an artificial improvisation algorithm based on unsupervised learning using *deep belief nets*, a form of probabilistic neural network based on restricted Boltzmann machines. We present a musical encoding scheme and specifics of a learning and creational method. Our approach creates novel jazz licks, albeit not yet in real-time. The present work should be regarded as a feasibility study to determine whether such networks could be used at all. We do not claim superiority of this approach for pragmatically creating jazz.

1 Introduction

Jazz musicians strive for innovation and novelty in creating melodic lines, in the context of chord progressions. Because of the structural characteristics of typical chord progressions, it is plausible that a machine could be taught to emulate human jazz improvisation. To this end, one might explicitly state the rules for jazz improvisation, e.g. in the form of grammars [1]-[2]. But structural rules may risk losing some of the flexibility and fluidity for which jazz is known. Here we try exploring a more organic approach: instead of teaching a machine *rules* for good jazz, we give the machine examples of the kind of melodies we want to hear stylistically, and let it determine for itself the features underlying those melodies, so that it can create similar ones.

Our current exposition concentrates on a single approach to learning, heretofore not applied to music creation as far as we are aware: *deep belief networks* (DBNs), a multi-layered composition of restricted Boltzmann machines (RBMs), a specific type of stochastic neural network. We focus on the creation of melodies, and do not attempt to tackle broader issues of real-time collaborative improvisation. In other words, our work tries to explore the application of a specific neural net technology, as opposed to trying solving the general problem of creating an improvising agent by any means necessary. At present, our learning method is necessarily off-line due to a fairly slow training method, but we hope this can be improved in the future.

We were attracted DBNs based on recent expositions of Hinton, et al. [3]-[7]. Such machines learn to recognize by attempting to create examples (in the form of bit vectors), comparing those examples to training examples, and adjusting their parameters to produce examples closer to the given examples, a form of unsupervised learning. This seemed to us to be very similar to the way some humans learn to improvise melodies by *emulation*. Although the *stochastic* nature of DBNs might be considered a liability in some application fields, we try to leverage that nature to achieve novelty in our generated melodies, a characteristic of the creativity required for jazz improvisation. Thus our objective is different than that of Hinton; we want to *create* interesting melodies and are less concerned about their recognition.

2 Restricted Boltzmann Machines

A *restricted Boltzmann machine* (RBM) is a type of neural network introduced by Smolensky [8] and further developed by Hinton, et al. [3]-[7]. It consists of two layers of neurons: a visible layer and a hidden layer. Each visible neuron is connected to each hidden neuron, and vice versa, through a series of symmetric, bi-directional weights.

A single training cycle for the machine takes a binary data vector as input, activating its visible neurons to match the input data. It then alternates activating its hidden nodes based on its visible nodes, and activating its visible nodes based on its hidden nodes. Each node is activated probabilistically based on a weighted sum of all nodes connected to it. Since nodes within a layer are not connected to each other, activation of the hidden nodes depends only on the states of the visible nodes, and vice versa. After the network has stabilized, the new configuration of visible nodes can be viewed as output.

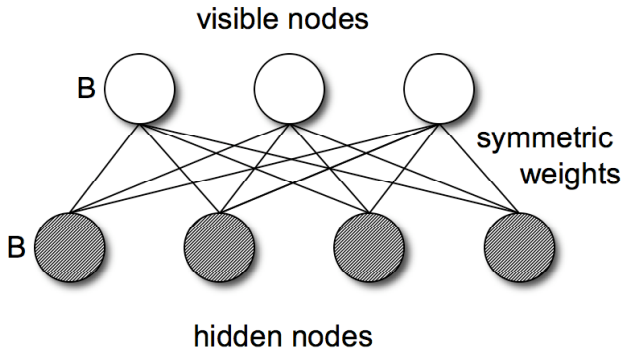


Figure 1: A restricted Boltzmann machine.
The first node B of each layer is a fixed *bias* node.

The objective of an RBM is to learn features in sets of data sequences. Toward this end, we implemented the *contrastive divergence* (CD) learning algorithm, as described by Hinton [3]. We modeled our implementation on an excellent tutorial supplied by Radev [9]. The CD algorithm allows for relatively inexpensive training

given the large number of nodes and weights in our networks. Once trained, an RBM can take a random data sequence and, through a series of activations, generate a new sequence that emulates features from the training data.

While a single RBM is capable of learning some patterns in the training data, multiple RBMs can be layered together to form a much more powerful machine known as a *deep belief network* (DBN) [4]. Multiple RBMs are combined by identifying the hidden layer of each RBM with the visible layer of the one below. The second RBM is able to learn features about the features learned by the first RBM, and thus, the entire layered machine should be able to learn far more intricate patterns than a single RBM could. Figure 2 illustrates the structure of a DBN.

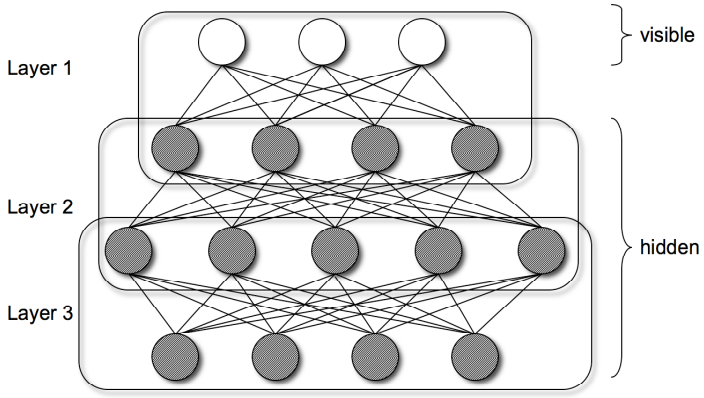


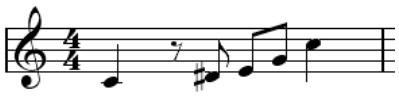
Figure 2: An illustration of a 3-layer Deep Belief Network

3 Data Representation

In order to train DBNs on musical data, we first encode the music as *bit vectors*. We divide each beat into beat subdivisions called *slots*, with the number of slots dependent on the smallest note duration to be represented. For our experiments, we chose twelve slots per beat, which allows us to represent all duplet or triplet note durations down to a sixteenth note triplet.

Each slot is filled by a block of thirty bits, divided into twelve chord bits and eighteen melody bits. A description of the melody bits follows. Twelve bits are used as a *one-hot encoding* for the chromatic pitch classes from C to B over one octave, four bits are used as a second one-hot encoding to designate one of four octaves, one bit designates a sustained extension of the previous note, i.e. the note is not attacked anew, and one bit represents a rest. If a note is being attacked at a given slot, its corresponding pitch and octave bits are on and all other bits are off. If a note is being sustained, then the pitch bits are ignored but the sustain bit is on. Representing octaves this way rather than using a single one-hot encoding to represent a four-octave chromatic range, gave us a significant improvement in training time, by reducing the number of pitch nodes in the input layer.

The sustained note bit is used to represent the same pitch value as the note previously played. Thus notes of long duration will be seen as *chains* of sustain bits being on. Figure 3 shows an example of a melody and its corresponding encoding at a coarser resolution of two slots per beat for brevity.



Beat	Auxiliary		Chromatic Pitch within Octave												Octave			
	Sustain	Rest	C	C#	D	D#	E	F	F#	G	G#	A	A#	B	1	2	3	4
1			1													1		
&	1																	
2		1														1		
&						1												
3							1									1		
&										1						1		
4			1														1	
&	1																	

Figure 3: A short melodic segment with a coarse encoding (only two slots per beat) To improve readability, 0 values are left blank.

Each chord is encoded as twelve bits representing the chromatic pitches from C to B. If a pitch is present in a chord, its corresponding bit is on. Melody and chord vectors are concatenated to form part of the input to the network corresponding to one slot. Thus the machine ideally learns to associate specific chords with various melodic features. Because the machine will be seeing more than one slot at a time, as we later describe, it can also learn about chord transitions.

4 Training Data

We initially trained on a small set of children's melodies such as "Twinkle, Twinkle, Little Star" and "Frère Jacques." These melodies were all in the same key and generally consisted of simple rhythms and notes that were in their respective chords. Once we taught a machine to learn from, and then create, similarly simple melodies, we moved on to teaching larger networks jazz.

Our primary dataset was a large corpus of 4-bar jazz *licks* (short coherent melodies) cycling over the common ii-V-I-VI^7 "turnaround" chord progression in a single key. The ii-V-I is a very common cadence in jazz; the VI^7 chord is a connecting chord that leads one lick into the next for the same progression, VI^7 being the dominant relative to the ii chord that follows. Most of the licks were either transcribed from notable jazz solos, or hand constructed, some with the help of the grammar-based "lick generator" of the Impro-Visor software tool [10].

5 Learning Method

Part of our goal is for the machine to learn how to create melodies that transition between chords in a progression. To add flexibility, rather than training our machine on inputs of all 4 bars of a lick at once, we break our data up into smaller windows of 1 measure each. For each 4 bar lick, we start the "window" at the beginning of the first bar. Then we move the window forward by one beat and look at the next 4 beats starting at beat 2 of the measure for the next window. We move the window forward by a beat at a time, taking measure-long snapshots of the window, until we reach the end of the 4-bar lick. In this way a single 4-bar lick is broken up into 13 overlapping shorter windows that are used sequentially as the inputs to the network. The scenario is analogous to that shown in Figure 4, except there are no question marks during training.

For creating new melodies, we start the machine with a "seed" consisting of specified chord bits defining our desired chord progression, and random melody input bits. The chord bits in the first layer of the machine are *clamped* so that, during any given creation cycle, they cannot be modified by the stochastic nature of the machine.

In creating a new melody, we use a procedure analogous to windowing during training. We start by generating the first few beats of a new melody and then clamping their corresponding bits. As each successive beat is generated, the whole melody and chord sequence is shifted forward to make room for the next beat. So in general, the machine only generates one beat at a time, but uses clamped chords and clamped beats of the preceding melody to influence the note choices. This process is illustrated in Figure 4.

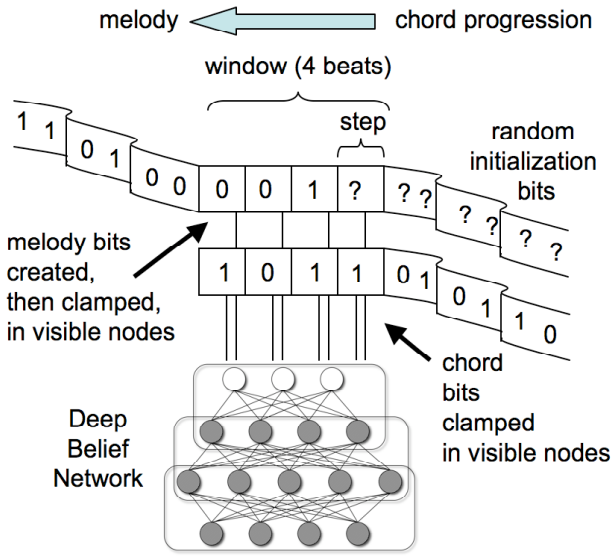


Figure 4: An illustration of the process of windowed generation.
The RBM generates small segments of melody over a fixed chord seed.
A newly generated segment is then fixed and used to generate the next segment of melody.

During the machine's final activation of its visible layer (which constitutes the newly generated melody), we group certain bits together for special consideration. Rather than letting the machine activate every bit probabilistically, we look at each slot individually and activate only the pitch bit and octave bit with the highest probabilities of activation among their group. Thus the machine is forced to choose whether to sustain, rest, or start a new pitch. We found that this approach allows for good variety of created melodies, while still resonating well with the given chords.

We also want to know if the machine can learn to create licks over a ii-V-I-VI^7 chord progression in an *arbitrary* key. Thus, we included the option to transpose each input into different keys and train on the transpositions simultaneously.

We implemented all of the functionality described thus far as a stand-alone tool we call "RBM-provisor" that we have made publicly available [11]. The tool is written in Java and supports input and output via the *leadsheet* format [12] used by Impro-Visor, so that the user can work with readable, symbolic encodings, rather than bit-vectors.

6 Results

Our initial experiments used our dataset of short segments of children's melodies, training on small 2-layer machines for 100 epochs. Results were encouraging, with chosen notes fitting well into the simple chords and flowing together melodically. Figure 5 shows a children's melody created over a simple chord progression. After achieving the ability to create stylistically similar melodies from a set of simple examples, we moved on to the more complex problem of learning jazz.

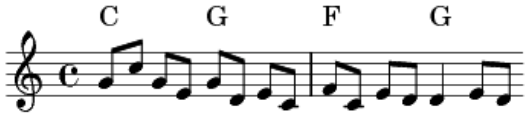


Figure 5: An example of a created children's melody over a specified chord progression.

In attempting to produce a successful jazz creation network, we experimented with various aspects of networks, including number of layers, number of nodes per layer, number of training epochs, and many others. We ultimately settled on a 3-layer network containing 1441 input nodes (4 beats x 12 slots per beat x 30 bits per slot + 1 bias), with 750, 375 and 200 hidden nodes respectively. A typical training involved 250 epochs on about 100 four-measure licks, which takes about nine hours on an inexpensive desktop computer.

The first stave of Figure 6 shows a sample of training data, with the second stave showing a typical lick created by the network. For comparison, the third stave shows random notes at the same resolution of 12 slots per beat. When analyzing the created music using Impro-Visor [10], we found the vast majority of generated notes were in the chord, with occasional color tones (tones not in the chord, but sonorous with it),

which is totally acceptable. Foreign tones were hardly ever present. Created melodies tended to avoid large interval jumps and rarely skipped octaves.

Additionally, we found the training method was able to deal well with transpositions. After training on four copies of each of our inputs, transposed up 0, 1, 2, and 3 semitones from the original, the machine still created chord-compatible music regardless of the set of chords that was provided as a seed. We have yet to test jazz generation on more than four transpositions due to the extensive added training time required for transposing inputs to all twelve keys. Nonetheless, we are optimistic regarding our machine's ability to handle any number of transpositions, given sufficient nodes and adequate training time.

The reason that ability to transpose is viewed as important is that, in jazz music, the chord progressions often have implied abrupt key changes that are not labeled as such explicitly. Ideally, an improvisational algorithm would be able to respond to chord changes based on the chords in whatever relative transpositions they occur, rather relative to a fixed reference key. For example, in the standard tune "Satin Doll", one finds an extended cadence Am7 D7 Abm7 Db7 C. The sub-progression Abm7 Db7 is the same as Am7 D7 transposed down a half-step. It would be more economical and modular to train a network on all transpositions of Am7 D7 that it would be to train it on all contexts that might surround that two-chord sequence.

We noticed some differences between input data and generated music. While half-step intervals were common in our inputs, generated licks tended to avoid them – skirting off-chord approach tones and opting instead for more familiar chord tones. The most striking difference between the two sets of music related to rhythms. While our inputs contained notes of duplet and triplet rhythms, our outputs contained almost exclusively duplet rhythms. This issue will be discussed in greater detail in the next section.

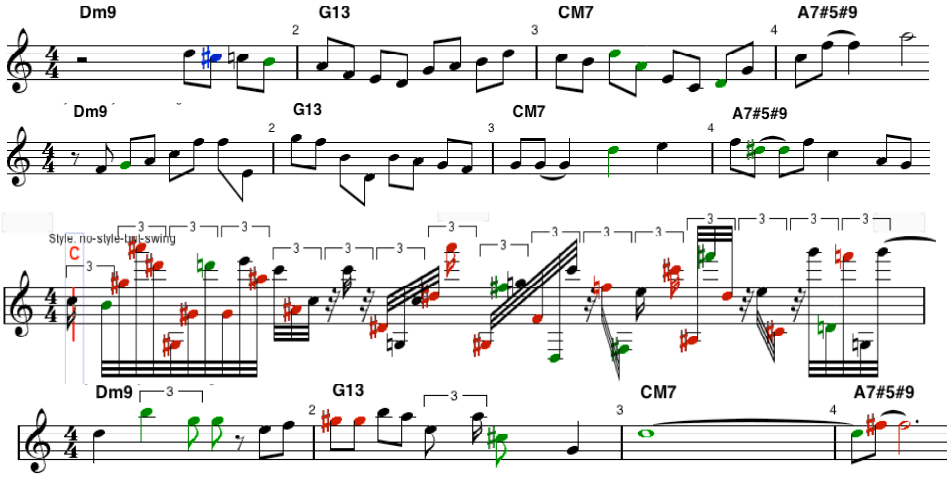


Figure 6: The first stave is a sample from our training data licks.
The second stave is a lick that was generated by a trained deep belief network.
The third stave shows random notes generated at the resolution of the network.
The fourth stave shows incoherence using selection not based on maximum probability.
In all cases, red notes represent discords.

Other approaches tried included selecting bits proportional to the neuron probability distribution, rather than always choosing the maximum probability. However, this produced melodies that were more disjointed and less coherent rhythmically, as in the bottom stave of Figure 6. We also experimented with encodings that included beat information, such as which beats were stronger. The results for such encodings were not superior to those for the chosen encoding presented here.

At this juncture, using deep belief networks would not be our first choice for a lick generator in a jazz education tool such as Impro-Visor [10]. The quality of licks generated by Impro-Visor's grammatical approach is sufficiently superior qualitatively to those generated by our DBN that it would be pointless to conduct a third-party blindfold test. The other drawback to DBNs is the large training time. On the other hand, DBN's may eventually prove to be less algorithmically biased than an unsupervised approach such as that in [2], which relies on clustering and Markov chains, and it is possible that the training time issue can be alleviated.

7 Future Work

The successes of our initial deep-belief improvisor are encouraging, but there is still much potential for improvement. Despite training on inputs containing both triplet and duplet rhythm patterns, our machine created mostly duplet rhythm patterns. We hypothesize that this results from a predominance of duplet rhythms in our training set, overshadowing the examples of triplet rhythms. Ideally, our machine should be able to generate triplet patterns at a lower frequency than duplet patterns, rather than excluding them from generation altogether. It is possible that a different note generation rule might yield more variety, but we have yet to find one that doesn't also result in less coherence.

Additionally, the music generated by our trained DBN tends to produce disproportionate numbers of repeated pitches, instances in which the same note is played twice in a row, compared with their relatively low frequency of occurrence in the training data. Repeated notes in jazz may tend to sound static and immobile, and we would like to avoid them if possible. One solution we implemented involved post-processing our generated music to merge all repeated notes. Ideally the machine should avoid producing as many of them in the first place. It is possible that a different encoding might resolve some of these issues.

Finally, we believe that our work naturally lends itself to the open problem of chord inference. Currently, we give our machine chords as input, and it creates a suitable melody. If we instead provide a melody as input, a DBN similar to ours might be able to determine one or more chord progressions that fit the melody.

8 Related Work

Geoffrey Hinton and his associates are responsible for much previous work related to restricted Boltzmann machines. They used RBMs and DBNs for various purposes, including handwritten digit recognition [3], facial recognition [7], and movie recommendation [6]. These contrast to our use, which is generation. A particularly useful tutorial for implementing an RBM has been written by Rossen Radev [9]. Our RBM implementation was largely influenced by these sources.

Early work on generation of music by neural networks includes Mozer [13], who used back propagation through time. See Todd and Loy [14] for other early examples. Bellgard and Tsang [15] used a different form of extended Boltzmann machine for the harmonization and analysis of chorales. Eck and Lapalme [16] describe an approach using LSTM (Long Short-Term Memory) neural networks. Additionally, Page [17] utilized neural networks for musical sequence recognition. Please see Todd and Werner [18] for a more extensive survey.

Various other approaches have been taken towards artificial composition. Biles [19] used genetic algorithms. Jazz generation using a grammar-based approach was demonstrated by Keller and Morrison [1], and learning by Gillick, Tang and Keller [3]. Please consult these papers for further references on related approaches. Please see Cope [20] for a broad survey of approaches to musical creativity, including neural networks.

9 Summary

The results of our experiments show that a deep belief network is capable of learning certain concepts about a set of jazz licks and in turn creating new melodies. The ability of a single machine to generate licks over a chord progression in several different keys demonstrates the power and flexibility of the approach and suggests that a machine could be taught to generate entire solos over more complex chord progressions given a sufficient dataset. While the licks created by our networks sometimes under-represented features of the training set, their novelty and choice of notes seem adequate to characterize them as jazz.

Despite a moderately-successful proof of concept, deep belief networks would not be our first choice for a *practical* lick-generation tool at this stage of our understanding. Our initial objective of exploring the possibility has been achieved, and further exploration is anticipated. We continue to be attracted to this approach as the basis for an algorithmically unbiased machine learning method.

Acknowledgment

This research was supported by grant 0753306 from the National Science Foundation. We are grateful to the anonymous referees for several helpful suggestions for revision and future work.

References

1. Keller, R. and Morrison, D.: A Grammatical Approach to Automatic Improvisation, In: Proceedings Fourth Sound and Music Computing Conference, Lefkada, Greece, July (2007)
2. Gillick, J., Tang, K., and Keller, R.: Learning Jazz Grammars, In: Proceedings Sixth Sound and Music Computing Conference, Porto, Portugal, pp. 125--130 (2009)
3. Hinton, G.E.: Training Products of Experts by Minimizing Contrastive Divergence Neural Computation, 14, 8, pp. 1771--1800 (2002)
4. Hinton, G. E., Osindero, S. and Teh, Y.: A Fast Learning Algorithm for Deep Belief Nets. Neural Computation, 18. pp. 1527--1554 (2006)
5. Hinton, G. E.: To Recognize Shapes, First Learn to Generate Images. In: Cisek, P., Drew, T. and Kalaska, J. (eds.) Computational Neuroscience: Theoretical Insights into Brain Function. Elsevier (2007)
6. Salakhutdinov, R. R, Mnih, A. and Hinton, G. E.: Restricted Boltzmann Machines for Collaborative Filtering. In: Proceedings International Conference on Machine Learning, Corvallis, Oregon (2007)
7. Susskind, J.M., Hinton, G., Movellan, J.R., and Anderson, A.K.: Generating Facial Expressions with Deep Belief Nets, In: Kordic, V. (ed.) Affective Computing, Emotion Modeling, Synthesis and Recognition, ARS Publishers (2008)
8. Smolensky, P.: Information processing in dynamical systems: Foundations of harmony theory. In: D. E. Rumelhart and J. L. McClelland, (eds.), Parallel Distributed Processing: Explorations in the Microstructure of Cognition. vol 1: Foundations. MIT Press (1986)
9. Radev, R.: Restricted Boltzmann Machine - Short Tutorial. iMonad Software. http://imonad.com/blog/2008/10/restricted-boltzmann-machine/ (2009)
10. Keller, R. et al.: Jazz Improvisation Advisor, http://www.impro-visor.com (2009)
11. RBM-provisor: https://sourceforge.net/projects/rbm-provisor/ (2009)
12. Keller, R.: Leadsheet notation, http://www.cs.hmc.edu/~keller/jazz/improvisor/LeadsheetNotation.pdf (2005)
13. Mozer, M.: Neural network music composition by prediction: Exploring the benefits of psychoacoustic constraints and multiscale processing, Connection Science, 6, 2-3, pp. 247-280 (1994).
14. Todd, P. and Loy, D.G (eds.).: Music and Connectionism, MIT Press, Cambridge, MA (1991)
15. Bellgard, M. and Tsang, C.: Harmonizing Music the Boltzmann Way. In: Griffith, N. and Todd, Peter M. (eds.), Musical Networks, MIT Press, Cambridge, MA, pp. 261--277 (1999)
16. Eck, D., and Laplme, J.: Learning Musical Structure Directly from Sequences of Music, Tech. Rept. 1300, Universite de Montreal DIRO (2008)
17. Page, M.: Modeling the Perception of Musical Sequences with Self-organizing Neural Networks. In: Connection Science, 6, pp 223--246 (1994)
18. Todd, P., and Werner, G.: Frankensteinian Methods for Evolutionary Music Composition. In: Griffith, N. and Todd, Peter M. (eds.), Musical Networks, MIT Press, Cambridge, MA, pp. 313--339 (1999)
19. Biles, J.: GenJam: A Genetic Algorithm for Generating Jazz Solos. In: Proceedings of the International Computer Music Association (1994)
20. Cope, M.: Computer Models of Musical Creativity, MIT Press, Cambridge, MA (2005)