ANALYZING DRUM PATTERNS USING CONDITIONAL DEEP BELIEF NETWORKS

Eric Battenberg
University of California, Berkeley
Dept. of Electrical Engineering and Computer Sciences
ericb@eecs.berkeley.edu

David Wessel
University of California, Berkeley
Center for New Music and Audio Technologies
wessel@cnmat.berkeley.edu

ABSTRACT

We present a system for the high-level analysis of beat-synchronous drum patterns to be used as part of a comprehensive rhythmic understanding system. We use a multi-layer neural network, which is greedily pre-trained layer-by-layer using restriced Boltzmann machines (RBMs), in order to model the contextual time-sequence information of a drum pattern. For the input layer of the network, we use a conditional RBM, which has been shown to be an effective generative model of multi-dimensional sequences. Subsequent layers of the neural network can be pre-trained as conditional or standard RBMs in order to learn higher-level rhythmic features. We show that this model can be fine-tuned in a discriminative manner to make accurate predictions about beat-measure alignment. The model generalizes well to multiple rhythmic styles due to the distributed state-space of the multi-layer neural network. In addition, the outputs of the discriminative network can serve as posterior probabilities over beat-alignment labels. These posterior probabilities can be used for Viterbi decoding in a hidden Markov model in order to maintain temporal continuity of the predicted information.

1. INTRODUCTION

Deep belief networks (DBNs) have shown promise in many discriminative tasks, such as written digit recognition [6] and speech recognition [8]. In addition, the generative nature of DBNs makes them especially well-suited for stochastic generation of images or sequences [5, 11].

In this paper, we apply DBNs to the analysis of drum patterns. The drum pattern analysis system presented here is to be part of a complete live drum understanding system, which is also composed of a drum detection front-end [1] and a low-level multi-hypothesis beat tracker. The goal of the drum understanding system is to go beyond simple beat tracking by providing additional high-level rhythmic information, such as time signature or style information, while being robust to expressive embellishments and dynamic song structure, such as tempo fluctuations or time signature changes. The pattern analysis system we present here can help achieve these goals, not only by providing the desired high-level information, but also by communicating with the low-level beat tracker to help it correct beat period and phase errors.

To demonstrate the effectiveness of this model, we focus on a specific discriminative task: identifying the alignment of beats within a measure. Measure alignment information is particularly important to high-level drum pattern analysis since each beat has a specific meaning depending upon the musical style. For example, song transitions typically occur on the first beat of a measure, and in rock music and relate styles, beats 2 and 4 typically feature an accented snare drum *back beat*.

Previous work on beat-measure alignment has focused on simple heuristic rules. In [7], Klapuri presents a beat tracker that determines beat-measure alignment by correlating multi-band onset patterns with two different back beat measure templates. In [3], Goto addresses beat alignment by detecting chord change locations and by alignment with 8 drum pattern templates. Approaches like these work well for the typical pop song but are ineffective when presented with exotic rhythm styles or many types of progressive music. To deal with these situations, rather than accruing a large list of hand-written heuristic rules, we can automatically encode a large amount of musical knowledge into the *distributed state-space* [2] of a deep belief network, which we introduce in the next section.

2. DEEP BELIEF NETWORKS

2.1 The Restricted Boltzmann Machine

The deep belief network is a probabilistic multi-layer neural network composed of *restricted Boltzmann machines*, or RBMs [2, 5]. The RBM, as shown in Figure 1, is a two layer probabilistic graphical model with undirected connections between visible layer units, v_i, and hidden layer units, h_j. The "restricted" part of the name points to the fact that there are no connections between units in the same layer. This allows the conditional distribution of the units of one layer given all the units of the other layer to be com-

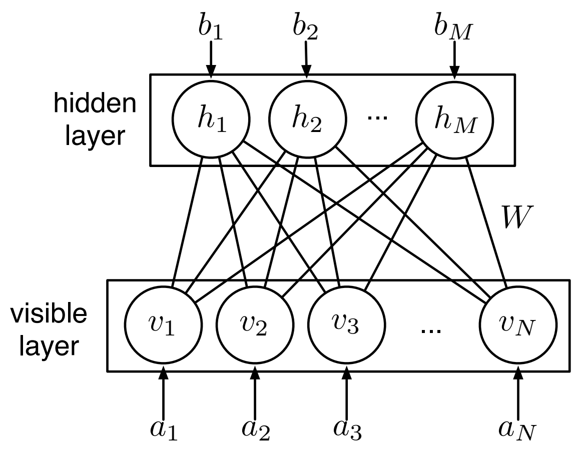

Figure 1. A restricted Boltzmann machine with N visible units and M hidden units.

pletely factorial, i.e.

$$P(\mathbf{v}|\mathbf{h}) = \prod_i P(v_i|\mathbf{h}) \tag{1}$$

$$P(\mathbf{h}|\mathbf{v}) = \prod_j P(h_j|\mathbf{v}) \tag{2}$$

The RBM is a probabilistic energy-based model, meaning the probability of a specific configuration of the visible and hidden units is proportional to the negative exponentiation of an energy function, $E(\mathbf{v}, \mathbf{h})$

$$P(\mathbf{v}, \mathbf{h}) = \frac{e^{-E(\mathbf{v},\mathbf{h})}}{Z} \tag{3}$$

Where $Z = \sum_{\mathbf{v},\mathbf{h}} \exp(-E(\mathbf{v}, \mathbf{h}))$ is a normalizing constant referred to as the *partition function*. Note that because Z is difficult to compute, it is typically intractable to compute the joint distribution $P(\mathbf{v}, \mathbf{h})$.

For binary-valued visible and hidden units, the energy function, $E(\mathbf{v}, \mathbf{h})$, can be written as:

$$E(\mathbf{v}, \mathbf{h}) = -\mathbf{a}^{\mathrm{T}}\mathbf{v} - \mathbf{b}^{\mathrm{T}}\mathbf{h} - \mathbf{v}^{\mathrm{T}}W\mathbf{h} \tag{4}$$

Where $\mathbf{a}$ and $\mathbf{b}$ are vectors containing the visible and hidden unit biases, respectively, and W is the weight matrix that connects the two layers.

The goal in training an RBM is to maximize the likelihood of the training data under the model, $P(\mathbf{v})$. The actual log-likelihood gradient is difficult to compute because it involves the intractable partition function Z; however, stochastic estimates of the gradient can be made by drawing Gibbs samples from the joint distribution $P(\mathbf{v}, \mathbf{h})$ using the factorial conditional distributions in (5),(6).

$$P(v_i = 1|\mathbf{h}) = \bar{\sigma}(a_i + \textstyle\sum_j W_{ij} h_j) \tag{5}$$

$$P(h_j = 1|\mathbf{v}) = \bar{\sigma}(b_j + \textstyle\sum_i W_{ij} v_i) \tag{6}$$

Where $\bar{\sigma}(x)$ is the logistic sigmoid function:

$$\bar{\sigma}(x) = \frac{1}{1 + e^{-x}} \tag{7}$$

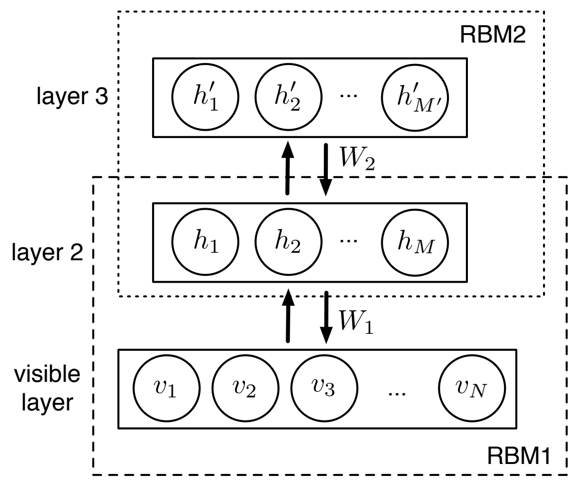

Figure 2. A 3-layer deep belief network comprised of 2 RBMs

The Gibbs sampling Markov chain can take quite a long time to produce actual samples from the joint distribution, so in practice the chain is started at a training example and run for a small number of iterations. Using this estimate of the log-likelihood gradient, we are instead minimizing a quantity referred to as the *contrastive divergence* between the training data and the model [2, 5]. Contrastive divergence updates for the RBM parameters are shown below:

$$\Delta W_{ij} \propto \langle v_i h_j \rangle_0 - \langle v_i h_j \rangle_k \tag{8}$$

$$\Delta a_i \propto \langle v_i \rangle_0 - \langle v_i \rangle_k \tag{9}$$

$$\Delta b_j \propto \langle h_j \rangle_0 - \langle h_j \rangle_k \tag{10}$$

Where $\langle \cdot \rangle_k$ denotes the value of the quantity after k iterations of Gibbs sampling, and for $k = 0$, v_i is simply the training data and h_j is a sample from (6) given the training data. Typically, these updates are performed using multiple training examples at a time by averaging over the updates produced by each example. This helps to smooth the learning signal and also helps take advantage of the efficiency of larger matrix operations. As $k \to \infty$, these updates approach maximum likelihood learning.

2.2 Stacking RBMs

A deep belief network is formed when multiple RBMs are stacked on top of each other as shown in Figure 2. After training a first-level RBM using the training data, we can perform a deterministic up-pass by setting the hidden units to their real-valued activation probabilities using (6) for each visible training vector. This is the same as what is done in the up-pass in a deterministic neural network. These deterministic hidden unit values are then used as the visible data in a subsequent higher-level RBM, which is also trained using contrastive divergence learning. This RBM stacking continues until the network reaches the desired depth. This greedy layer-by-layer training approach is a useful procedure for learning a set of non-linear features in an unsupervised manner [4], and it has been shown

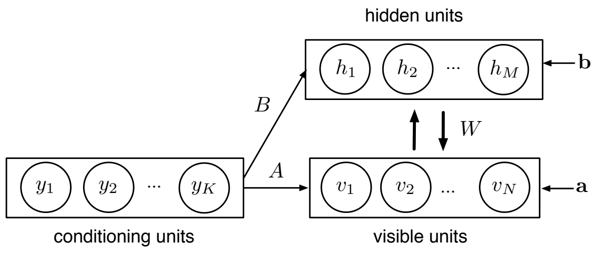

Figure 3. A conditional restricted Boltzmann machine. The correct label unit activations are provided as part of the visible unit data during training.

to be a beneficial pre-training procedure when followed by discriminative backpropagation [6].

2.3 The Conditional Restricted Boltzmann Machine

The conditional restricted Boltzmann machine (CRBM) takes the RBM a step further by adding directed connections between additional visible units, y_i, and the existing visible and hidden units, as shown in Figure 3. These additional units can represent any type of additional information, including visible data from the recent past. Because of this, the CRBM is an effective generative model of time sequence data [11]. This fact is what motivated our use of the CRBM to model drum patterns.

The directed connections, which are represented by weight matrices A and B, replace the bias terms, $\mathbf{a}$ and $\mathbf{b}$ in (5),(6), with dynamic bias terms, $\hat{\mathbf{a}}$ and $\hat{\mathbf{b}}$.

$$\hat{\mathbf{a}} = \mathbf{a} + A\mathbf{y} \tag{11}$$
$$\hat{\mathbf{b}} = \mathbf{b} + B\mathbf{y} \tag{12}$$

Where $\mathbf{y}$ is a vector containing the conditioning data. This modified RBM models the distribution $P(\mathbf{v}, \mathbf{h}|\mathbf{y})$, and the learning rules in (8)–(10) are unchanged except for the addition of the dynamic bias terms to the sampling expressions. The learning rules for the conditional weight matrices also have a familiar form:

$$\Delta A_{ij} \propto \langle v_i y_j \rangle_0 - \langle v_i y_j \rangle_k \tag{13}$$
$$\Delta B_{ij} \propto \langle h_i y_j \rangle_0 - \langle h_i y_j \rangle_k \tag{14}$$

Note that the y_j above are simply the training values and are not stochastically sampled in any way.

3. MODELLING AND ANALYZING DRUM PATTERNS

3.1 Bounded Linear Units

Drum patterns are not simply a series of ones and zeros, onset or no onset. Most drum patterns contain an appreciable sonic difference between accented and unaccented notes on every drum or cymbal, and it is these differences which give drum patterns their character. In order to effectively model drum patterns using the CRBM, we must modify the binary-valued visible units to be real-valued.

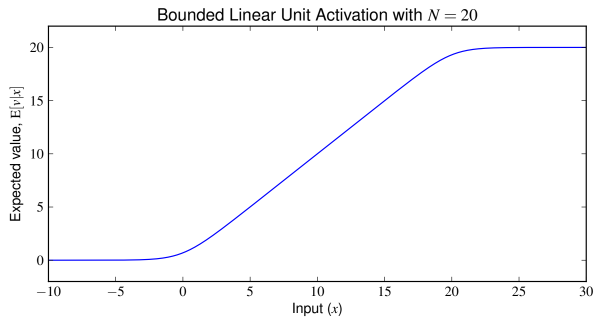

Figure 4. Bounded linear unit activation function

There are many options for getting real-valued visible activations out of RBMs; in fact, it has been shown that every distribution in the exponential family is a viable candidate [13]. A popular choice is the Gaussian distribution due to its familiarity and ubiquity; however, the unboundedness of Gaussian samples does not translate well to the space of dynamic levels possible within a drum pattern.

In order to model the bounded nature of playing dynamics, we use a modified version of the *rectified linear units* (RLUs) described in [9]. RLUs are constructed from a series of binary units with identical inputs but with fixed, increasing bias offsets. If the bias offsets are chosen appropriately, the expected value and variance of the number of active units out of these N tied binary units with common input x is:

$$\mathrm{E}[v|x] = \log(1 + e^x) - \log(1 + e^{x-N}) \tag{15}$$
$$\mathrm{Var}(v|x) = \bar{\sigma}(x) - \bar{\sigma}(x - N) \tag{16}$$

As can be seen in Figure 4, (15) yields a sigmoidal curve that saturates when $x > N$, bottoms out when $x < 0$, and is approximately linear in between. In the linear region, the variance is equal to one, so the value of N is chosen to achieve the desired level of noisiness in the samples, and the training data can be rescaled to the interval $[0, N]$. In this work, we have chosen $N = 20$, so that a value of 20 represents the loudest possible drum strike, while zero represents the absence of a drum strike. To sample from these *bounded linear units* (BLUs), instead of actually sampling from N binary units with stepped offsets, we approximate their total output with:

$$P(v|x) \sim \Big[\mathcal{N}\big(\mathrm{E}[v|x], \mathrm{Var}(v|x)\big)\Big]_0^N \tag{17}$$

where $\mathcal{N}(\cdot)$ is a normal distribution with mean and variance provided by (15) and (16), and $\big[\,\cdot\,\big]_0^N$ snaps values outside of the interval $[0, N]$ to the boundaries of the interval. Because these BLUs are constructed from logistic binary units, all of the RBM learning rules from Section 2 are still valid; the only thing that changes is how we sample from the visible BLUs.

3.2 Label Units

If bounded linear units give us a way to get drum onset information into the network, label units are how we get information out of the network. A standard approach to multi-class classification with neural networks is to use a

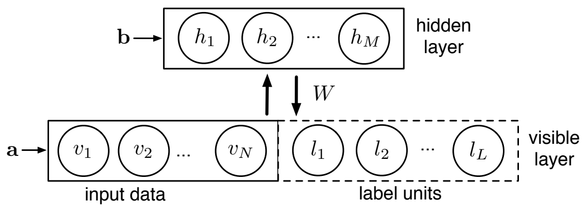

Figure 5. An RBM with an added group of visible label units.

group of softmax output units, which assigns a value to each of its units (each with input x_i) based on the softmax activation function shown in (18). This activation function is convenient for classification because the activation values of the group sum to one, which allows the output values to be interpreted as posterior class probabilities given the input data.

$$\mathcal{S}_{\max}(x_i, \mathbf{x}) = \frac{e^{x_i}}{\sum_j e^{x_j}} \tag{18}$$

In the realm of RBMs and deep learning, a different approach can be used which entails providing the ground truth class labels as part of the visible data during training. This approach has been shown to be more effective than using a separate softmax output layer in certain cases [6], and it indeed achieves better results for our application. Instead of adding the label units to a separate output layer, we augment the visible layer in the top-level RBM of a deep belief network with a group of softmax label units, as shown in Figure 5. This allows us to train the top-level RBM using the label units as visible data, by turning on only the correct label unit during training. Once this labeled RBM has been trained, we can compute the posterior activation probability under the model of each of the label units given the data, $P(l|\mathbf{v})$, using (19) and (20) (see [5]):

$$\mathcal{F}(\mathbf{v}|l) = -\sum_i v_i^{(l)} a_i - \sum_j \log\left(1 + \exp\left(x_j^{(l)}\right)\right) \tag{19}$$

$$P(l|\mathbf{v}) = \frac{e^{-\mathcal{F}(\mathbf{v}|l)}}{\sum_k e^{-\mathcal{F}(\mathbf{v}|k)}} \tag{20}$$

Where $x_j^{(l)} = b_j + \sum_i W_{ij} v_i^{(l)}$, and $v_i^{(l)}$ denotes the visible data but with only unit l of the label unit group activated. This calculation is tractable due to the typically small number of label units being evaluated.

3.3 Modelling Drum Patterns

In our drum pattern analysis network, we always start with a CRBM at the first layer. This CRBM models the current drum beat or subdivision using one BLU visible unit per drum or cymbal. In our experiments, we use a minimal three-drum setup: bass drum, snare drum, and hi-hat, but this can be expanded to work with any percussion setup. The conditioning units, y_j, of the CRBM contain drum activations from the recent past. In our experiments, $\mathbf{y}$ is fed with drum activations from the most recent two measures (or 32 subdivisions given a 4/4 time signature with sixteenth note subdivisions).

Subsequent layers use binary visible units instead of BLUs. Intermediate layers of the DBN can be made up of either additional CRBMs or standard RBMs, and the final layer must have visible label units to represent the classifier output. Using an intermediate-layer CRBM allows the layer to take into account past hidden unit activations of the layer below it, which allows it to learn higher-level time dependencies. In doing so, it increases the past time context that the top-level layer sees, since the past hidden unit activations of a first-level CRBM have been conditioned on the past relative to themselves. In order to make a fair comparison between DBNs that use different numbers of CRBM layers, we must make sure that the top layer always has access to the same amount of visible first-layer data from the past.

In our experiments, we train the label units to detect the current sixteenth note beat subdivision within the current 4/4 measure. In the next section, we give details on the configuration and training of the various DBNs that we test for this task.

4. TRAINING THE SYSTEM

4.1 Training Data

The dataset consists of 173 twelve-measure sequences comprising a total of 33,216 beat subdivisions, each of which contains bass drum, snare drum, and hi-hat activations. The data was collected using electronic Roland V-drums [1], quantized to exact sixteenth note subdivisions, and converted to a subdivision-synchronous drum activation matrix.

The sequences were intended to span a sizeable, but by no means complete, collection of popular rhythmic styles. There is a strong rock bias, with many beats featuring a prominent back beat; however, also included are more syncopated styles such as funk and drum ‘n’ bass as well as the Brazilian styles samba and bossa nova. We use a randomized 70/20/10 split for training, testing, and validation data, respectively.

4.2 DBN Configurations

We test four DBN configurations. For each of the four network architectures, we tested multiple hidden unit configurations and have chosen to present only those which performed best on the test data for each architecture. They are as follows:

1. *1-layer: Labeled-CRBM*
   3 visible data units + 16 visible label units, 100 hidden units, and 32 past subdivisions of context (96 conditioning units)

2. *2-layers: CRBM → Labeled-RBM*
   Each with 100 hidden units. The CRBM again has a context of 32 subdivisions.

[1] `http://rolandus.com`

3. *2-layers: CRBM → Labeled-CRBM*
   With 100 and 200 hidden units respectively. Each CRBM has a context of 16.

4. *3-layers: CRBM → CRBM → Labeled-RBM*
   With 100, 200, and 100 hidden units respectively. Both CRBMs have a context of 16 subdivisions.

4.3 DBN Training

Each non-labeled layer was trained using contrastive divergence with $k = 1$ (CD-1) for 300 sweeps through the training data with an update batch size of 100. The order of the training data was randomized in order to smooth the learning.

Top-level labeled layers were trained with the correct visible label unit switched on and the other label units switched off. We pre-trained each labeled layer using CD with $k = 1$ for 150 epochs and then k was linearly increased from 1 to 10 for an additional 150 epochs.

After pre-training each layer, we used discriminative backpropagation to fine-tune the network by backpropagating the cross-entropy label unit error to the lower layers [6]. Backpropagation was run for 400 epochs, but in the end we used the model parameters which produced the lowest cross-entropy validation error during training.

This type of training relies heavily on multiplying large matrices, which can be done considerably faster using highly data-parallel graphics processing units (GPUs). We use Gnumpy [12], a Python module which provides Numpy-like [2] bindings for matrix operations on Nvidia GPUs. Using an Nvidia Tesla C2050 GPU, training the single-layer model (#1) took around 20 minutes, while the 3-layer model (#4) took around 30 minutes. The typical split between pre-training time and backpropagation time was around 60%/40%.

4.4 Viterbi Decoding

In addition to simply classifying each subdivision individually, we can take into account additional sequential context by providing the label probabilities as posterior state probabilities in a hidden Markov model (HMM) [10]. In order to maximize coherence between successive beat subdivision estimates, we assign a high probability of a transition to the next successive beat and give an equal division of the remaining probability to other transitions. Since our system is designed for live use, we use strictly causal Viterbi decoding to estimate the current beat subdivision.

5. RESULTS

5.1 Independent Subdivision Classification

Here we present the classification results for beat-measure alignment. The test data contains 16 beat subdivisions per 4/4 measure, so we use 16 separate label units in the training. We were concerned with the prevalence of half-note symmetry in most back-beat-oriented drum patterns. For example, distinguishing between the first quarter note and the third quarter note of many basic rock patterns is virtually impossible without additional contextual information. Even though this phenomenon caused the majority of classification errors, the networks seemed to do well on the whole despite it.

[2] `http://numpy.org`

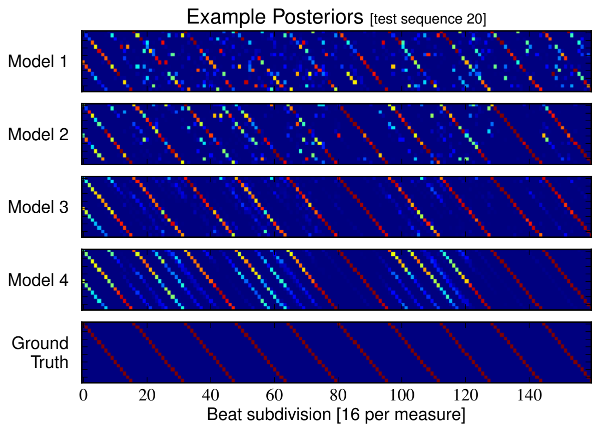

Figure 6. Example posteriors subdivision probabilities from the four models and the ground truth labels. The columns in each matrix show the posterior probability of each label for a particular beat subdivision.

Table 1 shows the classification results for each model. The single layer model was significantly outperformed by all multi-layer models, and adding a third layer did not seem to provide additional benefit on our test data. Example posterior label probabilities for each model are shown in Figure 6.

DBN Configuration	Train Accuracy	Test Accuracy
L-CRBM	97.2	76.2
CRBM→L-RBM	94.9	80.8
CRBM→L-CRBM	91.0	**83.7**
CRBM→CRBM→L-RBM	89.6	81.1

Table 1. Subdivision classification accuracy for each network configuration

5.2 With Viterbi Decoding

Now we present the classification results when using Viterbi decoding. We were concerned there would be a tendency for the high sequential state transition probability to increase the number of classification errors in the presence of half-note offset ambiguities; however, the decoding only seemed to help classification. Strong half-note ambiguities seemed to provide strong enough evidence for both alternatives that the original independent classification decisions were typically unaffected by the Viterbi decoding.

As shown in Figure 7, increasing the sequential transition probability increases the overall beat classification accuracy; however, in a real-world application, one cannot simply set this probability to one or else the decoder could

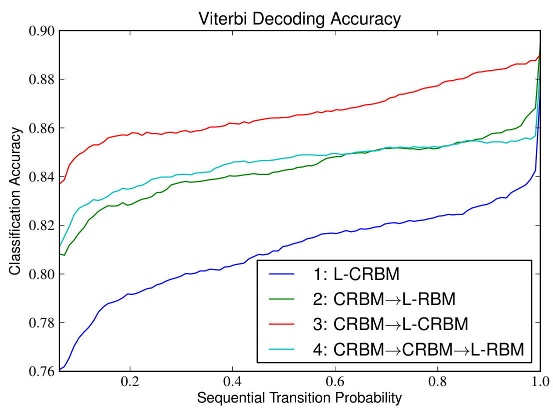

Figure 7. Viterbi decoding classification accuracy with increasing sequential transition probability

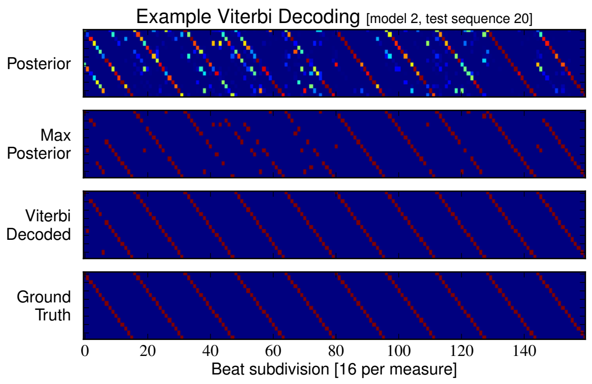

Figure 8. Row 1: Example posterior probabilities. Row 2: Independent classifications. Row 3: Viterbi decoded classifications. Row 4: Ground truth labels.

possibly be locked into an incorrect beat-measure alignment for the entire song. The decoder must also be allowed to adjust its estimates when beats are purposely left out by the drummer. Therefore, this parameter should be set with the end use case in mind. Example viterbi decoding results are shown in Figure 8.

6. DISCUSSION

Our results show the benefit of using a multi-layer neural network that is pre-trained as a deep belief network for analyzing drum patterns; however, it is likely that the actual optimal network configuration will be highly dependent on the diversity of the drum patterns in the dataset.

The results in Table 1 show an inverse relationship between training and test accuracy, which suggests overfitting was occurring during backpropagation. Subsequent work should focus on a more robust evaluation of the results using a larger dataset, cross-validation, and more attention to regularization techniques. In addition, comparison with existing drum pattern analysis methods is necessary.

Although, we do not objectively evaluate the use of these models for generating drum patterns, it is important to note that because the RBM is inherently a generative model, these networks are especially well-suited to serve as stochastic drum machines. Even a single labeled-CRBM works well for this purpose, and turning on the label unit of the desired subdivision during Gibbs sampling helps increase the metric stability of the generated patterns.

This type of model has significant potential to be of use in many music information retrieval and computer music tasks. We plan to explore the ability of the model to discriminate between rhythmic styles, discern between different time signatures, or to detect rhythmic transitions or fills. We also plan to do a more in-depth evaluation of the generative abilities of the model as well as to pursue methods which will allow interactive improvisation between human and computer performers.

Additional information as well as the dataset and code used in this work will be made available at: `http://www.eecs.berkeley.edu/~ericb/`.

7. REFERENCES

[1] E Battenberg, V Huang, and D Wessel. Toward live drum separation using probabilistic spectral clustering based on the itakura-saito divergence. *Proc. AES Conference on Time-Frequency Processing*, 2011.

[2] Y Bengio. Learning deep architectures for AI. *Foundations and Trends in Machine Learning*, 2009.

[3] M Goto. An audio-based real-time beat tracking system for music with or without drum-sounds. *Journal of New Music Research*, 30(2):159–171, 2001.

[4] P Hamel and D Eck. Learning features from music audio with deep belief networks. *Proc. of the 11th International Society for Music Information Retrieval Conference (ISMIR 2010)*, pages 339–344, 2010.

[5] G Hinton and S Osindero. A fast learning algorithm for deep belief nets. *Neural Computation*, 2006.

[6] G. E. Hinton. To recognize shapes, first learn to generate images. *Progress in brain research*, 165:535–547, 2007.

[7] AP Klapuri, AJ Eronen, and JT Astola. Analysis of the meter of acoustic musical signals. *IEEE Transactions on Speech and Audio Processing*, 14(1):342, 2006.

[8] A Mohamed and G Dahl. Deep belief networks for phone recognition. *NIPS Workshop on Deep Learning for Speech Recognition and Related Applications*, 2009.

[9] V. Nair and G. E. Hinton. Rectified linear units improve restricted boltzmann machines. *Proc. 27th International Conference on Machine Learning*, 2010.

[10] L.R. Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.

[11] G Taylor and G Hinton. Two Distributed-State Models For Generating High-Dimensional Time Series. *Journal of Machine Learning Research*, 2011.

[12] T Tieleman. Gnumpy: an easy way to use GPU boards in Python. Technical Report 2010-002, University of Toronto, 2010.

[13] M Welling and M Rosen-Zvi. Exponential family harmoniums with an application to information retrieval. *Advances in Neural Information Processing Systems*, 2005.

Research supported by Microsoft (Award #024263) and Intel (Award #024894) funding and by matching funding by U.C. Discovery (Award #DIG07-10227). Additional support comes from Par Lab affiliates National Instruments, Nokia, NVIDIA, Oracle, and Samsung.