A Dual Classification Approach to Music Language Modelingl

Yoav Zimmerman (304125151)
CS 260: Machine Learning Algorithms

March 24, 2016

1 Abstract

I investigate the problem of modeling symbolic sequences of polyphonic music in a completely general piano-roll representation. I introduce an original architecture that poses the problem as a dual-classification task rather than one with a multimodal probability distribution. This architecture has the flexibility to optimize performance for melodic or harmonic prediction separately, or both. The approach achieves an accuracy of **77.84%** on the Nottingham dataset and is used to generate original sequences of music.

2 Background

Modeling temporal sequences is an important area of neural network research, since many naturally occurring phenomena such as music, speech, or human motion are inherently sequential. Musical sequences are one example of sequential data that is highly dependent on temporal relations. For example, a musical pattern and theme appearing early in one part of a musical piece may appear again several measures later and many more times throughout the piece. Recurrent neural networks (RNN) [10] contain an internal memory that can learn over arbitrary sequential data, making them a good fit, in principle, for modeling this type of data. Training them effeciently by gradient based optimization methods such as stochastic gradient descent has long been known to be a difficult due to the exploding gradient problem [3], but Long-Short Term Memory (LSTM) cells [6] have been shown as a way to offset this problem. LSTM cells in particularly have been shown to perform exceptionally well at language modeling tasks [8], making them ideal candidates for this task.

In this paper, I investigate *Music Language Modeling*, the problem of modeling symbolic sequences of polyphonic music in a completely general piano-roll representation. It is important to note that this paper will only be examining the timing and pitch information typically contained in a score or a MIDI file, rather than more complex, acoustically rich audio signals. The main use case for these models is in polyphonic transcription, the challenge of predicting the underlying notes of a polyphonic audio signal without access to the underlying score. Music language models are usually trained to predict the conditional probability notes to be played together in a time steps, given all the time steps that have occurred before it. Eck. et al [5] is an early attempt to use a LSTM network to learn the structure of monophonic blues music. Allan et. al [2] have approached this problem from using

Hidden Markov Models to learn over chord harmonizations of the late composer J.S. Bach.

One major issue with predicting high-dimensional sequences of music is that they are highly multi-modal. Concretely, the appearance of a note in a given time step considerably modifies the probability with which other notes may occur at the same time. Boulanger-Lewandowski et. al [4] is a leading paper in the field that deals with these difficulties by using energy based models such as the restricted Boltzmann machine (RBM). Specifically, they propose the RNN-RBM, a model in which the output of an RNN feeds directly into the parameters of an RBM used to predict a multi-modal distribution. This paper instead takes the approach of constraining the potential configurations of each time step so the task can be posed as a classification problem by introducing musical structure.

3 Model

To begin the description of our model, I first describe the naive RNN formulation proposed by Boulanger-Lewandowski et. al [4].

a. Cross-Entropy Formulation

To apply a sequence of musical sequences directly to a RNN, we first discretize the sequence into time steps, then encode each time step into an input vector. This input vector includes 88 binary visible units that span the whole range of the piano from A0 to C8. The time step is a hyperparameter that should be varied appropriately to the dataset being learned over. Once the melodic sequence is encoded, it can be passed into a RNN to predict the next time step $t+1$, given all the previous time steps $t, t-1, ...$ The loss function used is log-likelihood cross-entropy, to account for multiple notes potentially being active at each time step. This loss function for a sequence of T time steps $x_t \in \mathbb{R}^N$ is shown in (1).

$$\mathcal{L}(\{x_t\}; f) = \frac{1}{T} \sum_{t}^{T-1} \sum_{n}^{N} \Big(x_n^{t+1} log(f(x_n^t)) + (1 - x_n^{t+1}) log(1 - f(x_n^t)) \Big) \quad (1)$$

where $f(x_n^t)$ is the output layer of the RNN for input x_n^t.

b. Dual Softmax Formulation

The original contribution of this paper is to pose this problem as a two-class classification task by seperating each time step t into seperate melody and harmony parts. One part of the input vector $x_m^t \in \mathbb{R}^M$ is a one-hot encoding of the melody note playing at time t. The other part of the input vector $x_h^t \in \mathbb{R}^H$ is a one-hot encoding of the harmony class playing at time t, which is usually a chord class such as *C Major* or *G Minor*. The full input vector $x^t \in \mathbb{R}^{M+H}$ is a concatenation of the melody and harmony segments. Note the important restriction that only one melody and harmony class are playing at a time, which makes the model more restrictive than the naive case above. However, this restriction allows for the dual-softmax loss function (3) for a sequence of T time steps $x_t \in \mathbb{R}^{M+H}$ with melody and harmony targets $m_t \in \mathbb{R}^M$ and $h_t \in \mathbb{R}^H$ respectively.

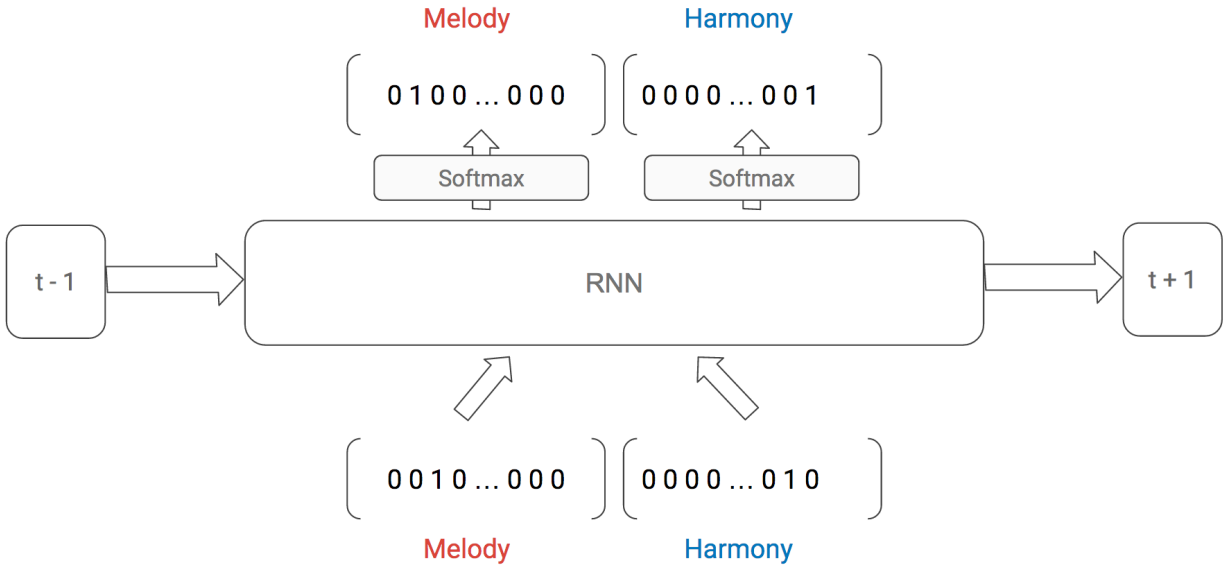

Figure 1: Dual Softmax Network

$$l(x, m, h; f) = \alpha \, log\left(\frac{e^{f(x)_m}}{\sum_{n=0}^{M-1} e^{f(x)_n}}\right) + (1-\alpha) \, log\left(\frac{e^{f(x)_{M+h}}}{\sum_{n=M}^{M+H} e^{f(x)_n}}\right) \quad (2)$$

$$\mathcal{L}(\{x_t\}; f) = \frac{1}{T} \sum_{t}^{T-1} l(x_t, m_{t+1}, h_{t+1}; f) \quad (3)$$

α is the *melody coefficient*, which is used to tune the importance of predicting the melody. If α is at 1.0, the model will ignore the harmony and learn exclusively over the melody targets. Conversely, α can be set to zero to learn exclusively over the harmony targets. This hyperparameter introduces a flexibility that is uniquely advantageous to this model over the cross-entropy formulation. For example, a model that needs to sample a harmonic backing in real-time to a improvised melody can lower this coefficient to reflect the goals of their task.

4 Experiments

a. Dataset

The **Nottingham** dataset from [4], a collection of 1200 folk tunes, was used to test this model. 203 (16%) melodic sequences that did not fit the singular melody and harmony described above were filtered from the dataset, leaving 997 sequences. A 65/15/15 train, test, and validation split were used to train all described models. For discretization of the melodic sequences, a time step of one sixteenth note was used (Boulanger-Lewandowski et. al [4] uses a twice as long time-

step of one eighth note). Using this time step length, the average sequence length was 516 with a max sequence length of 3588. The python-mingus [7] open source music library was used to parse the harmonies into 31 chord classes + 1 class representing a harmony rest. The range of melody note classes was 33 notes + 1 class representing a melody rest, for a input dimension of 66.

b. Implementation

The RNN was implemented using Google's open-source TensorFlow library [1]. The network was implemented as l hidden layers with k LSTM cells each and one output dual-softmax layer. The network was trained with minibatch stochastic gradient descent, using the RMSProp learning scheme [9] with a learning rate of 0.005 and a learning rate decay of 0.9. Backpropagation through time (BPTT) was truncated to a length of 128 time steps (8 measures). For most experiments, dropout with a probability of 50% was applied to the hidden layers and with a probability of 20% to the input layers. Dropout was only applied to non-recurrent connections as described in Zaremba et. al [11].

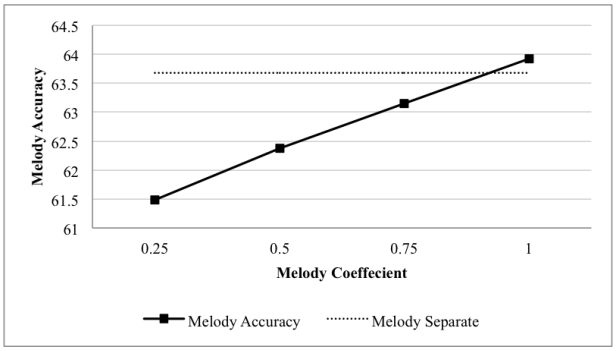

Figure 2: Melody accuracy on the test set over several different melody coeffecient values.

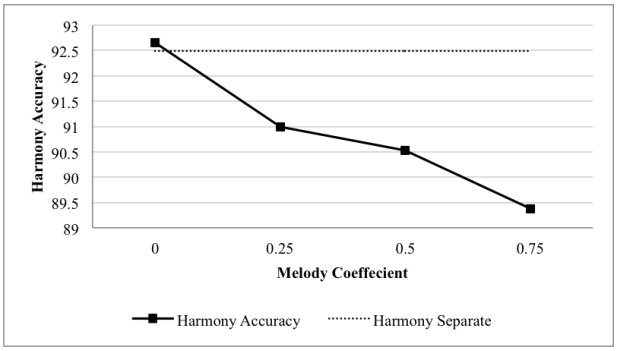

Figure 3: Harmony Accuracy on the test set over several different melody coeffecient values.

c. Results

Figure 1 shows the test set accuracy of several different hyperparameter combinations attempted. Note that this is the classification accuracy, which is not the same as the accuracy metric computed in Boulanger-Lewandowski et. al [4].

To evaluate the effect of the melody coeffecient (α), several models were trained using different values of coeffecients on an architecture of 1 hidden layer with 100 LSTM units. Figures 2 and 3 show the results of these experiments. The dashed line on each graph is a baseline created by training a model that only uses the harmony or melody vectors as input, instead of the entire concatenated input vector as described in the model above. The melody baseline, for example, is equivalent to a model trained with a melody coefficient of 1.0, except that it only takes the melody segment as input (ignoring the harmony segment). The same can be said for the harmony baseline.

Num Layers	1			2		
Hidden Size	50	100	200	50	100	200
Testing Loss	0.5896	0.5749	0.5343	0.5721	0.5575	**0.5363**
Accuracy (%)	75.52	75.59	77.23	76.26	76.72	**77.84**

Table 1: Model accuracy over a variety of configurations

5 Analysis

It is difficult to compare these results to Boulanger-Lewandowski et. al [4], as this task is a constant label classification, and theirs is a variable label classification. No similar results were found in related papers to compare these to. However, one qualitative method to evaluate the models is to generate an original musical sequence by "priming" it with an initial few measures from a test sequence. Then, the output of the model is sample from to generate harmony and melody notes, which are then plugged back into the input of the next time step. Musical sequences generated by the best performing model are included as supplementary material with this paper.

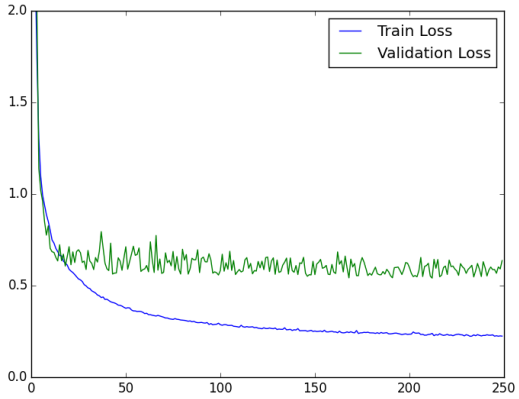

Figure 4: Training graph for a model with 2 hidden layers and 200 hidden units, using a dropout of probability 0.5 for the hidden layers and 0.8 for the input layer

The melody coeffecient experiments verify the intuition that a higher melody coefficient corresponds to a higher melody accuracy and vice versa. However, the baselines in figures 2 and 3 point out a weakness in the model. Although the best performing melody and harmony accuracies do beat their respective baselines, the performance gain is slim, with 0.25% better for melody and 0.16% better for harmony. This performance gain should be more substantial, as the harmony input is extremely helpful in predicting correct notes to play during the melody and vice versa. For example, the melody note C is more likely to be played than $C\#$ if the active harmony class is *C Major*.

One difficulty that may have caused this issue is overfitting, which was a difficulty encountered in training every model throughout this project. Although extensive hyperparameter search was performed over aggressive dropout values, most training graphs looked similar to the one in figure 4, reaching a point where validation loss became noisy and nondecreasing on average. In the case of the baseline models, the reduced size of the input vector could have resulted in a smaller overall number of parameters and lessened the

harm of overfitting.

6 Future Work

In future work, the problem of overfitting could be addressed by methods other than dropout. One option is to find a larger dataset than Nottingham. Another option is to synthetically produce an arbitrarily large musical dataset by generating permutations of chord progressions and playing melodic scales over those harmonic classes.

Additionally, I would like to explore different architectures based on the dual-softmax model. One idea is to place one or more separate hidden layers of melody and/or harmony before concatenating or element-wise multiplicating them into one or more combined hidden layers. This is motivated by the effectiveness of the melody and harmony baselines in figures 2 and 3; perhaps the model benefits from learning seperate abstractions of the inputs before combining them. Another idea is to combine more than 2 instruments together for an *n-softmax* architecture; one could imagine a seperate softmax target for each instrument of an orchestral composition, for example.

References

[1] TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[2] Allan, M., and Williams, C. K. Harmonising chorales by probabilistic inference. *Advances in neural information processing systems 17* (2005), 25–32.

[3] Bengio, Y., Simard, P., and Frasconi, P. Learning long-term dependencies with gradient descent is difficult. *Neural Networks, IEEE Transactions on 5*, 2 (1994), 157–166.

[4] Boulanger-Lewandowski, N., Bengio, Y., and Vincent, P. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392* (2012).

[5] Eck, D., and Schmidhuber, J. Learning the long-term structure of the blues. In *Artificial Neural NetworksICANN 2002*. Springer, 2002, pp. 284–289.

[6] Hochreiter, S., and Schmidhuber, J. Long short-term memory. *Neural computation 9*, 8 (1997), 1735–1780.

[7] Spaans, B. python-mingus. `https://github.com/bspaans/python-mingus`, 2015.

[8] Sundermeyer, M., Schlüter, R., and Ney, H. Lstm neural networks for language modeling. In *INTERSPEECH* (2012), pp. 194–197.

[9] Tieleman, T., and Hinton, G. Lecture 6.5—RmsProp: Divide the gradient by a running average of its recent magnitude. COURSERA: Neural Networks for Machine Learning, 2012.

[10] Williams, R. J., and Zipser, D. A learning algorithm for continually running fully recurrent neural networks. *Neural computation 1*, 2 (1989), 270–280.

[11] Zaremba, W., Sutskever, I., and Vinyals, O. Recurrent neural network regularization. *arXiv preprint arXiv:1409.2329* (2014).