Simulating Human Tonal Expectation using Recurrent Neural Networks

Carlos Cancino-Chacón[1,2], Maarten Grachten[2], and Kat Agres[3]

[1]Austrian Research Institute for Artificial Intelligence
[2]Johannes Kepler University of Linz
[3]Institute of High Performance Computing, A*STAR, Singapore

Technical Report; Version 0.01

Abstract

This document includes some of the technical details for reproducing the work presented in

From Bach to the Beatles: The simulation of human tonal expectation using ecologically-trained predictive models,

presented at the 18th International Society for Music Information Retrieval Conference (ISMIR 2017) in Suzhou, China.

1 Data preprocessing

An audio signal can be represented using a Constant-Q Transform (CQT) [Brown, 1991], which is a discrete frequency domain representation of audio.

The t-th slice of a CQT spectrogram is a non-negative vector $\boldsymbol{x}_t \in \mathbb{R}^{334}$ that represents frequencies between 27.5 and 16744.04 Hz with a resolution of 36 frequency bins per octave. The audio in all recordings has a sample rate of 44.1 kHz. We normalize each slice so that all of its components lie between 0 and 1 as

$$\hat{\boldsymbol{x}}_t \leftarrow \frac{\tanh(\boldsymbol{x}_t)}{\tanh(\max(\boldsymbol{x}_t) + \varepsilon)}, \tag{1}$$

where $\max(\boldsymbol{x}_t)$ represents the value of the largest component of $\boldsymbol{x}_t$, $\tanh(\cdot)$ is the element-wise hyperbolic tangent function and ε is a machine epsilon for numerical stability. In this work, we set $\varepsilon = 10^{-10}$. Slightly abusing notation, in the following we will assume that all CQT slices have been normalized, and therefore simply refer to the normalized CQT slices $\hat{\boldsymbol{x}}$ as $\boldsymbol{x}$.

2 Recurrent Neural Networks

2.1 Architectures

An RNN is a neural architecture that allows for modeling dynamical systems [Graves, 2013]. Let $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t$ be a sequence of N-dimensional (normalized) input vectors and

$\boldsymbol{y}_1, \ldots, \boldsymbol{y}_t$ be its corresponding sequence of outputs. An RNN provides a natural way to model $\boldsymbol{x}_{t+1}$, the next event in the sequence, by using the outputs of the network to parametrize a predictive distribution given by

$$p(x_{t+1,i} \mid \boldsymbol{x}_t, \ldots, \boldsymbol{x}_1) = y_{t,i} \tag{2}$$

where $x_{t+1,i}$ and $y_{t,i}$ are the i-th component of $\boldsymbol{x}_{t+1}$ and $\boldsymbol{y}_t$ respectively.

The basic component of an RNN is the *recurrent layer*, whose activation at time t depends on both the input at time t and its activation at time $t-1$. Although theoretically very powerful, in practice RNNs with *vanilla* recurrent layers are known to have problems learning long term dependencies due to a number of problems, including vanishing and exploding gradients [Pascanu et al., 2013]. Other recurrent layers such as LSTM layers [Hochreiter and Schmidhuber, 1997] and gated recurrent units (GRUs)[Chung et al., 2014] try to address some of these problems by introducing special structures within the layer, such as purpose-built memory cells and gates to better store information. More recently, recurrent layers with multiplicative integration (MI-RNNs) [Wu et al., 2016] have been shown to extend the expressivity of traditional additive RNNs by changing the way the information from different sources is aggregated within the layer while introducing just a small number of extra parameters.

The predictions of the network can be written as

$$\boldsymbol{y}_t = \sigma\left(\mathbf{W}_y \boldsymbol{h}_t(\boldsymbol{x}_t \mid \theta_h) + \mathbf{b}_y\right), \tag{3}$$

where $\boldsymbol{h}_t \in \mathbb{R}^H$ is the output of the hidden layer at time t, θ_h are the parameters of the layer, $\mathbf{W}_y \in \mathbb{R}^{334 \times H}$ is a matrix of weights connecting the recurrent layer to the output $\mathbf{b}_y \in \mathbb{R}^{334}$ is a bias vector and $\sigma(\cdot)$ is the element-wise sigmoid function.

A detailed description of the recurrent layers used in this paper can be found in Appendix A.

2.2 Training

Given a set of T training sequences $\mathbf{X} = \{\{\boldsymbol{x}_1^{(k)}, \cdots \boldsymbol{x}_{T_i-1}^{(k)}\} \mid 1 \leq k \leq T\}$, where T_k is the length of the k-th sequence, and their respective targets $\mathbf{T} = \{\{\boldsymbol{x}_2^{(k)}, \cdots \boldsymbol{x}_{T_i}^{(k)}\} \mid 1 \leq k \leq T\}$, i.e. a shifted version of the input sequences, the parameters of the network are learned by minimizing the mean cross entropy (MCE) [Bishop, 2006]

$$MCE = -\frac{1}{TN} \sum_{k=1}^{T} \frac{1}{T_k} \sum_{t=1}^{T_k} \sum_{i=1}^{N} CE_{kti}, \tag{4}$$

where CE_{kti} is the cross entropy of the i-th feature at time-step t for the k-th sequence given by

$$CE_{kti} = t_{kti} \log(p_{kti}) + (1 - t_{kti}) \log(1 - p_{kti}), \tag{5}$$

and $p_{kti} = p(x_{t,i}^{(k)} \mid \boldsymbol{x}_{t-1}^{(k)}, \ldots, \boldsymbol{x}_1^{(k)})$ and $t_{kti} = x_{t,i}^{(k)}$.

2.2.1 Biasing learning towards predicting change

A crucial question when applying discrete time recurrent models to a continuous stream of data such as audio is how to choose the rate of discrete time steps with

respect to the absolute time of the data. This choice depends on the approximate rate or temporal density of relevant events in the data—in our case the notes that make up the musical material. Ideally, we would like the discrete time steps to be small enough to capture the occurrence of even the shortest notes individually, but if the discrete time step is chosen much smaller than the median event rate, this leads to strong correlations between data at consecutive time steps. A result of this is that training models to predict the data at time-step $t+1$ teaches them to strongly expect the data at $t+1$ to be approximately equal to the data at t. Choosing a larger discrete step size for the model alleviates this problem, but has the disadvantage that the data the model sees at a particular time may actually be an average over consecutive events that happened within that larger step.

We slightly revise the training objective of the models as a remedy to this unfortunate trade-off. This revised objective biases the models to care more about correctly predicting the data at $t+1$ when the change from t to $t+1$ is large (e.g. the start of a new note) than when it is small (e.g a transition without any starting or ending note events). This allows us to use a relatively small step size without causing the models to trivially learn to expect the data to stay constant between consecutive time steps.

More specifically, we modify the original cross-entropy objective CE_{kti} by multiplying it with a time-varying weight w_{kt} as follows:

$$\tilde{CE}_{kti} \leftarrow w_{kt} CE_{kti}, \tag{6}$$

where w_{kt} is given by

$$w_{kt} = \begin{cases} 1 & \text{if } \sum_i^N |x_{t+1,i}^{(k)} - x_{t,i}^{(k)}| > \varepsilon \\ \beta & \text{otherwise} \end{cases} \tag{7}$$

where $\varepsilon \in \mathbb{R}$ acts as a threshold distinguishing small and large change transitions, and $\beta \in \mathbb{R}$ controls the relative influence of prediction errors on the training in the case of small change transitions[1].Based on an informal inspection of the model predictions in a grid search on β and ε, we choose $\beta = 10^{-3}$, and ε such that

$$P_{training}(\sum_i^N |x_{t+1,i} - x_{t,i}| \ \leq \ \varepsilon) = 0.505 \tag{8}$$

where $P_{training}(X)$ denotes the empirical probability of event X under the training data.

2.3 Hyperparameters

The models are trained using RMSProp [Tieleman and Hinton, 2012], a variant of stochastic gradient descent that adaptively updates the step-size using a moving average of the magnitude of the gradients. The initial learning rate is set to 10^{-3}. The gradients are computed using truncated back propagation through time, where computation of the gradients is truncated after 100 steps and are clipped at 1. Each training batch consists of 20 sequences of 100 CQT slices. Each sequence is selected randomly out of the training data. Thus, an epoch of training corresponds to the

[1]We empirically found a binary distinction between small and large change transitions to be more effective than a gradual weighting scheme

model seeing roughly the same number of time steps as in the whole fold. Early stopping is used after 100 epochs without any improvement in the test set.

3 Probe-tone experiments

Let $\mathbf{X} = \{\boldsymbol{x}_1, \cdots \boldsymbol{x}_T\}$ be an input musical stimulus, also referred to as context, and $\mathbf{T} = \{\boldsymbol{\tau}_1, \ldots, \boldsymbol{\tau}_{12}\}$ the set of probe-tones each corresponding to one of the 12 pitch classes. In order to quantitatively assess how well a probe-tone $\boldsymbol{\tau}$ fits the musical stimulus, we compare $\boldsymbol{y}^*$, the next prediction of the RNN given the input stimulus $\mathbf{X}$, and the probe-tone using the Kullback-Leibler (KL) divergence, given by

$$D_{KL}(\boldsymbol{\tau}||\boldsymbol{y}^*) = \sum_{i=1}^{N} \tau_i \log\left(\frac{\tau_i}{y_i}\right), \tag{9}$$

where τ_i and y_i are the i-th components of $\boldsymbol{\tau}$ and $\boldsymbol{y}^*$ respectively. In this case, we can see that the KL divergence is small when the expectations of the model are similar to the probe-tones.

We can construct the probe-tone profile for a stimulus in key *key*, denoted by$\boldsymbol{P}_{key}^{model} \in \mathbb{R}^{12}$, as

$$\boldsymbol{P}_{key}^{model} = \text{minmax}\begin{pmatrix} -D_{KL}(\boldsymbol{\tau}_1||\boldsymbol{y}_{key}^*) \\ \vdots \\ -D_{KL}(\boldsymbol{\tau}_{12}||\boldsymbol{y}_{key}^*). \end{pmatrix}, \tag{10}$$

where minmax$(\cdot)$ is a normalization function given by

$$\text{minmax}(\mathbf{x}) = \frac{\mathbf{x} - \min(\mathbf{x})\mathbf{1}}{\max(\mathbf{x}) - \min(\mathbf{x})}, \tag{11}$$

where $\min(\mathbf{x})$ and $\max(\mathbf{x})$ are the minimal and maximal components of the input $\mathbf{x}$ and $\mathbf{1} \in \mathbb{R}^{12}$ is a vector of ones.

The average profile for a stimulus is computed as

$$\boldsymbol{P}^{model} = \frac{1}{12}\sum_{key} \boldsymbol{P}_{key}^{model}. \tag{12}$$

We also normalize the human profiles reported by Krumhansl and Kessler by using the minmax function. We use Pearson's correlation coefficient to compare the profiles of the model to those of humans

$$r = \frac{\sum_{i=1}^{12}(P_i^{model} - \bar{P}^{model})(P_i^{human} - \bar{P}_i^{human})}{\sqrt{\sum_{i=1}^{12}(P_i^{model} - \bar{P}^{model})^2}\sqrt{\sum_{i=1}^{12}(P_i^{human} - \bar{P}^{human})^2}}, \tag{13}$$

where P_i re presents the i-th component of a profile P, and $\bar{P} = \frac{1}{12}\sum_i P_i$.

References

[Bishop, 2006] Bishop, C. M. (2006). *Pattern Recognition and Machine Learning*. Springer Verlag, Microsoft Research Ltd.

[Brown, 1991] Brown, J. C. (1991). Calculation of a constant Q spectral transform. *The Journal of the Acoustical Society of America*, 89(1):425–434.

[Chung et al., 2014] Chung, J., Gulcehre, C., Cho, K., and Bengio, Y. (2014). Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*.

[Graves, 2013] Graves, A. (2013). Generating Sequences With Recurrent Neural Networks. *arXiv*, 1308:850.

[Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long Short-Term Memory. *Neural Computation*, 9(8):1735–1780.

[Pascanu et al., 2013] Pascanu, R., Mikolov, T., and Bengio, Y. (2013). On the difficulty of training recurrent neural networks. In *Proceedings of the 30th International Conference on Machine Learning*, pages 1–9, Atlanta, Georgia, USA.

[Tieleman and Hinton, 2012] Tieleman, T. and Hinton, G. (2012). Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. In *COURSERA Neural Networks for Machine Learning*.

[Wu et al., 2016] Wu, Y., Zhang, S., Zhang, Y., Bengio, Y., and Salakhutdinov, R. (2016). On Multiplicative Integration with Recurrent Neural Networks. In *30th Conference on Neural Information Processing Systems (NIPS 2016)*, Barcelona, Spain.

A Recurrent layers

A.1 LSTM with Multiplicative Integration

Let $\boldsymbol{x}_t \in \mathbb{R}^N$ be an input vector representing the CQT at time t. The activation of the MI-LSTM layer is given by

$$\boldsymbol{z}_t = \tanh(\boldsymbol{\beta}_{z,1} \odot \mathbf{U}_z \boldsymbol{h}_{t-1} + \boldsymbol{\beta}_{z,2} \odot \mathbf{W}_z \boldsymbol{x}_t + \boldsymbol{\alpha}_z \odot \mathbf{W}_z \boldsymbol{x}_t \odot \mathbf{U}_z \boldsymbol{h}_{t-1} + \mathbf{b}_z) \tag{14}$$

$$\boldsymbol{i}_t = \sigma(\boldsymbol{\beta}_{i,1} \odot \mathbf{U}_i \boldsymbol{h}_{t-1} + \boldsymbol{\beta}_{i,2} \odot \mathbf{W}_i \boldsymbol{x}_t + \boldsymbol{\alpha}_i \odot \mathbf{W}_i \boldsymbol{x}_t \odot \mathbf{U}_i \boldsymbol{h}_{t-1} + \mathbf{b}_i) \tag{15}$$

$$\boldsymbol{f}_t = \sigma(\boldsymbol{\beta}_{f,1} \odot \mathbf{U}_f \boldsymbol{h}_{t-1} + \boldsymbol{\beta}_{f,2} \odot \mathbf{W}_f \boldsymbol{x}_t + \boldsymbol{\alpha}_f \odot \mathbf{W}_f \boldsymbol{x}_t \odot \mathbf{U}_f \boldsymbol{h}_{t-1} + \mathbf{b}_f) \tag{16}$$

$$\boldsymbol{o}_t = \sigma(\boldsymbol{\beta}_{o,1} \odot \mathbf{U}_o \boldsymbol{h}_{t-1} + \boldsymbol{\beta}_{o,2} \odot \mathbf{W}_o \boldsymbol{x}_t + \boldsymbol{\alpha}_o \odot \mathbf{W}_o \boldsymbol{x}_t \odot \mathbf{U}_o \boldsymbol{h}_{t-1} + \mathbf{b}_o) \tag{17}$$

$$\boldsymbol{c}_t = \boldsymbol{i}_t \odot \boldsymbol{z}_t + \boldsymbol{f}_t \odot \boldsymbol{c}_{t-1} \tag{18}$$

$$\boldsymbol{h}_t = \boldsymbol{o}_t \odot \tanh(\boldsymbol{c}_t), \tag{19}$$

where $\boldsymbol{z}_t, \boldsymbol{h}_t \in \mathbb{R}^{N_h}$ are the *block input* and *output* (i.e. the activation of the MI-LSTM layer), respectively, $\boldsymbol{c}_t \in \mathbb{R}^{N_h}$ is the *cell state* and $\boldsymbol{i}_t, \boldsymbol{f}_t, \boldsymbol{o}_t \in \mathbb{R}^{N_h}$ are the *input*, *forget* and *output gates*, respectively. The parameters of an MI-LSTM layer are $\{\mathbf{W}_k \mid k \in \{z, i, f, o\}\}$, input-to-hidden weight matrices in $\mathbb{R}^{N_h \times N}$, $\{\mathbf{U}_k \mid k \in \{z, i, f, o\}\}$, hidden-to-hidden weight matrices in $\mathbb{R}^{N_h \times N_h}$, $\{\mathbf{b}_k \mid k \in \{z, i, f, o\}\}$, bias vectors in $\mathbb{R}^{N_h}$ and $\{\boldsymbol{\alpha}_{\boldsymbol{k}}, \boldsymbol{\beta}_{k,1}, \boldsymbol{\beta}_{k,2} \mid k \in \{z, i, f, o\}\}$, gating bias vectors for multiplicative integration in $\mathbb{R}^{N_h}$. $\sigma(\cdot)$ and $\tanh(\cdot)$ are the elementwise sigmoid and hyperbolic tangent functions, respectively.