State-Frequency Memory Recurrent Neural Networks

Hao Hu [1] **Guo-Jun Qi** [1]

Abstract

Modeling temporal sequences plays a fundamental role in various modern applications and has drawn more and more attentions in the machine learning community. Among those efforts on improving the capability to represent temporal data, the Long Short-Term Memory (LSTM) has achieved great success in many areas. Although the LSTM can capture long-range dependency in the time domain, it does not explicitly model the pattern occurrences in the frequency domain that plays an important role in tracking and predicting data points over various time cycles. We propose the State-Frequency Memory (SFM), a novel recurrent architecture that allows to separate dynamic patterns across different frequency components and their impacts on modeling the temporal contexts of input sequences. By jointly decomposing memorized dynamics into state-frequency components, the SFM is able to offer a fine-grained analysis of temporal sequences by capturing the dependency of uncovered patterns in both time and frequency domains. Evaluations on several temporal modeling tasks demonstrate the SFM can yield competitive performances, in particular as compared with the state-of-the-art LSTM models.

1. Introduction

Research in modeling dynamics of time series has a long history and is still highly active due to its crucial role in many real world applications (Lipton et al., 2015). In recent years, the advancement of this area has been dramatically pushed forward by the success of recurrent neural networks (RNNs) as more training data and computing resources are available (Mikolov et al., 2010; Bahdanau et al., 2014; Graves et al., 2013). Although some sophisticated RNN models such as Long Short-term Memory (LSTM) (Hochreiter & Schmidhuber, 1997) have been proven as powerful tools for modeling the sequences, there are some cases that are hard to handle by RNNs. For instance, (Mittelman, 2015) demonstrates that RNN models either perform poorly on predicting the optimal short-term investment strategy for the high frequency trading or diverge, making them less preferred than other simpler algorithms.

One of the possible reasons for such situations is that RNN models like LSTM only consider the pattern dependency in the *time* domain, which is insufficient if we want to predict and track the temporal sequences over time at various frequencies. For example, in phonemes classification, some phonemes like 'p', 't' are produced by short, high-frequency signals, while others like 'iy', 'ey' are related to longer, low-frequency signals. Thus modeling such frequency patterns is quite helpful for correctly identifying phonemes in a sentence. Similarly, music clips are often composed of note sequences across a rich bands of frequencies. Automatically generating music clips often requires us to model both short and long-lasting notes by properly mixing them in a harmonic fashion. These examples show the existence of rich frequency components in many natural temporal sequences, and discovering them plays an important role in many prediction and generation tasks.

Thus, we strive to seamlessly combine the capacity of multi-frequency analysis with the modeling of long-range dependency to capture the temporal context of input sequences. For this purpose, we propose the State-frequency Memory (SFM), a novel RNN architecture that jointly decomposes the memory states of an input sequence into a set of frequency components. In this fashion, the temporal context can be internally represented by a combination of different state-frequency basis. Then, for a prediction and generation task, a suitable set of state-frequency components can be selected by memory gates deciding which components should be chosen to predict and generate the target outputs. For example, the high-frequency patterns will be chosen to make very short-term prediction of asset prices, while the low-frequency patterns of price fluctuations will be selected to predict returns in deciding low-term investment. Even more, we also allow the model to automatically adapt its frequency bands over time, resulting in an Adaptive SFM that can change its Fourier bases

[1]University of Central Florida, Orlando, FL, USA. Correspondence to: Guo-Jun Qi <Guojun.Qi@ucf.edu>.

Proceedings of the 34^{th} International Conference on Machine Learning, Sydney, Australia, 2017. JMLR: W&CP.

to more accurately capture the state-frequency components as the dynamics of input sequences evolves.

First we demonstrate the effectiveness of the proposed SFM model by predicting different forms of waves that contain rich periodic signals. We also conduct experiments on several benchmark datasets to model various genres of temporal sequences, showing the applicability of the proposed approach in the real world, non-periodic situations. Our results suggest the SFM can obtain competitive performance as compared with the state-of-the-art models.

The remainder of this paper is organized as follows. Section 2 reviews relevant literature on different RNN-based architectures and their applications. Then we introduce the proposed SFM model in Section 3 with its formal definitions and mathematical analysis. Section 4 demonstrates the experiment results for different evaluation tasks. Finally, we conclude the paper in section 5.

2. Related works

Recurrent Neural Networks (RNNs), which is initially proposed by (Elman, 1990; Jordan, 1997), extend the standard feed forward multilayer perceptron networks by allowing them to accept sequences as inputs and outputs rather than individual observations. In many sequence modeling tasks, data points such as video frames, audio snippets and sentence segments, are usually highly related in time, making RNNs as the indispensable tools for modeling such temporal dependencies. Unfortunately, some research works like (Bengio et al., 1994), has pointed out that training RNNs to capture the long-term dependencies is difficult due to the gradients vanishing or exploding during the back propagation, making the gradient-based optimization struggle.

To overcome the problem, some efforts like (Bengio et al., 2013), (Pascanu et al., 2013) and (Martens & Sutskever, 2011), aim to develop better learning algorithms. While others manage to design more sophisticated structures. The most well-known attempt in this direction is the Long Short-Term Memory (LSTM) unit, which is initially proposed by (Hochreiter & Schmidhuber, 1997). Compared to the vanilla RNN structures, LSTM is granted the capacity of learning long-term temporal dependencies of the input sequences by employing the gate activation mechanisms. In the realistic applications, LSTM has also been proved to be very effective in speech and handwriting recognition (Graves et al., 2005; Graves & Schmidhuber, 2009; Sak et al., 2014). Recently, (Cho et al., 2014) introduce a modification of the conventional LSTM called Gated Recurrent Unit, which combines the the forget and input gates into a single update gate, and also merges the cell state and hidden state to remove the output gate, resulting in a simpler architecture without sacrificing too much performance.

Besides LSTM and its variations, there are a lot of other efforts to improve RNN's sequence modeling ability. For example, Hierarchical RNN (El Hihi & Bengio, 1995) employs multiple RNN layers to model the sequence in different time scale. Moreover, (Schuster & Paliwal, 1997) and (Graves & Schmidhuber, 2005) connect two hidden layers of RNN and LSTM with opposite directions to the same output, respectively. Such bidirectional structures allow the output layer to access information from both past and future states. In addition, Clockwork RNN (Koutnik et al., 2014) and Phased LSTM (Neil et al., 2016), attempt to design new schema to allow updating hidden states asynchronously.

Instead of developing novel structures, many researchers focus on applying existing RNN model to push the boundary of the real-world applications. For example, (Bahdanau et al., 2014) and (Sutskever et al., 2014) have reached the same level performance as the well-developed systems in machine translation with RNN-encoder-decoder framework; (Fernández et al., 2007) proposes the hierarchical Connectionist Temporal Classification (CTC) (Graves et al., 2006) network and its deep variants has achieved the state-of-the-art performance for phoneme recognition on TIMIT database (Graves et al., 2013). Lastly, (Boulanger-lewandowski et al., 2012) reports that RNN yields a better prior for the polyphonic music modeling and transcription.

3. State-Frequency Memory Recurrent Neural Networks

To introduce the State-Frequency Memory (SFM) recurrent neural networks, we begin with the definition of several notations. Suppose we are given a sequence $\mathbf{X}_{1:T} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T]$ of T observations, where each observation belongs to a N-dimensional space, i.e., $\mathbf{x}_t \in \mathbb{R}^N$ for $t = 1, \cdots, T$. Then we use a sequence of memory cells of the same length T to model the dynamics of the input sequence.

Like the conventional LSTM recurrent networks, each memory cell of the SFM contains D-dimensional memory states; however, unlike the LSTM, we decompose these memory states into a set of frequency components, saying $\{\omega_1, \cdots, \omega_K\}$ of K discrete frequencies. This forms a joint state-frequency decomposition to model the temporal context of the input sequence across different states and frequencies. For example, in modeling the human activities, action patterns can be performed at different rates.

For this purpose, we define a SFM matrix $\mathbf{S}_t \in \mathbb{C}^{D \times K}$ at each time t, the rows and columns of which correspond to D dimensional states and K frequencies. This provides us with a finer-grained multi-frequency analysis of memory states by decomposing them into different frequencies, modeling the frequency dependency patterns of input se-

quences.

3.1. Updating State-Frequency Memory

Like in the LSTM, the SFM matrix of a memory cell is updated by combining the past memory and the new input. On the other hand, it should also take into account the decomposition of the memory states into K frequency domains, which can be performed in a Fourier transformation fashion (see Section 3.2 for a detailed analysis) as:

$$\mathbf{S}_t = \mathbf{f}_t \circ \mathbf{S}_{t-1} + (\mathbf{g}_t \circ \mathbf{i}_t) \begin{bmatrix} e^{j\omega_1 t} \\ \cdots \\ e^{j\omega_K t} \end{bmatrix}^T \in \mathbb{C}^{D\times K} \quad (1)$$

where $\circ$ is an element-wise multiplication, $j = \sqrt{-1}$, and $[\cos\omega_1 t + j\sin\omega_1 t, \cdots, \cos\omega_K t + j\sin\omega_K t]$ are Fourier basis of K frequency components for a sliding time window over the state sequence; $\mathbf{f}_t \in \mathbb{R}^{D\times K}$ and $\mathbf{g}_t \in \mathbb{R}^D$ are forget and input gates respectively, controlling what past and current information on states and frequencies are allowed to update the SFM matrix at t. Finally, $\mathbf{i}_t \in \mathbb{R}^D$ is the input modulation that aggregates the current inputs fed into the current memory cell.

The update of SFM matrix $\mathbf{S}_t$ can be decomposed into the real and imaginary parts as follows.

$$\mathrm{Re}\,\mathbf{S}_t = \mathbf{f}_t \circ \mathrm{Re}\,\mathbf{S}_{t-1} + (\mathbf{g}_t \circ \mathbf{i}_t)\,[\cos\omega_1 t, \cdots, \cos\omega_K t] \quad (2)$$

and

$$\mathrm{Im}\,\mathbf{S}_t = \mathbf{f}_t \circ \mathrm{Im}\,\mathbf{S}_{t-1} + (\mathbf{g}_t \circ \mathbf{i}_t)\,[\sin\omega_1 t, \cdots, \sin\omega_K t] \quad (3)$$

Then the amplitude part of $\mathbf{S}_t$ is defined as

$$\mathbf{A}_t = |\mathbf{S}_t| = \sqrt{(\mathrm{Re}\,\mathbf{S}_t)^2 + (\mathrm{Im}\,\mathbf{S}_t)^2} \in \mathbb{R}^{D\times K} \quad (4)$$

where $(\cdot)^2$ denotes element-wise square, each entry $|\mathbf{S}_t|_{d,k}$ captures the amplitude of the dth state on the kth frequency, and the phase of $\mathbf{S}_t$ is

$$\angle\mathbf{S}_t = \arctan(\frac{\mathrm{Im}\,\mathbf{S}_t}{\mathrm{Re}\,\mathbf{S}_t}) \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]^{D\times K} \quad (5)$$

where $\arctan(\cdot)$ is an element-wise inverse tangent function. It is well known that the amplitude and phase encode the magnitude and the shift of each frequency component.

Later, we will feed the amplitude of state-frequency into the memory cell gates, and use the distribution of memory states across different frequencies to control which information should be allowed to update the SFM matrix. The phase $\angle\mathbf{S}_t$ of state-frequency is ignored as we found it does not affect the result in experiments but incurs extra computational and memory overheads.

The Joint State-Frequency Forget Gate

To control the past information, two types of forget gates are defined to decide which state and frequency information can be allowed to update SFM matrix. They are the state forget gate

$$\mathbf{f}_t^{\text{ste}} = \sigma(\mathbf{W}^{\text{ste}}\mathbf{z}_{t-1} + \mathbf{V}^{\text{ste}}\mathbf{x}_t + \mathbf{b}^{\text{ste}}) \in \mathbb{R}^D \quad (6)$$

and the frequency forget gate

$$\mathbf{f}_t^{\text{fre}} = \sigma(\mathbf{W}^{\text{fre}}\mathbf{z}_{t-1} + \mathbf{V}^{\text{fre}}\mathbf{x}_t + \mathbf{b}^{\text{fre}}) \in \mathbb{R}^K \quad (7)$$

where $\sigma(\cdot)$ is an element-wise sigmoid function; $\mathbf{z}_t$ is an output vector which will be discussed later; $\mathbf{W}^*$ and $\mathbf{V}^*$ are weight matrices; and $\mathbf{b}^*$ is a bias vector.

Then a joint state-frequency gate is defined as an outer product $\otimes$ between $\mathbf{f}_t^{\text{ste}}$ and $\mathbf{f}_t^{\text{fre}}$

$$\mathbf{f}_t = \mathbf{f}_t^{\text{ste}} \otimes \mathbf{f}_t^{\text{fre}} = \mathbf{f}_t^{\text{ste}} \cdot \mathbf{f}_t^{\text{fre}'} \in \mathbb{R}^{D\times K} \quad (8)$$

In other words, the joint forget gate is decomposed over states and frequencies to control the information entering the memory cell.

Input Gates and Modulations

The input gate can be defined in the similar fashion as

$$\mathbf{g}_t = \sigma(\mathbf{W}_g\mathbf{z}_{t-1} + \mathbf{V}_g\mathbf{x}_t + \mathbf{b}_g) \in \mathbb{R}^D \quad (9)$$

where the parameter matrices are defined to generate a compatible result for the input gate. The input gate decides how much new information should be allowed to enter the current memory cell to update SFM matrix.

Meanwhile, we can define the following information modulation modeling the incoming observation $\mathbf{x}_t$ as well as the output $\mathbf{z}_{t-1}$ fed into the current memory cell from the last time

$$\mathbf{i}_t = \tanh(\mathbf{W}_i\mathbf{z}_{t-1} + \mathbf{V}_i\mathbf{x}_t + \mathbf{b}_i) \in \mathbb{R}^D \quad (10)$$

which combines $\mathbf{x}_t$ and $\mathbf{z}_{t-1}$.

Multi-Frequency Outputs and Modulations

To obtain the outputs from the SFM, we produce an output from each frequency component, and an aggregated output is generated by combing these multi-frequency outputs modulated with their respective gates.

Specifically, given the amplitude part $\mathbf{A}_t$ of the SFM matrix $\mathbf{S}_t$ at time t, we can produce an output vector $\mathbf{z}_t^k \in \mathbb{R}^M$ for each frequency component k as

$$\mathbf{z}_t^k = \mathbf{o}_t^k \circ f_o(\mathbf{W}_{\mathbf{z}}^k\mathbf{A}_{\mathbf{z}}^k + \mathbf{b}_{\mathbf{z}}^k), \text{for } k = 1, \cdots, K \quad (11)$$

where $\mathbf{A}_t^k \in \mathbb{R}^D$ is the kth column vector of $\mathbf{A}_t$ corresponding to frequency component k, $f_o(\cdot)$ is an output activation function, and $\mathbf{o}_t^k$ is the output gate controlling whether the information on frequency component k should be emitted from the memory cell at time t,

$$\mathbf{o}_t^k = \sigma(\mathbf{U}_\mathbf{o}^k \mathbf{A}_t^k + \mathbf{W}_\mathbf{o}^k \mathbf{z}_{t-1}^k + \mathbf{V}_\mathbf{o}^k \mathbf{x}_t^k + \mathbf{b}_\mathbf{o}^k) \in \mathbb{R}^M \quad (12)$$

where $\mathbf{U}_\mathbf{o}^k$ are weight matrices. These multi-frequency outputs $\{\mathbf{z}_t^k\}$ can be combined to produce an aggregated output vector

$$\mathbf{z}_t = \sum_{k=1}^{K} \mathbf{z}_t^k = \sum_{k=1}^{K} \mathbf{o}_t^k \circ f_o(\mathbf{W}_\mathbf{z}^k \mathbf{A}_\mathbf{z}^k + \mathbf{b}_\mathbf{z}^k) \in \mathbb{R}^M \quad (13)$$

Then $\{\mathbf{o}_t^k | k = 1, \cdots, K\}$ can be explained as the modulators controlling how the multi-frequency information is combined to yield the output.

3.2. Fourier Analysis of SFM Matrices

Now we can expand the update equation for the SFM matrix to reveal its temporal structure by induction over t. By iterating the SFM matrix $\mathbf{S}_t$ over the time t, Eq. 1 can be written into the following final formulation

$$\begin{aligned}\mathbf{S}_t = (\mathbf{f}_t \circ \mathbf{f}_{t-1} \circ \cdots \circ \mathbf{f}_1) \circ \mathbf{S}_0 + (\mathbf{g}_t \circ \mathbf{i}_t) \begin{bmatrix} e^{j\omega_1 t} \\ \cdots \\ e^{j\omega_K t} \end{bmatrix}^T \\ + \sum_{t'=2}^{t} \mathbf{f}_t \circ \cdots \circ \mathbf{f}_{t'} \circ \mathbf{g}_{t'-1} \circ \mathbf{i}_{t'-1} \begin{bmatrix} e^{j\omega_1 (t'-1)} \\ \cdots \\ e^{j\omega_K (t'-1)} \end{bmatrix}^T \end{aligned} \quad (14)$$

where $\mathbf{S}_0$ is the initial SFM matrix at time 0.

By this expansion, it is clear that $\mathbf{S}_t$ is the Fourier transform of the following sequence

$$\begin{aligned}&\{(\mathbf{f}_t \circ \mathbf{f}_{t-1} \circ \cdots \circ \mathbf{f}_1) \circ \mathbf{S}_0\} \cup \{\mathbf{g}_t \circ \mathbf{i}_t\} \\ &\quad \cup \{\mathbf{f}_t \circ \cdots \circ \mathbf{f}_{t'} \circ \mathbf{g}_{t'-1} \circ \mathbf{i}_{t'-1} | t' = 2, \cdots, t\}\end{aligned} \quad (15)$$

In this sequence,the forget and input gates $\mathbf{f}_{t'}$ and $\mathbf{g}_{t'}$ weigh the input modulations $\mathbf{i}_{t'}$ that aggregates the input information from the observation and past output sequences. In other words, the purpose of these gates is to control how far the Fourier transform should be applied into the past input modulations.

If some forget or input gate has a relatively small value, the far-past input modulations would be less involved in constituting the frequency components in the current $\mathbf{S}_t$. This tends to localize the application of Fourier transform in a short time window prior to the current moment. Otherwise, a longer time window would be defined to apply the Fourier transform. Therefore, the forget and input gate dynamically define time windows to control the range of temporal contexts for performing the frequency decomposition of memory states by the Fourier transform.

3.3. Adaptive SFM

The set of frequencies $\{\omega_1, \cdots, \omega_K\}$ can be set to $\omega_k = \frac{2\pi k}{K}$ for $k = 0, \cdots, K-1$, i.e., a set of K discrete frequencies evenly spaced on $[0, 2\pi]$. By Eq. 14, this results in the classical Discrete-Time Fourier Transform (DTFT), yielding K frequency coefficients stored column-wise in SFM matrix for each of D memory states.

Alternatively, we can avoid prefixing the discrete frequencies $\boldsymbol{\omega} = [\omega_1, \cdots, \omega_K]^T$ by treating them as variables that can be dynamically adapted to the context of the underlying sequence. In other words, $\boldsymbol{\omega}$ is not a static frequency vector with fixed values any more; instead they will change over time and across different sequences, reflecting the changing frequency of patterns captured by the memory states. For example, a certain human action (modeled as a memory state) can be performed at various execution rates changing over time or across different actors. This inspires us to model the frequencies as a function of the memory states, as well as the input and output sequences,

$$\boldsymbol{\omega} = \mathbf{W}_{\omega x} \mathbf{x}_t + \mathbf{W}_{\omega z} \mathbf{z}_{t-1} + \mathbf{b}_\omega \quad (16)$$

where $\mathbf{W}_*$ and $\mathbf{b}_\omega$ are the function parameters, and multiplying 2π with a sigmoid function maps each ω_k onto $[0, 2\pi]$. This makes the SFM more flexible in capturing dynamic patterns of changing frequencies. We call the Adaptive SFM (A-SFM) for brevity.

4. Experiments

In this section, we demonstrate the evaluation results of the SFM on three different tasks: signal type prediction, polyphonic music modeling and phoneme classification. For all the tasks, we divide the baselines into two groups: the first one (BG1) contains classic RNN models such as the conventional LSTM (Hochreiter & Schmidhuber, 1997) and the Gated Recurrent Unit (GRU) (Cho et al., 2014); while the second group (BG2) includes latest models like Clockwork RNN (CW-RNN) (Koutnik et al., 2014), Recurrent Highway Network (RHN) (Zilly et al., 2016), Adaptive Computation Time RNNs (ACT-RNN) (Graves, 2016), Associative LSTM (A-LSTM) (Danihelka et al., 2016) and Phased LSTM (P-LSTM) (Neil et al., 2016). Compared to the proposed SFM, baselines in BG2 share similarities like hierarchical structures and complex representation. However, they either don't contain any modules aiming to learn frequency dependencies (RHN, ACT-RNN, A-LSTM), or only have implicit frequency modeling abilities that are not enough to capture the underlying frequency patterns in the

Table 1. Hidden neuron numbers of different networks for each task. The total number of parameters (weights) will keep the same for all the networks in each task. Task 1, 2 and 3 stand for signal type prediction (sec 4.1), polyphonic music prediction (sec 4.2) and phone classification (sec 4.3), respectively. The last column indicates the unique hyperparameters of each network.

	Task 1	Task 2	Task 3	Note
# of Params	$\approx$ 1k	$\approx$ 139k	$\approx$ 80k	-
GRU	18	164	141	-
LSTM	15	139	122	-
CW-RNN	30	295	245	$T_n \in \mathcal{T}$[1]
ACT-RNN	18	164	141	-
RHN	9	94	76	$d = 4$[1]
A-LSTM	15	139	122	$n = 4$[1]
P-LSTM	15	139	122	$r_{on} = 0.05$[1]
SFM	50×4	50×4	30×8	-

input sequences (CW-RNN, P-LSTM).

In order to make a fair comparison, we use different numbers of hidden neurons for these networks to make sure the total numbers of their parameters are approximately the same. Per the discussion in Section 3, the hidden neuron number is decided by the size $D \times K$ of the SFM matrix $\mathbf{S}_t \in \mathbb{C}^{D \times K}$, where D stands for the dimension of the memory states and K is the number of frequency components. The hidden neuron setups for three tasks are summarized in Table 1. We implement the proposed SFM model using Theano Python Math Library (Team et al., 2016).

Unless otherwise specified, we train all the networks through the BPTT algorithm with the AdaDelta optimizer (Zeiler, 2012), where the decay rate is set to 0.95. All the weights are randomly initialized in the range $[-0.1, 0.1]$ and the learning rate is set to 10^{-4}. The training objective is to minimize the frame level cross-entropy loss.

4.1. Signal Type Prediction

First, we generate some toy examples of sequences, and apply the SFM to distinguish between different signal types. In particular, all signal waves are periodic but contain different frequency components. Without loss of generality, we choose to recognize two types of signals: square waves and sawtooth waves. By the Fourier analysis, these two types of waves can be represented as the following Fourier series, respectively.

$$y_{square}(t) = \frac{4A}{\pi} \sum_{n=1,3,5,\cdots}^{\infty} \frac{1}{n} \sin(\frac{2n\pi}{T}(t+P)) + V, t \in [0, L] \tag{17}$$

[1] T_n - clock period, $\mathcal{T} = \{2^0, \cdots, 2^4\}$; d - recurrent depth; n - # of copies; r_{on} - open ratio. Please refer to each baseline paper for more details.

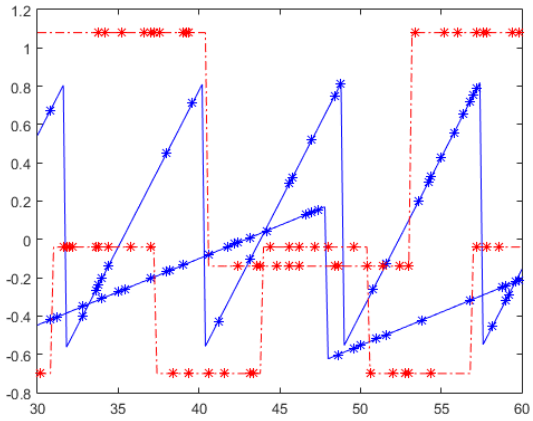


Figure 1. Several examples of the generated waves on the interval $[30, 60]$ with different periods, amplitudes, and phases. The red dash lines represent the square waves while the blue solid lines represent square waves. The '$*$' markers indicate the sampled data points that are used for training and testing.

$$y_{sawtooth}(t) = \frac{A}{2} - \frac{A}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin(\frac{2n\pi}{T}(t+P)) + V, t \in [0, L] \tag{18}$$

where L, T, A, P, V stand for the length, period, amplitude, phase and bias, respectively. The square waves contain the sine base functions only with the odd n's, while sawtooth waves contain both the odd and even n's. This makes their frequency components quite different from each other, and thus they are good examples to test the modeling ability of the proposed SFM network.

We artificially generate $2,000$ sequences, with $1,000$ sequences per signal type. Figure 1 illustrates the generated waves samples. Our goal is to correctly classify each of the blue solid and red dash signals. We denote by $\mathcal{U}(a, b)$ a uniform distribution on $[a, b]$, then the sequence of each wave is generated like this: we first decide the length of the wave $L \sim \mathcal{U}(15, 125)$ and its period $T \sim \mathcal{U}(50, 75)$. Then we choose amplitude $A \sim \mathcal{U}(0.5, 2)$, phase $P \sim \mathcal{U}(0, 15)$ and the bias $V \sim \mathcal{U}(0.25, 0.75)$. At the last step, we randomly sample each signal 500 times along with their time stamps, resulting in a sequence of 2-dimensional vectors. Specifically, a vector in a sequence has the form of (y, t), where y is the signal value and t is the corresponding time stamp. In experiments, we randomly select 800 sequences per type for training and the remaining are for testing.

We compare the proposed SFM with all BG1 and BG2 baselines and summarize the prediction results in Figure 2. The result shows by explicitly modeling the frequency patterns of these two types of sequences, both SFM and

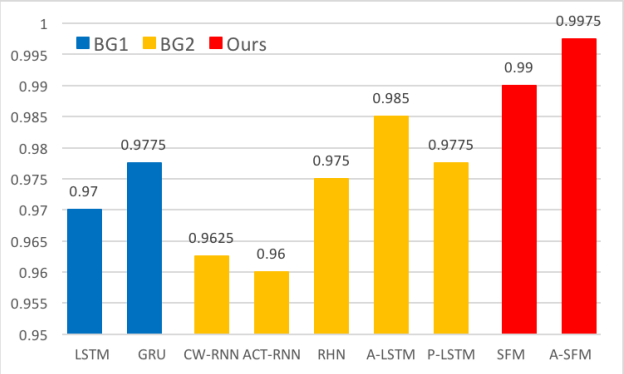


Figure 2. Signal type prediction accuracy of each model.

Adaptive SFM achieve best performance. In particular, the Adaptive SFM has achieved 0.9975 in accuracy – almost every signal has been classified correctly.

4.2. Polyphonic Music Modeling

In this subsection, we evaluate the proposed SFM network on modeling polyphonic music clips. The task focuses on modeling the symbolic music sequences in the piano-roll form like in (Boulanger-lewandowski et al., 2012). Specifically, each piano roll can be regarded as a matrix with each column being a binary vector that represents which keys are pressed simultaneously at a particular moment. This task of modeling polyphonic music is to predict the probability of individual keys being pressed at next time $t+1$ given the previous keys pressed in a piano roll by capturing their temporal dependencies. Such a prediction model plays a critical role in polyphonic transcription to estimate the audible note pitches from acoustic music signals.

The experimental results are obtained on four polyphonic music benchmarks that have been used in (Boulanger-lewandowski et al., 2012): MuseData, JSB chorales (Allan & Williams, 2004), Piano-midi.de (Poliner & Ellis, 2007) and Nottingham. In addition to the baselines in BG1 and BG2, we also compare with the following methods that have achieved the best performance in (Boulanger-lewandowski et al., 2012).

- **RNN-RBM:** Proposed by (Boulanger-lewandowski et al., 2012), RNN-RBM is a modification of RTRBM (Sutskever et al., 2009) by combining a full RNN with the Restrict Boltzmann Machine.
- **RNN-NADE-HF:** RNN-NADE (Larochelle & Murray, 2011) model with the RNN layer pretrained by the Hessian-free (HF) (Martens & Sutskever, 2011)

We directly report the results of these methods from (Boulanger-lewandowski et al., 2012).

We follow the same protocol (Boulanger-lewandowski et al., 2012) to split the training and test set to make a fair comparison on the four datasets. The MIDI format music files are publicly available[2] and have been preprocessed into piano-roll sequences with 88 dimensions that span the range of piano from A0 to C8. Then, given a set of N piano-roll sequences and $\mathbf{X}^n = [\mathbf{x}_1^n, \cdots, \mathbf{x}_{T_n}^n]$ is the nth sequence of length T_n, the SFM uses a softmax output layer to predict the probability of each key being pressed at time t based on the previous ones $\mathbf{x}_{1:t-1}$.

The log-likelihood of correctly predicting keys to be pressed has been used as the evaluation metric in (Boulanger-lewandowski et al., 2012), and we adopt it to make a direct comparison across the models. The results are reported in the Table 2. As it indicates, both SFM and Adaptive SFM have outperformed the state-of-the-art baselines except on the Nottingham dataset. On the rest of three datasets, both SFM and Adaptive SFM consistently perform $0.2 \sim 1.0$ better than BG1 and BG2 baselines in terms of the log-likelihood. We also note that the Adaptive SFM obtains almost the same result as the SFM, suggesting that using static frequencies are already good enough to model the polyphony.

On the Nottingham dataset, however, the two compared models, RNN-RBM and RNN-NADE-HF, reach the best performance. The dataset consists of over 1000 folk tunes, which are often composed of simple rhythms with few chords. Figure 3 compares some example music clips from the Nottingham to the MuseData datasets. Clearly, the Nottingham music only contains simple polyphony patterns that can be well modeled with the RNN-type models without having to capture complex temporal dependencies. On the contrary, the music in the MuseData is often a mixture of rich frequency components with long-range temporal dependencies, which the proposed SFM models are better at modeling as shown in the experiment results.

4.3. Phoneme Classification

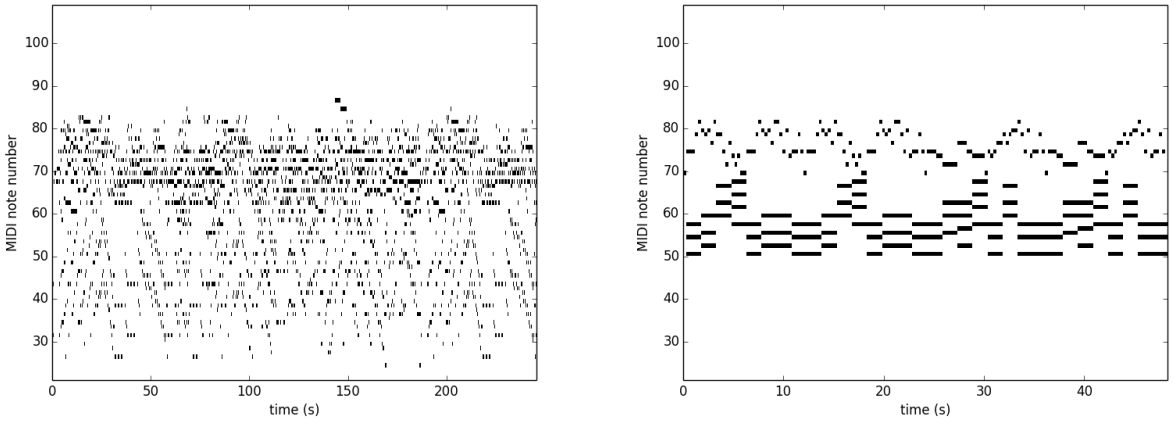
Finally, we evaluate the proposed SFM model by conducting the frame level phoneme classification task introduced by (Graves & Schmidhuber, 2005). The goal is to assign the correct phoneme to each speech frame of an input sequence. Compared with other speech recognition tasks like spoken word recognition, phoneme classification focuses on identifying short-range sound units (phonemes) rather than long-range units (words) from input audio signals. We report the frame-level classification accuracy as the evaluation metric for this task.

We perform the classification task on TIMIT speech cor-

[2] http://www-etud.iro.umontreal.ca/∼boulanni/icml2012

Table 2. Log-likelihood on the four music datasets. The last two columns contain the results from the proposed SFM models.

Dataset	LSTM	GRU	CW-RNN	ACT-RNN	RHN	A-LSTM	P-LSTM	RNN-RBM	RNN-NADE-HF	SFM	A-SFM
MuseData	−5.44	−5.36	−5.35	−5.20	−5.79	−5.03	−5.09	−6.01	−5.60	−4.81	**−4.80**
JSB chorales	−6.24	−6.14	−6.04	−5.89	−5.74	−5.63	−5.65	−6.27	−5.56	−5.47	**−5.45**
Piano-midi.de	−7.27	−7.14	−7.83	−7.41	−7.58	−6.96	−7.02	−7.09	−7.05	**−6.76**	−6.80
Nottingham	−5.60	−5.63	−5.90	−5.82	−5.60	−5.64	−5.70	−2.39	**−2.31**	−5.67	−5.63



(a) MuseData (b) Nottingham

Figure 3. Piano rolls of the exemplar music clips from the MuseData and Nottingham dataset. Classical musics from MuseData are presented by complex, high frequently-changed sequences, while folk tunes from Nottingham contains simpler, lower-frequency sequences.

pus (Garofolo et al., 1993). We preprocess the dataset in the same way as (Graves & Schmidhuber, 2005). First we perform Short-time Fourier Transform (STFT) with 25ms input windows and 10ms frame size. Then for each frame, we compute the Mel-Frequency Cepstrum Coefficients (MFCCs), the log-energy and its first-order derivatives as the frame-level features. Similarly, in order to maintain the consistency with (Graves & Schmidhuber, 2005), we use the original phone set of 61 phonemes instead of mapping them into a smaller set (Robinson, 1991).

We train the proposed and compared models by following the standard splitting of training and test sets for the TIMIT dataset (Garofolo et al., 1993). In addition, we randomly select 184 utterances from the training set as the validation set and keep the rest for training.

Figure 4 compares the classification accuracy by different models. In addition, we include the result of the bidirectional LSTM (Bi-LSTM) (Graves & Schmidhuber, 2005) for comparison. From the figure, we can see that both SFM and Adaptive SFM outperform the BG1 and BG2 baselines with approximately the same number of parameters. Especially when compared with the conventional LSTM, CW-RNN and ACT-RNN, the proposed SFM models have significantly improved the performance by more

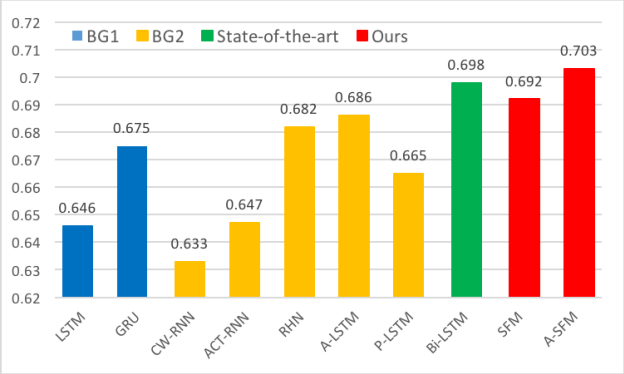
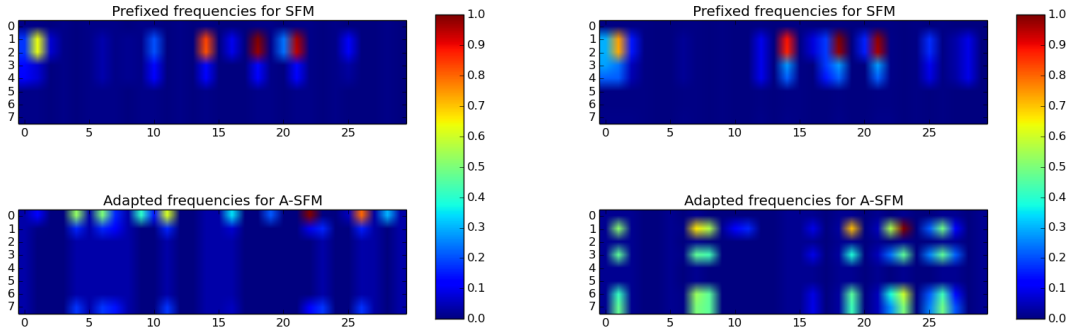


Figure 4. Accuracy for frame-level phoneme classification on TIMIT dataset.

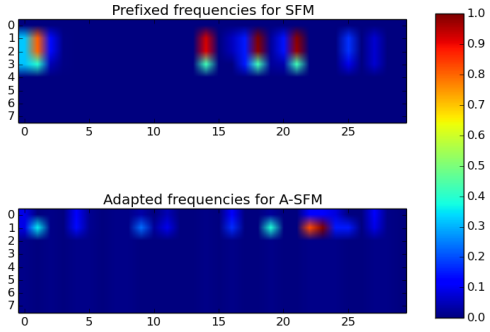
than 5%. This demonstrates the advantages on explicitly modeling the frequency patterns in short-range windows, which plays a important role in characterizing the frame-level phoneme. Besides, the Adaptive SFM also perform slightly better than the state-of-the-art Bi-LSTM model.

Additionally, we find adapting frequencies, which is the main difference between the Adaptive SFM and SFM,

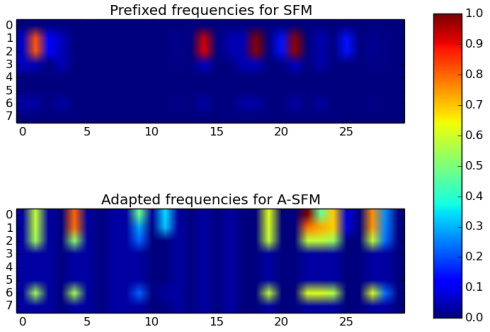


(a) At $\frac{T}{4}$

(b) At $\frac{T}{2}$



(c) At $\frac{3T}{4}$



(d) At T

Figure 5. The amplitudes of SFM matrices for both the prefixed (SFM) and adaptive (A-SFM) frequencies. For all subfigures, each row represents a frequency component.

yields improved performance on this dataset. In order to further analyze such difference, we visualize the SFM matrices of both the SFM and Adaptive SFM by forwarding a sampled TIMIT sequence. Suppose the length of the sampled sequence is T, Figure 5 illustrates the matrix amplitudes of both the SFM and Adaptive SFM at time $\frac{T}{4}$, $\frac{T}{2}$, $\frac{3T}{4}$ and L, from which the two networks demonstrate distinct ways to model the sequence. Based on section 3.3, the frequency set of the SFM keeps the same across the time. And in all subfigures of Figure 5, most highlights are around frequency component 1 and 2, indicating the two are the primary components for modeling the sequence, while other components are rarely involved. On the contrary, the frequency set of Adaptive SFM is constantly updated and different at each time. Under such conditions, modeling the sequence calls for more frequency components, which are varied dramatically across the time in Figure 5. Compared with the SFM, Adaptive SFM is able to model richer frequency patterns.

5. Conclusion

In this paper, we propose a novel State-Frequency Memory (SFM) Recurrent Neural Network which aims to model the frequency patterns of the temporal sequences. The key idea of the SFM is to decompose the memory states into different frequency states such that they can explicitly learn the dependencies of both the low and high frequency patterns. These learned patterns on different frequency scales can be separately transferred into the output vectors and then aggregated to represent the sequence point at each time step. Compared to the conventional LSTM, the proposed SFM is more powerful in discovering different frequency occurrences, which are important to predict or track the temporal sequences at various frequencies. We evaluate the proposed SFM model with three sequence modeling tasks. Our experimental results show the proposed SFM model can outperform various classic and latest LSTM models as well as reaching the competitive performance compared to the state-of-the-art methods on each benchmark.

References

Allan, Moray and Williams, Christopher KI. Harmonising chorales by probabilistic inference. In *NIPS*, pp. 25–32, 2004.

Bahdanau, Dzmitry, Cho, Kyunghyun, and Bengio, Yoshua. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

Bengio, Yoshua, Simard, Patrice, and Frasconi, Paolo. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2): 157–166, 1994.

Bengio, Yoshua, Boulanger-Lewandowski, Nicolas, and Pascanu, Razvan. Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pp. 8624–8628. IEEE, 2013.

Boulanger-lewandowski, Nicolas, Bengio, Yoshua, and Vincent, Pascal. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. In *Proceedings of the 29th International Conference on Machine Learning (ICML-12)*, pp. 1159–1166, 2012.

Cho, Kyunghyun, Van Merriënboer, Bart, Gulcehre, Caglar, Bahdanau, Dzmitry, Bougares, Fethi, Schwenk, Holger, and Bengio, Yoshua. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

Danihelka, Ivo, Wayne, Greg, Uria, Benigno, Kalchbrenner, Nal, and Graves, Alex. Associative long short-term memory. In *Proceedings of The 33rd International Conference on Machine Learning*, pp. 1986–1994, 2016.

El Hihi, Salah and Bengio, Yoshua. Hierarchical recurrent neural networks for long-term dependencies. In *Nips*, volume 409, 1995.

Elman, Jeffrey L. Finding structure in time. *Cognitive science*, 14(2):179–211, 1990.

Fernández, Santiago, Graves, Alex, and Schmidhuber, Jürgen. Sequence labelling in structured domains with hierarchical recurrent neural networks. In *IJCAI*, pp. 774–779, 2007.

Garofolo, John S, Lamel, Lori F, Fisher, William M, Fiscus, Jonathon G, and Pallett, David S. Darpa timit acoustic-phonetic continous speech corpus cd-rom. *NASA STI/Recon technical report n*, 93, 1993.

Graves, Alex. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

Graves, Alex and Schmidhuber, Jürgen. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5): 602–610, 2005.

Graves, Alex and Schmidhuber, Jürgen. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pp. 545–552, 2009.

Graves, Alex, Beringer, Nicole, and Schmidhuber, Juergen. Rapid retraining on speech data with lstm recurrent networks. *Technical Report IDSIA-09-05, IDSIA*, 2005.

Graves, Alex, Fernández, Santiago, Gomez, Faustino, and Schmidhuber, Jürgen. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pp. 369–376. ACM, 2006.

Graves, Alex, Mohamed, Abdel-rahman, and Hinton, Geoffrey. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pp. 6645–6649. IEEE, 2013.

Hochreiter, Sepp and Schmidhuber, Jürgen. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

Jordan, Michael I. Serial order: A parallel distributed processing approach. *Advances in psychology*, 121:471–495, 1997.

Koutnik, Jan, Greff, Klaus, Gomez, Faustino, and Schmidhuber, Juergen. A clockwork rnn. In *Proceedings of The 31st International Conference on Machine Learning*, pp. 1863–1871, 2014.

Larochelle, Hugo and Murray, Iain. The neural autoregressive distribution estimator. In *AISTATS*, volume 1, pp. 2, 2011.

Lipton, Zachary C, Berkowitz, John, and Elkan, Charles. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

Martens, James and Sutskever, Ilya. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pp. 1033–1040, 2011.

Mikolov, Tomas, Karafiát, Martin, Burget, Lukas, Cernockỳ, Jan, and Khudanpur, Sanjeev. Recurrent neural network based language model. In *Interspeech*, volume 2, pp. 3, 2010.

Mittelman, Roni. Time-series modeling with undecimated fully convolutional neural networks. *arXiv preprint arXiv:1508.00317*, 2015.

Neil, Daniel, Pfeiffer, Michael, and Liu, Shih-Chii. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pp. 3882–3890, 2016.

Pascanu, Razvan, Mikolov, Tomas, and Bengio, Yoshua. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

Poliner, Graham E and Ellis, Daniel PW. A discriminative model for polyphonic piano transcription. *EURASIP Journal on Applied Signal Processing*, 2007(1):154–154, 2007.

Robinson, Tony. *Several improvements to a recurrent error propagation network phone recognition system*. 1991.

Sak, Haşim, Senior, Andrew, and Beaufays, Françoise. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

Schuster, Mike and Paliwal, Kuldip K. Bidirectional recurrent neural networks. *IEEE Transactions on Signal Processing*, 45(11):2673–2681, 1997.

Sutskever, Ilya, Hinton, Geoffrey E, and Taylor, Graham W. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*, pp. 1601–1608, 2009.

Sutskever, Ilya, Vinyals, Oriol, and Le, Quoc V. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pp. 3104–3112, 2014.

Team, The Theano Development, Al-Rfou, Rami, Alain, Guillaume, Almahairi, Amjad, Angermueller, Christof, Bahdanau, Dzmitry, Ballas, Nicolas, Bastien, Frédéric, Bayer, Justin, Belikov, Anatoly, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

Zeiler, Matthew D. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

Zilly, Julian Georg, Srivastava, Rupesh Kumar, Koutník, Jan, and Schmidhuber, Jürgen. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.