2019

Leaning Robust Sequence Features via Dynamic Temporal Pattern Discovery

Hao Hu
University of Central Florida

LEARNING ROBUST SEQUENCE FEATURES VIA DYNAMIC TEMPORAL PATTERN DISCOVERY

by

HAO HU
M.S. University of Central Florida, 2014
B.S. Nankai University, 2010

A dissertation submitted in partial fulfilment of the requirements
for the degree of Doctor of Philosophy
in the Department of Computer Science
in the College of Engineering and Computer Science
at the University of Central Florida
Orlando, Florida

Spring Term
2019

Major Professor: Liqiang Wang

ABSTRACT

As a major type of data, time series possess invaluable latent knowledge for describing the real world and human society. In order to improve the ability of intelligent systems for understanding the world and people, it is critical to design sophisticated machine learning algorithms for extracting robust time series features from such latent knowledge. Motivated by the successful applications of deep learning in computer vision, more and more machine learning researchers put their attentions on the topic of applying deep learning techniques to time series data. However, directly employing current deep models in most time series domains could be problematic. A major reason is that temporal pattern types that current deep models are aiming at are very limited, which cannot meet the requirement of modeling different underlying patterns of data coming from various sources. In this study we address this problem by designing different network structures explicitly based on specific domain knowledge such that we can extract features via most salient temporal patterns. More specifically, we mainly focus on two types of temporal patterns: order patterns and frequency patterns. For order patterns, which are usually related to brain and human activities, we design a hashing-based neural network layer to globally encode the ordinal pattern information into the resultant features. It is further generalized into a specially designed Recurrent Neural Networks (RNN) cell which can learn order patterns in an online fashion. On the other hand, we believe audio-related data such as music and speech can benefit from modeling frequency patterns. Thus, we do so by developing two types of RNN cells. The first type tries to directly learn the long-term dependencies on frequency domain rather than time domain. The second one aims to dynamically filter out the "noise" frequencies based on temporal contexts. By proposing various deep models based on different domain knowledge and evaluating them on extensive time series tasks, we hope this work can provide inspirations for others and increase the community's interests on the problem of applying deep learning techniques to more time series tasks.

To my family.

ACKNOWLEDGMENTS

I would like to thank my academic advisors Dr. Guo-Jun Qi and Dr. Liqiang Wang for all their academic guidance during my PhD career. It is Dr. Qi who led me into the field of deep learning research, and Dr. Wang helped me prepare this dissertation. Part of works in this dissertation were done with my former lab mates Dr. Jun Ye and Joey Velez-Ginorio. Thank you for your contributions and I enjoyed our collaborations. I have spent two summers in California, working closely with Dr. Zhaowen Wang, and Dr. Qiong Liu, respectively. I learned a lot on how to do great research from them, and it is my honor to have worked with them. Besides, I also want to thank Dr. Shaojie Zhang, Dr. Fei Liu and Dr. Qun Zhou to be my committee members. Thank all of my family for supporting me when I encountered difficulties in my life. It is impossible for me to write this dissertation without you.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1: INTRODUCTION

Time series is a fundamental existing form for a large amount of data in the real world. Every day, tones of sequential data is generated through the daily activities of both people and machines: visitors and tourists post traveling videos on the Internet to share their journey with audiences around the world; doctors collect health monitoring data from various medical devices in order to make correct treatment decisions; and sensors deployed on precisional manufacturing instruments continuously transmit surveillance sequences for quality analysis such that there is little chance for creating defective products. Thus, times series have been a part of our lives today, and usually contain massive hidden information that can reflect essential patterns for understanding and solving realistic problems. Due to the fact that modern people need to frequently interact with many types of time series such as music, videos, speech and sensor readings, it is important to build reliable intelligent processing systems that can improve the efficiency of such interactions. A promising way is to design machine learning algorithms for extracting time series features, which can be considered as abstract representations of original data that are easier to analyze. The quality of features can vary dramatically according to different machine learning models, it is important to develop the sophisticated models to acquire robust features for time series. Currently, the most popular machine learning models for time series are Deep Neural Networks (DNN), the core technique of the deep learning.

1.1 Deep Neural Networks

Recently, inspired by the advancement in computer vision, which is fueled by the development of deep Convolutional Neural Networks (CNN), topics on applying deep learning techniques to time series data are drawing more and more attention. As shown in Figure 1.1, CNN, a family of

specially designed DNN that inspired by the human visual perception system, usually consists of a set of stacked convolutional layers and fully-connected layers. For each convolutional layer, a set of kernels make convolution with a small neighborhood region at each location and slide along the dimensions of the input. The convolved outputs are further activated by non-linearity functions and less salient outputs are discard for reducing the size. Thus, CNN can generate hierarchical features and different level. Since CNN simulates the human visual system, it works well for visual-related tasks such as image recognition and segmentation etc [116, 56].

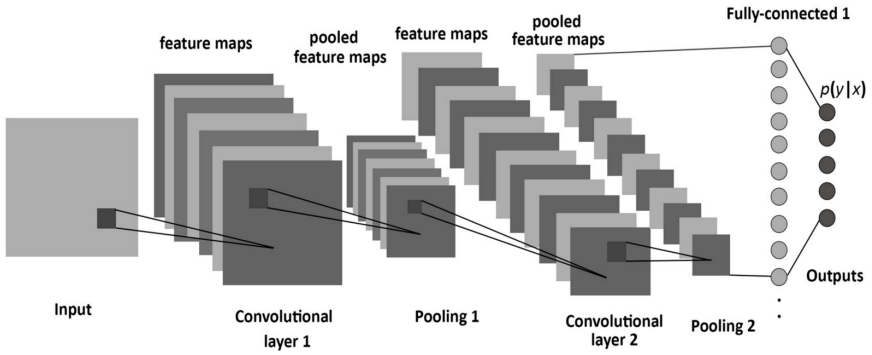


Figure 1.1: The architecture of Convolutional Neural Networks.

Comparing to vision tasks, learning to extract high quality features from time series is more challenging, as the deep models need to consider various additional temporal patterns. Instead of CNN, Recurrent Neural Network (RNN) becomes a much more suitable choice for times series. Unlike CNN, whose nodes are all connected to other nodes, part of RNN nodes are connected to themselves and can be updated by their previous states and current inputs. These self-connections can be further unrolled along time to generate a deep forward structure without spawning inner circles. In this fashion, the nodes of RNN are connected to form a directed graph that can be aligned with a temporal sequences. Such unrolling architecture provides natural neural structures for temporal pattern modeling. Some of specially designed RNN cells, like Long-Short Term Memory (LSTM) [57], have achieved impressive performance gains on several natural language processing (NLP)

tasks, demonstrating the potential power of modeling time series through RNNs.

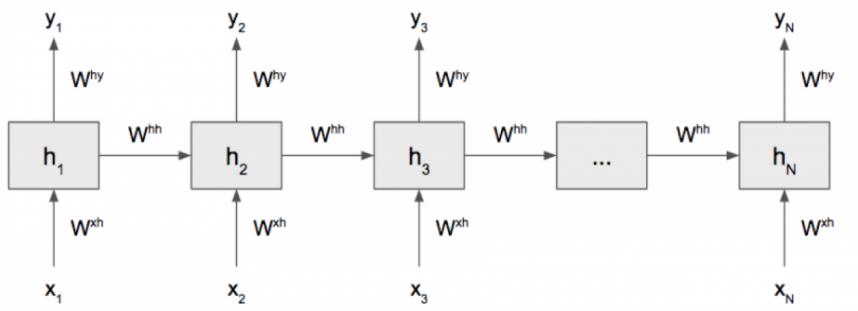


Figure 1.2: The unrolling architecture of Recurrent Neural Networks.

1.2 Challenges and Research Objectives

However, it is still very struggle to populate RNN models from NLP to a wider range of application domains such as signal processing and music modeling, etc., since most of such attempts didn't yield superior performance gain comparing with traditional state-of-the-art, non deep learning-based methods. The reasons behind this phenomenon are twofold: first, based on the experience from computer vision and NLP, large-scale datasets with labels are indispensable for training deep neural networks; while it is impractical to collect so much data and accurately label them in other application tasks. For example, commonly used NLP benchmarks like Penn Treebank dataset [84] includes over one million English words, on the contrary, the GTZAN dataset [131], a popular benchmarks for music genre classification, only contains one thousand music tracks, whose label are hard to determine due to various music tastes of people. This makes training RNN even harder for music modeling-related tasks. Moreover, the pattern types that current RNN models are aiming to model are also limited: most of gated structures target on the long-term dependency modeling, which might not be a critical consideration for certain data like signals.

In order to handle such issues, revisiting the well-studied techniques for individual domain could be helpful to find out new perspectives for training better deep learning models. In this study, we are driven to explore the potential ways to develop hybrid solutions for deep learning of general-purposed time series, which are implemented by incorporating the domain-specific knowledge with the purely data-driven RNN models. This is motivated by the fact that domain-specific knowledge can serve as the constraint rules to compensate the shortage of well-labeled data, and different underlying patterns characterize time series accordingly based on different tasks. We are particularly interested in two types of temporal patterns: The order pattern and the frequency pattern, as they are commonly measured in a lot of sequential modeling tasks. For example, we may differentiate actions via the ordinal relationships between their subactions; and traditional signal processing techniques tell us the frequency patterns could be critical for analyzing signal data. More specifically, we explicitly encode the order patterns by developing a hashing-based layer and a temporal sensitive RNN structure. These two approaches are employed to solve the real world applications of brain disorder diagnosis and video representation learning, respectively. On the other hand, we propose two different RNN variants to integrate the frequency pattern modeling with the deep learning models, and evaluate them with a bunch of sequential modeling tasks across various domains. Our evaluation results provide positive evidence to support the combination of domain-specific patterns and data-driven deep models as a promising strategy for a wider range of time series modeling tasks.

The rest of this dissertation is organized as follows: Chapter 2 describes the details of First-Take-All (FTA), a hashing-based deep learning layer to encode order patterns for brain disorder diagnosis; then TPRNN, another order-based RNN design for video representation learning will be introduced in Chapter 3. Chapter 4 and Chapter 5 reveal the details of two frequency pattern-related RNN variants, which are named as State-Frequency Memory (SFM) and Adaptively Scaled RNN (ASRNN), respectively. Finally, Chapter 6 concludes the dissertation.

CHAPTER 2: MINING ORDER PATTERNS FOR BRAIN DISORDER DIAGNOSIS

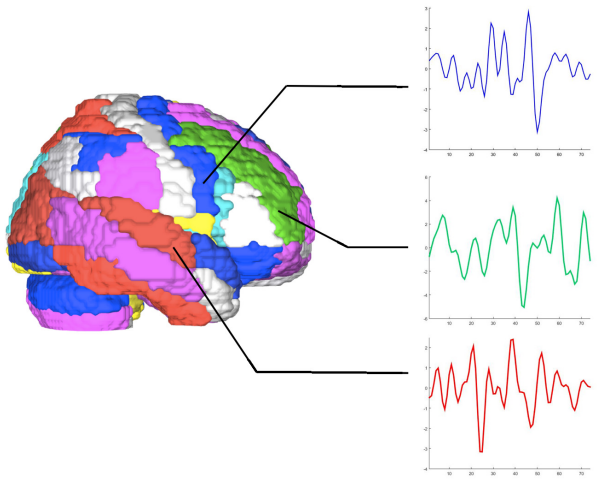


Figure 2.1: ROIs (AAL[132]) of a human brain and their fMRI time courses. FTA hashes time courses into fixed-size hash codes by encoding temporal order differences between latent patterns among them.

In this chapter, we present a new hashing-based learning algorithm by explicitly encoding order patterns for brain disorder diagnosis. Recent advancements in brain imaging technology, one of the greatest efforts in Neuroscience, aim to uncover features unique to certain neurophysiological phenomena [13, 71]. The intuition is that the neural activity pattern of a healthy human subject ought to appear different from that of a patient suffering from some neural disorder, such as Attention Deficit Hyperactive Disorder (ADHD) and Alzheimer's disease.

Categorized alongside other neurodevelopmental disorders, ADHD is one of the most common brain diseases. It manifests early and is typically first diagnosed in one's childhood. The predominant effects consist of sustained difficulty in maintaining focus, often offering difficulties

assimilating at school, at home, or within the community. Fortunately, despite lacking a cure, brain imaging technology provides an opportunity to timely observe and diagnose ADHD[23, 24].

Several methods for recording sequences of neural activity from the brain exist. Generally, they range from invasive to non-invasive procedures. Whilst the data from invasive techniques such as Electrocorticography (ECoG) or multielectrode arrays possess higher quality data [94], the opportunity to collect such data seldom arises. By nature of the procedure, it is much more practical for researchers and medical practitioners to opt for non-invasive techniques such as functional Magnetic Resonance Imaging (fMRI). fMRI indirectly measures changes in neural activity by detecting changes in blood flow caused by increased activations of neurons during specific tasks (or resting-state conditions) [104, 26]. In this context, fMRI time-courses [1] offer a feature-rich representation of high level functional organization in the brain (Figure 2.1).

Considering the representations, we aim to exploit its rich nature for the task of ADHD diagnosis as a pattern classification problem [79]. Utilizing the structure of the resting-state fMRI time-courses, we generate hash codes encoding the temporal structure of the data. These hash codes can then be compared to detect similarities and differences between the fMRI time-courses of healthy patients versus those diagnosed with ADHD. It is known that neural connections exist whether or not regions of the brain are functionally active, hence forth the resting-state fMRI provides a controlled dataset to test for fundamental differences in functional neural networks in the brain.

Tasked with hashing time-series data, our approach centers on fast detection based on retrieval of the similar disorder patterns from a database of brain neural imaging activities. Specifically, we propose the First-Take-All (FTA) hashing method for encoding varied-length fMRI sequences into fixed-size hash codes. The problem of fast matching similar fMRI sequences boils down to a fast

[1] In medical imaging terminology, a "time course" refers to an obtained sequence for an imaged area. In this chapter, we will use this term interchangeably with "time series" when there is no confusion in the context.

search of similar hash codes based on their Hamming distances that can be calculated efficiently.

Specifically, the algorithm first projects an input sequence of varied length onto different subspaces, each representing a sequence of latent patterns. After encoding the temporal order of these patterns to hash the fMRI time-courses, the pattern that appears first among a selection of patterns is used to index the time-course [2]. This scheme can yield a compact encoding of temporal relations between the selected patterns that really matter in distinguishing between healthy individuals and those diagnosed with the ADHD. The optimal pattern projections will be learned to result in the hash codes that minimize the diagnosis errors. In this way, FTA allows for not only high detection rates, but a scalable solution for detecting fMRI sequences. Since the projections are learned as opposed to randomly generated, the solution scales well with large-scale input fMRI sequences using compact hash codes.

Suitably, the objective of this chapter is to provide a scalable and efficient [69, 105] solution to the problem of detecting neurodevelopmental disorders (specifically ADHD) via fMRI time-courses. Doing so also reverberates to improved success in brain imaging technologies. The proposed temporal order-based hashing algorithms are much more generic, providing a new framework for fast matching and detection to other forms of time-series data. In this chapter, the method has implications on functional neural analysis[40, 55]. Being able to reliably infer causal relationships between brain structures and functions presents an interesting opportunity for further investigations, the range of which include areas of classification outside neurodevelopmental disorders [37]. Ultimately, learning the projections to hash a time-series space efficiently provides important practicality to the design.

The contributions of this chapter are:

[2]Without loss of generality, we can also designate the second or the third appearing pattern or so on to hash a fMRI sequence. As a convention, we choose the first-appearing pattern in this chapter.

1. We propose a novel FTA hashing algorithm to hash time series with varied length into fixed-size hash codes by encoding the temporal order of the latent patterns inside the time series.

2. In order to acquire the optimal projections, we formulate it as a learning problem whose training loss can be minimized in an efficient fashion.

3. We perform extensive experiment studies on benchmarks of ADHD detection and demonstrate the superior performance of the proposed FTA hashing with several evaluation metrics.

The remainder of this chapter is organized as follows: Section 2.1 briefly reviews the related work. The ADHD detection paradigm including FTA hashing algorithm is introduced and discussed in Section 2.2, with the learning algorithm for searching optimal projections. Experiments and performance studies are presented in Section 2.3. Finally, Section 2.4 summarizes the chapter.

2.1 ADHD Detection

In this section, we review several related topics pertinent to the task of ADHD detection. Among these, the overall theme fosters support for classification analysis of fMRI time-courses.

It is known that multivariate data mining and machine learning methods have been used to approach the classification of fMRI data[37, 24]. Similar to the approach in the chapter, multivariate methods presume from a core tenet of neuroscience that neural data encodes itself across larger functional regions in the brain. Intriguingly, the motivation behind utilizing multiple ROIs in test and training exists within other works as well; as ROI selection can be viewed as a form of feature selection [37, 97]. Similar works [24] also suggest the widespread adoption of multivariate techniques including Support Vector Machines (SVMs) [22] and Linear Discriminant Analysis (LDA) [11] in spite of the traditional uni-variate alternatives. Fueling this shift was a dissatisfaction with

how univariate models rely exclusively on the information contained in time-courses contained in individual voxels[24].

In tune with our method's focus on preserving temporal structure is the Dynamic Time Warping (DTW) to find similar patterns between two time series [10]. The idea is to create a sequence alignment algorithm that preserves and efficiently discovers knowledge from potentially large data archives. The approach used in this chapter (FTA) maintains similar motivations, insofar as it aims to solve the task of presenting a model for efficient and scalable knowledge discovery in the domain of neural data.

The related works presented offer several intriguing points of support to our investigation. Foremost, the premise has been set for the intuitions behind fMRI analysis for ADHD detection[24]. For example, [29] extracts features from fMRI time courses to improve the ADHD detection rate. Meanwhile, the method proposed by [36] combines both unsupervised and supervised algorithms and achieves best performance in ADHD-200 Global Competition. This reliably shows that the problem of knowledge discovery in brain imaging can be improved through utilization of Machine Learning models. In addition, other efforts suggest that a detection scheme for analysis of fMRI time-courses can be expanded to other neural datasets and disorders[29, 24]. This offers an expanded utility to the model presented in this chapter, in which the FTA can be used on other problems within Neuroscience and a host of other topic areas, all whilst preserving efficiency. Lastly, DTW offers an element of distinction between former methods in time-series analysis versus those of which focus specifically on preserving temporal order during encoding of time-series data [10]. In doing so, FTA provides clear advantages over methods which eschew these temporal features [42].

2.2 First-Take-All: ADHD Detection by Time-Series Hashing

In this section, we introduce a novel hashing-based paradigm to automatically identify ADHD subjects. The structure of our method can be summarized as following: First, brain atlases containing a number of Regions of Interest (ROIs) will be constructed and corresponding time courses of each ROI will be extracted based on the ADHD subjects' resting state fMRI data of the brain. Then, we propose a new temporal order-preserving hashing algorithm called First-Take-All to hash time courses into binary sequences. With those sequences, we compare the distance (similarity) between them to determine whether a patient is an ADHD subject.

2.2.1 Brain Atlas Construction and Time Course Extraction

In brain neuroanatomy, many approaches, such as automated anatomical labeling (AAL) [132], Eickhoff-Zilles (EZ) [32], Talairach and Tournoux (TT) [75] and Harvard-Oxford (HO) [41], have been proposed to construct brain atlases by using structural anatomic or functional information. After that, voxels in the regions that have structural or functional similarities will be grouped into Regions of Interest (ROIs) and the time courses of these ROIs can be extracted from the voxels of the subjects' resting state fMRI data. Since the brain atlas construction and time-course extraction are not the focus of this chapter, we will employ the existing brain atlases pre-constructed by the neuroanatomy community. The details will be presented in Section 2.3.

With time courses of ROIs extracted, every subject can be mapped to a unique vector (multivariate) sequence which describes the brain activities of that subject. For example, $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T]$ can be a subject's time-courses of ROIs, where each $\mathbf{x}_t \in \mathbb{R}^D, t = 1, \cdots, T$ represents the brain activities of that subject in D different ROIs.

2.2.2 *Temporal Order-Preseving Hashing*

Now we propose a hashing algorithm for time-series data which can map a TC into a fixed-size hash codes regardless of its original length. It can be roughly divided into 2 parts. First, a TC will be projected into several subspaces to produce a set of projection sequences. Then, we generate hash codes for entire TC by conducting an operation called First-Take-All (**FTA**) on those projection sequences.

2.2.2.1 *Sequence Projection*

Consider a TC $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_T]$ of length T described in Section 2.2.1. The first step is to project $\mathbf{X}$ into several subspaces defined by an optimized projection matrix $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_K] \in \mathbb{R}^{D\times K}$. Each $\mathbf{w}_k \in \mathbb{R}^D$ is a projection vector taken from $\mathbf{W}$ that generates a sequence $\mathbf{s}_k = \mathbf{w}_k^\intercal \mathbf{X}$ for $k \in \{1, \cdots, K\}$. The way to find the optimal $\mathbf{W}$ will be given in section 2.2.3.

Intuitively, each sequence $\mathbf{s}_k$ represents the score over the occurrence of a latent pattern[3] $\mathbf{w_k}$, and any TC is composed of a sequence of temporally-ordered patterns. The **orders** of certain **unknown** neural activation patterns often matter in ADHD detection. Thus, we seek to find these relevant patterns as well as compactly represent their orders in a hash code space, where the similarity between TCs can be directly computed by their Hamming distance.

To model the temporal order of patterns, first we need to locate the moment they appear. Here we use softmax to compute the probability that a pattern k appears at time t:

$$p_{k,t} = \frac{\exp(\mathbf{w}_k^\intercal \mathbf{x}_t)}{\sum_{t'=1}^{T} \exp(\mathbf{w}_k^\intercal \mathbf{x}_{t'})} \tag{2.1}$$

[3]These patterns are latent because they are unlabeled.

Let $\mathbf{u} = [1/T, 2/T, \cdots, t/T, \cdots, 1]^\intercal$ be the normalized timescale, each entry of which denotes a relative time moment on the range of $[0, 1]$ in an input sequence of length T. Then, the expected moment m_k that the pattern k appears can be calculated as

$$m_k = \mathbb{E}_{t \sim p_{k,t}} \left[\frac{t}{T} \right] = \sum_{t=1}^{T} \frac{t \cdot p_{k,t}}{T} = \mathbf{u}^\intercal \mathbf{p}_k \tag{2.2}$$

where $\mathbf{p}_k = [p_{k,1}, \cdots, p_{k,T}]^\intercal$ is a vector containing the probability of pattern k appearing at each moment.

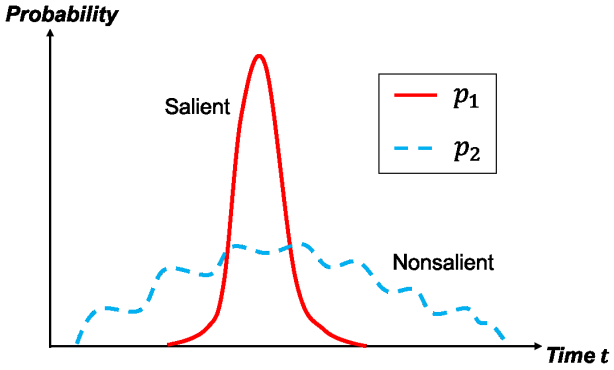


Figure 2.2: Comparison between projections generated by salient and nonsalient patterns. The red solid line represents the projection of a salient pattern with a small variance, while the blue dotted line represents the projection of a non-salient pattern with a large variance over the time axis.

Note that a pattern related with ADHD detection usually corresponds to a salient pattern of neural activation in brain ROIs. Thus, we expect that it should have a sharp appearance in the projection sequence such as the p_1 shown in figure 2.2. Accordingly, we propose to minimize the **variance of pattern occurrence**

$$v_k = \mathrm{Var}_{t \sim p_{k,t}} \left[\frac{t}{T} \right] = \sum_{t=1}^{T} \frac{(t - m_k)^2 \cdot p_{k,t}}{T^2} \tag{2.3}$$

together with the other criteria to learn the projection matrix $\mathbf{W}$ in Section 2.2.3. This regularization term is more likely to generate a salient pattern that really matters in detecting ADHD.

2.2.2.2 First-Take-All Temporal-Order Comparison

Now putting the expected appearing moments of K patterns into $\mathbf{m} = [m_1, \cdots, m_K]$, we wish to develop an ordinal hashing algorithm directly encoding their temporal order for a TC. Specifically, we perform a First-Take-All (FTA) comparison to rank the patterns by their temporal order – the pattern whose expected appearing moment comes first wins the FTA comparison, and its index is used to hash the entire TC.

For example, when we have two projected sequences (i.e., $K = 2$), FTA simply encodes the pairwise order between two corresponding patterns. When $K > 2$, FTA makes a higher-order comparison to decide which pattern appears first. Note that the output FTA hash code is not binary; instead it is a K-ary code. For this reason, we call K the FTA base.

For a pairwise FTA comparison involving only two patterns, knowing the first coming pattern completely encodes the temporal order between these two patterns. This can be generalized to high-order comparison if more than two patterns are involved for choosing the first-coming pattern. Such a high-order FTA comparison could generate more compact code to distinguish between different types of TCs. For example, suppose there are three types of TCs have different orders of patterns 1 – 2 – 3 – 4, 1 – 3 – 2 – 4 and 1 – 4 – 3 – 2, respectively. Then an effective FTA comparison only needs to make a order-3 comparison between the last three patterns 2, 3 and 4, which will output the FTA code 2, 3 and 4 to distinguish these three types of TCs. In this case, there is no need to make comprehensive comparisons between all possible pairs of patterns.

However, the patterns whose orders matter in classifying different types of TCs are unknown a-

priori. A suitable group of patterns must be learned so that the same type of TCs will have similar pattern orders. We will discuss the detail about the learning of these patterns in Section 2.2.3.

Mathematically, the index of the first-appearing pattern can be expressed as

$$\mathbf{h} = \arg\min_{\boldsymbol{\theta}} \boldsymbol{\theta}^{\intercal}\mathbf{m} = \boldsymbol{\theta}^{\intercal}[\mathbf{u}^{\intercal}\mathbf{p}_1|\cdots|\mathbf{u}^{\intercal}\mathbf{p}_K] = \mathbf{u}^{\intercal}\mathbf{P}\boldsymbol{\theta} \tag{2.4}$$

where $\boldsymbol{\theta} \in \{0,1\}^K, \mathbf{1}^{\intercal}\boldsymbol{\theta} = 1$ and $\mathbf{h}$ is an 1-of-K indicator of the FTA winner – its unique nonzero entry is indexed by the first-appearing pattern in the input TC, and $\mathbf{P} = [\mathbf{p}_1, \cdots, \mathbf{p}_K]$.

Algorithm 1 First-Take-All Hashing

Input: TC $\mathbf{X}$, code length L, a set of projection matrices $\{\mathbf{W}_i\}_{i=1}^{L}$
Initialize: $b \leftarrow$ empty sequence
for $i = 1$ to L **do**
    $\mathbf{S} = \mathbf{W}_i^{\intercal}\mathbf{X}$
    for each row $\mathbf{s}_k$ of $\mathbf{S}$, calculate m_k through Eq.(2.1) and Eq. (2.2)
    $k^* \leftarrow \arg\min_{1\le j\le K} m_k$.
    $b \leftarrow bk^*$ (concatenation)
end for
return b

An algorithmic description for the entire FTA hashing procedure is shown in Algorithm 1. Multiple FTA codes can be generated with a set of projection matrices. The code length L in the algorithm represents the number of hash codes generated for a TC.

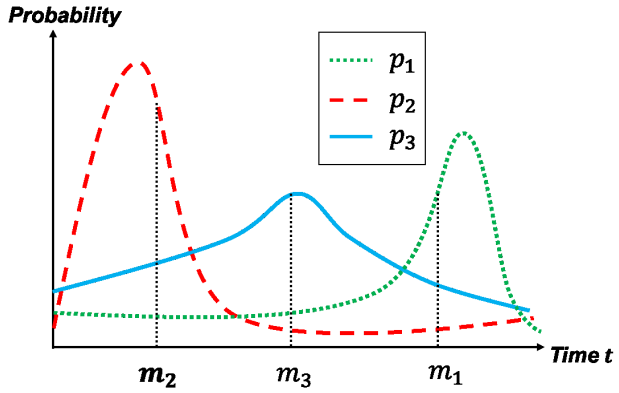


Figure 2.3: Illustration for First-Take-All Temporal-Order comparison when K is set to 3

Figure 2.3 illustrates the FTA comparison when K is set to 3. Here p_1, p_2 and p_3 in Figure 2.3 are the probability of each pattern appearing over the time axis t. From them, we can get their expected pattern appearing moments m_1, m_2 and m_3, which are approximately shown in the figure, where we have $m_2 < m_3 < m_1$. By the FTA comparison, we choose the index of m_2 as the hash code for the TC, which is 2.

Before the end of this section, let us analyze the computational complexity of hashing a sequence by FTA. First, it costs $O(TDK)$ to apply the projection matrix to an input TC $\mathbf{X}$. Then finding the expected moments of K projected sequences costs $O(TK)$. It also costs $O(K)$ to find the first-appearing pattern which wins FTA comparison out of K candidates. Hence, FTA totally costs $O(TDK)$ to hash an input TC up to a constant factor.

2.2.3 Learning Optimal Projections

A learning algorithm to find out the optimal projections $\mathbf{W}$ will be presented in this section, including the formulation of the optimizing problem and its efficient solution.

2.2.3.1 *Training Loss*

Given a TC $\mathbf{X}$ and the expected appearing moments of K patterns $\mathbf{m} = [m_1, \cdots, m_K]$, which is acquired from a fixed $\mathbf{W}$. Then we can apply the following softmin to calculate the probability that the kth pattern will appear first:

$$h_k \triangleq P(\text{pattern } k \text{ comes first}|\mathbf{X}) = \frac{\exp(-m_k)}{\sum_{k'=1}^{K} \exp(-m_{k'})} \tag{2.5}$$

The smaller the m_k, the more likely the pattern k will appear first.

Now consider a pair of TCs $\mathbf{X}^{(i)}$ and $\mathbf{X}^{(j)}$, along with their label s_i and s_j. We hope that through our learning algorithm, the resultant FTA hash codes can reflect the label similarity between two TCs. In other word, when $s_i = s_j$, there is a greater chance that the same pattern will appear first in both $\mathbf{X}^{(i)}$ and $\mathbf{X}^{(j)}$; otherwise, different patterns will appear first if $s_i \neq s_j$ if $s_i \neq s_j$.

Mathematically, it is easy to see that $h_k^{(i)} h_k^{(j)}$ is the probability that the kth pattern will appear first in both $\mathbf{X}^{(i)}$ and $\mathbf{X}^{(j)}$. Suppose h^{ij} represents the probability that the same pattern will appear first in both TCs. It can be computed as by summing up over all patterns

$$h^{ij} = \sum_{k=1}^{K} h_k^{(i)} h_k^{(j)} \tag{2.6}$$

Our goal is to maximize h^{ij} when $s_i = s_j$ and to minimize it when $s_i \neq s_j$. This results in the following objective function

$$O^{ij} = (1 - h^{ij})^{s_{ij}} (h^{ij})^{(1-s_{ij})} \tag{2.7}$$

Here, $s_{ij} = 1$ *i.f.f.* $s_i = s_j$; and $s_{ij} = 0$ otherwise. We wish to minimize it for all TC pairs. Suppose the training set is $\mathcal{T} = \{\mathbf{X}^{(i)}, s_i\}_{i=1}^{N}$ with N TCs. Then the total logarithmic training loss

over $\mathcal{T}$ becomes

$$\mathcal{L} = \sum_{i=1}^{N} \sum_{j=1}^{N} \left[s_{ij} \log \left(1 - h^{ij}\right) + (1 - s_{ij}) \log \left(h^{ij}\right) \right] \tag{2.8}$$

Minimizing it can minimize the pairwise diagnosis errors on the training set incurred by the resultant FTA hash codes.

2.2.3.2 Projection Orthogonality

In addition to the minimization of training loss, it is worth mentioning that the redundancy between the learned patterns also affects the FTA hashing performance. With a set of redundant patterns learned, their projection sequences would be highly correlated or even identical to one another. This could reduce the degree of the temporal order being distinguished between different patterns. In this case, a smaller perturbation or local warping would change the temporal orders significantly, thereby degenerating the FTA's performance in presence of noises.

To improve the resiliency of FTA against perturbations or noises on TCs, we wish to reduce the redundancy between patterns by minimizing the following normalized inner products between projection vectors

$$\Omega = \sum_{k \neq k'=1}^{K} \left(\frac{\mathbf{w}_k^{\mathsf{T}} \mathbf{w}_{k'}}{\|\mathbf{w}_k\| \|\mathbf{w}_{k'}\|} \right)^2 \tag{2.9}$$

Clearly, minimizing it can make the learned projection vectors as orthogonal to each other as possible, thereby minimizing the redundancy between the corresponding patterns [4].

[4]The projection orthogonality can also be imposed as a hard constraint in the optimization problem. However, by experiments, we found that the performance is more stable by posing it as a soft term that penalizes the projection redundancy in the objective function.

2.2.3.3 *Putting Together*

In addition to the training loss (2.8) and the projection orthogonality (2.9), we also consider to minimize the variance of pattern occurrences as shown in Eq. (2.3). This can be expressed as the following total variance over the training set:

$$\mathcal{V} = \sum_{i,k=1}^{N,K} v_k^{(i)}$$

where $v_k^{(i)}$ is the occurrence variance of pattern k in the ith TC of training set. As aforementioned, minimizing the variance of pattern occurrences can generate salient patterns for ADHD diagnosis.

Then putting them together, we can define the following minimization problem to learn the projection matrix $\mathbf{W}$

$$\begin{aligned} \min_{\mathbf{W}} \mathcal{F} \triangleq \mathcal{L} \quad & \cdots \text{ training loss} \\ + \gamma\, \Omega \quad & \cdots \text{ Projection Orthogonality} \\ + \eta\, \mathcal{V} \quad & \cdots \text{ Variance of pattern occurences} \end{aligned} \tag{2.10}$$

where two positive coefficients γ and η are two hyper parameters that control the contributions of the projection orthogonality and the minimization of pattern occurrence variance.

2.2.3.4 *Optimization*

We adopt the stochastic gradient descent method to minimize $\mathcal{F}$ as to find the optimal projection $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_K] \in \mathbb{R}^{D\times K}$. For each training iteration, we randomly pick up a pair of TCs and their labels and calculate the gradient $\nabla_{\mathbf{W}}\mathcal{F}$. Since $\mathcal{F}$ is differentiable w.r.t. $\mathbf{W}$, it allow us to calculate $\nabla_{\mathbf{W}}\mathcal{F}$, leading to an efficient learning procedure. All equations needed to calculate $\nabla_{\mathbf{W}}\mathcal{F}$ can be found in Appendix and the entire optimization procedure is described in Algorithm

2.

Algorithm 2 Learning Optimal Projections

1: **Input:** training TC set $\chi = \{\mathbf{X}^{(i)}, s_i\}_{i=1}^{N}$, K, learning rate α

2: **Initialize:** Randomly initialize $\mathbf{W} = [\mathbf{w}_1, \cdots, \mathbf{w}_K]$, $\mathbf{w}_k \in \mathbb{R}^{D\times 1}, k = 1, \cdots, K$

3: **repeat**

4: Select training pair $\mathbf{X}^{(i)}$, $\mathbf{X}^{(j)}$.

5: $P^{(i)} = [p_{k,t}^{(i)}]_{K\times T}$, $P^{(j)} = [p_{k,t}^{(j)}]_{K\times T}$, calculate $p_{k,t}^{(i)}$ and $p_{k,t}^{(j)}$ based on Eq.(2.1)

6: for each $\mathbf{w}_k, k = 1, \cdots, K$, compute $\frac{\partial \mathcal{F}}{\partial \mathbf{w}_k}$ with $p_{k,t}^{(i)}$ and $p_{k,t}^{(j)}$ based on Eq.(.2), (.3), (.4), (.5), (.6), (.7), (.8), (.9).

7: $\nabla_{\mathbf{W}}\mathcal{F} \leftarrow [\frac{\partial \mathcal{F}}{\partial \mathbf{w}_1}, \cdots, \frac{\partial \mathcal{F}}{\partial \mathbf{w}_K}]$

8: $\mathbf{W} \leftarrow \mathbf{W} - \alpha\nabla_{\mathbf{W}}\mathcal{F}$

9: **until** Convergence.

The above paragraph depicts the training algorithm for a projection matrix resulting in one FTA hash code. The above learning algorithm can be used as a subroutine in a standard ensemble method like AdaBoost. This will yield multiple FTA codes for a TC. The similarity between TCs can be computed with the Hamming distance between their concatenated FTA codes. Then, ADHD can be fast detected by retrieving the similar TCs from a labeled database.

2.3 Results for ADHD Detection with FTA Hashing

In this section, we demonstrate the effectiveness of the proposed method by conducting experiments on ADHD 200 dataset, a dataset developed for ADHD detection. First we give a brief introduction on ADHD dataset. Then we discuss the experiment setting. We compare the proposed method with several supervised and unsupervised baselines with different evaluation metrics. Fi-

nally, we study the impact of the hyper-parameters K and L on the performance.

2.3.1 Datasets and Background

We evaluate the proposed FTA approach on ADHD-200 dataset. ADHD-200 was initially prepared by ADHD-200 Consortium[89] for the ADHD-200 Global Competition, a competition that aimed to improve the understanding of the neural basis of ADHD through the implementation of the scientific discovery. It contains 776 records of the resting-state fMRI and anatomical data across 8 independent imaging sites, 491 of which come from typically developing individuals and 285 from children and adolescents diagnosed with ADHD (ages: 7-21 years old). Accompanying phenotypic information includes: diagnostic status, dimensional ADHD symptom measures, age, sex, intelligence quotient (IQ) and lifetime medication status. Preliminary quality control assessments (usable vs. questionable) based upon visual time-series inspection are included for all resting state fMRI scans. An additional 197 individuals from six imaging sites were released without the diagnosis labels during the competition for testing purposes and their labels were released separately afterwards. More information on the dataset can be found at http://fcon_1000.projects.nitrc.org/indi/adhd200/.

In order to bring the ADHD-200 Global Competition to a wider audience, The Neuro Bureau[5] made preprocessed versions of the competition data freely available to the general public to help those whose specialities lay outside of resting-state fMRI analysis to bypass technical obstacles. There are several preprocessed datasets available which were preprocessed by different pipelines. In order to fairly compare the proposed method with the baselines, we choose the dataset preprocessed by Athena pipeline[6] which is also used by the baseline methods. More information about the

[5]http://www.neurobureau.org/

Athena pipeline can be found at Neuro Bureau's website[6].

2.3.2 Experimental Setting and Baselines

For the sake of fair comparison, we follow the experiment setting similar with the baselines. For the proposed FTA hashing, we determine the values of hyper parameters K, L, γ and η by conducting 5-fold cross validation on the training set. As mentioned in section 2.2.1, the TCs we used for evaluation were extracted from the pre-constructed brain atlas which was built with automated anatomical labeling (AAL)[132]. The way to extract TCs is averaging the time courses within each ROI voxel[6]. Note that the AAL atlas was constructed using anatomic and cyto-architectonic information and did not incorporate functional information. Thus the resultant TCs do not contain any prior phenotypic information which may impact the evaluation. Based on the cross validation, FTA base K and code length L are set to 2 and 200 respectively.

We compare the proposed method with following algorithms:

- Dynamic Time Warping (DTW)[103]: A well known time series alignment technique that computes optimal distance between two time series of different lengths while preserving their temporal order.
- Derivative Dynamic Time Warping (DDTW)[70]: This method uses derivatives of the original time series to improve alignment by DTW.
- Canonical Time Warping (CTW)[145]: This method combines canonical correlation analysis (CCA) with DTW.

[6]http://www.nitrc.org/plugins/mwiki/index.php/neurobureau:AthenaPipeline

- Method from Johns Hopkins University (JHU)[36]: The winner of ADHD-200 Global Competition which achieved the state-of-the-art performance.
- Attributed Graph Distance Measure (AGDM)[29]: This method proposed a graph based feature called Attributed Graph Distance Measure which can be used to classify ADHD subjects.

Note that DTW, DDTW and CTW are unsupervised methods while JHU and AGDM are supervised. Following the settings of these baselines, we evaluate the proposed method with four statistical metrics: Prediction Accuracy, Specificity (= True Negative Rate), Sensitivity (= True Positive Rate) and J-Statistic (= Specificity + Sensitivity - 1). Among them the Prediction Accuracy is the primary metric for scoring. Note that the detection rate used in [29] is identical to prediction accuracy.

2.3.3 Comparison with Unsupervised Baselines

We begin our evaluation by demonstrating comparison results with the unsupervised baselines. As mentioned in Section 2.3.1, the training set consists of 8 subsets collected from different sites while the test set consists of 6 subsets. We implement FTA in Matlab and use the Matlab implementation of DTW, DDTW and CTW provided by [144] to conduct the experiments. The training and testing are performed on the training and testing subsets across all the eight imaging sites. The hardware configuration for the experiment is Intel i7-4790 CPU at 3.6GHz and 8GB RAM. Table 2.1 shows all four evaluation metrics and the computing time of three unsupervised baselines and the FTA.

As shown, the FTA outperforms all three baselines by nearly 15% to 18% on the prediction accuracy, a significant improvement to these well known time-series alignment methods. This demonstrates the FTA can better encode the temporal patterns which are important for predicting ADHD

subjects than the unsupervised baselines. All four methods have a high specificity but a relatively low sensitivity. The FTA can reach the best specificity of 0.9149 but with a sensitivity of 0.2727. This is reasonable since there are much more TDC subjects than ADHD children diagnosed in the training set (491 TDC and 285 ADHD children) thus the ability of a classifier to detect ADHD children is dampened – the prior distribution of TDC is biased by a large number of TDC samples in a general population. Such an unbalance between TDC and ADHD subjects can be relieved by imposing a larger penalty on missing ADHD subjects. However, we do not perform such a re-balance for the sake of a fair comparison with the other baselines.

Table 2.1: Performance comparison between the unsupervised baselines and the FTA.

Metric	DTW	DDTW	CTW	FTA
Accuracy	0.4678	0.4386	0.4678	**0.6140**
Specificity	0.7127	0.6915	0.8085	**0.9149**
Sensitivity	**0.2987**	**0.2987**	0.1818	0.2727
J-Statistics	0.0115	-0.0098	-0.0097	**0.1876**
Time(s)	149.5502	267.8659	47996.5866	**20.771**

Next we perform efficiency evaluation for all four methods. Since all three baselines adopt Dynamic Programming (DP) to perform similarity search, it can be expected that they will have a much slower speed than the proposed FTA hashing which performs similarity search by the Hamming distance. Figure 2.4 shows the Prediction Accuracy versus the Execution Time for all the methods. Compared with the baselines, FTA achieves more than 30% higher prediction accuracy on the entire test set and at least 7 times faster than DP based baselines. That suggests the proposed FTA hashing can be used for fast detection of ADHD subjects.

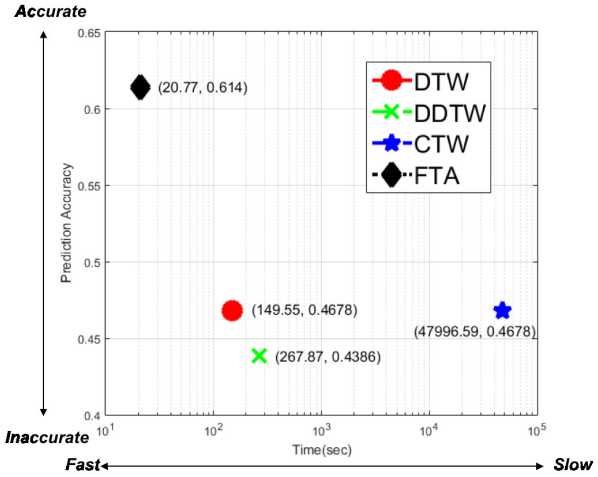


Figure 2.4: Prediction Accuracy versus Execution Time. Comparison between FTA and unsupervised baselines

2.3.4 Comparison with Supervised Baselines

Now we compare the proposed method with two supervised baselines: JHU[36] and AGDM[29]. The approach proposed by JHU is a weighted combination of several algorithms including CUR decompositions, random forest, gradient boosting and support vector machine et al. It adopts both fMRI data and the accompanying phenotypic information like IQ to predict the ADHD subjects. On the contrary, another baseline AGDM only requires fMRI data to make the prediction. According to [29], features called AGDM were extracted from fMRI data to encode the brain network structure first. Then they were used to train a SVM classifier which made the final prediction. Since two baseline methods have different experiment settings, we follow their individual settings to make a fair comparison.

2.3.4.1 FTA vs JHU

Similar to Section 2.3.3, we train and test the FTA on all the training subsets across the eight sites. More specifically, we randomly sample $3,000$ pairs from all the 8 training subsets and use them to find the optimal projections. We also report the evaluation metrics following the definitions from the ADHD-200 Global Competition [89]. The comparison results between the JHU and the FTA are summarized in Table 2.2.

Table 2.2: Performance evaluation of JHU and FTA

Metric	JHU	FTA
Prediction Accuracy	0.6102	**0.6140**
Specificity	**0.94**	0.9149
Sensitivity	0.21	**0.2727**
J-Statistics	0.15	**0.1876**

From the table, we can see the proposed FTA outperforms the JHU on three metrics – prediction accuracy, sensitivity and J-Statistics. This is an impressive result, considering FTA only uses fMRI time courses to achieve such performance, whereas the JHU method involves both fMRI time courses and phenotypic information. Especially, compared with the JHU result, FTA improves the sensitivity by 29.85% while still keeping a competitive specificity (over 0.9). Such results demonstrate that encoding the temporal orders of the different neural activity patterns is definitely a helpful clue to identify the ADHD subjects.

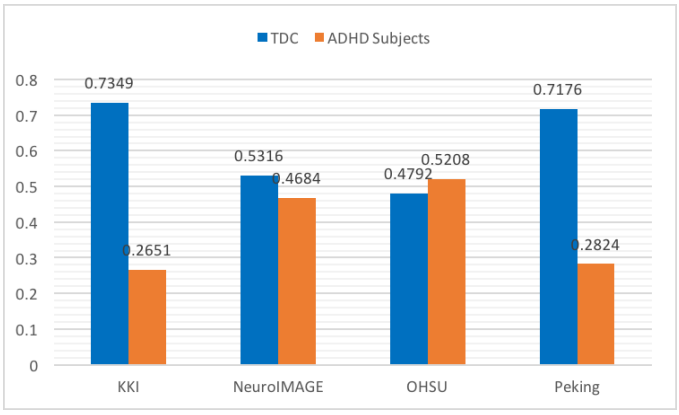
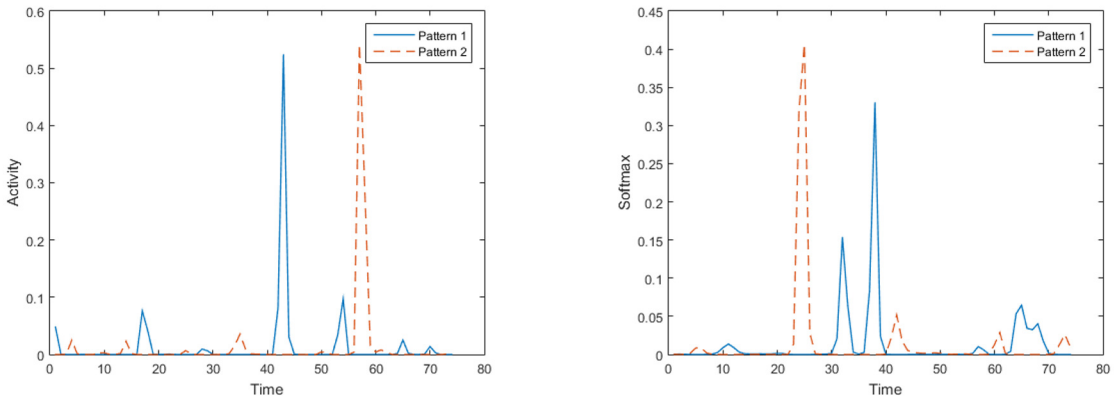


Figure 2.5: Percentage of TDC/ADHD Subjects in each training subset.



a. Typically Developing Children

b. ADHD Patient

Figure 2.6: Comparison of the temporal order of two patterns between the TDC and ADHD subjects. It illustrates how the order of these two patterns matters in detecting ADHD.

We further justify this claim by showing some real examples of temporal orders of learned patterns. Figure 2.6 shows the sequences of two patterns projected from the TCs corresponding to the TDC and ADHD subjects. Figure 2.6a comes from a TDC subject while Figure 2.6b is obtained from a ADHD subject. It is clear that these two patterns exhibit different temporal orders between the

TDC and ADHD subjects. Since the FTA can encode the temporal order of patterns, the resultant hash codes can characterize the neural activity difference between different types of TCs.

2.3.4.2 FTA vs AGDM

Unlike the JHU method, the AGDM conducted their experiments on the individual subsets. There are totally eight subsets collected by different imaging sites and AGDM chose four of them to evaluate their approach. The chosen subsets are collected by 1) Kennedy Krieger Institute (KKI), 2) NeuroIMAGE, 3) Oregon Health & Science University (OHSU) and 4) Peking University (Peking) respectively. To evaluate the proposed method, we follow the same experiment setting as the AGDM, and train and test on each subset separately.

Table 2.3: Performance evaluation on individual sets

Sites	Prediction Accuracy		Specificity		Sensitivity		J-Statistic	
	AGDM	FTA	AGDM	FTA	AGDM	FTA	AGDM	FTA
KKI	0.5455	**0.8182**	0.625	**1**	**0.3333**	**0.3333**	-0.0417	**0.3333**
NeuroImage	0.48	**0.8**	0.6429	**0.8571**	0.2727	**0.7273**	-0.0844	**0.5844**
OHSU	0.8235	**0.8676**	0.8929	**0.9643**	0.5	**0.6667**	0.3929	**0.6310**
Peking	0.5882	**0.6176**	0.9259	**1**	0.2083	**0.25**	0.1342	**0.25**
Average	0.6281	**0.7759**	0.8312	**0.9554**	0.2727	**0.4943**	0.1003	**0.4497**

Table 2.3 shows the comparison results between the FTA and the AGDM on individual sets. Apparently, the FTA significantly outperforms the AGDM on every evaluation metrics. Specifically, the average prediction accuracy and the average sensitivity of FTA outperform the AGDM by around 15% and 20% respectively. This means more than 75% of subjects can be correctly classified, and nearly half of ADHD patients can be successfully detected by the FTA. Moreover, the specificities of FTA on all four subsets are obviously boosted compared with the AGDM. Especially on KKI and Peking, the specificity achieves 1 – all TDC subjects are correctly identified.

Similar results can be found about the sensitivity. On the NeuroIMAGE and the OHSU subsets, the sensitivities have reached over 0.7 and 0.6, much closer to the corresponding specificity. Consider our discussion in Section 2.3.3 – the low sensitivity is caused by high ratio of TDC in the training set. We calculated the percentage of TDC and ADHD subjects on each subset, which are illustrated in Figure 2.5. It shows that the ratio of TDC/ADHD subjects on NeuroIMAGE and OHSU is close to 1, while it is much higher on KKI and Peking. This implies that the low sensitivity is caused by a high ratio of TDC/ADHD.

2.3.5 Parameter Sensitivity Analysis

Finally, we evaluate the FTA's performance by varying its hyper-parameters. In particular, we study the impact of the code length L and FTA base K on the performance. Besides the AAL, we also conduct experiments with TCs extracted from 3 other pre-built brain atlas: Talairach and Tournoux (TT)[75], Harvard-Oxford (HO)[41] and CC200[25].

2.3.5.1 Prediction Accuracy vs K

By varying the FTA base K, it can be shown that it affects the test accuracy of the FTA model. For a fixed code length of 200, a variety of K values were used: from 2 through 7. The results show differences in impact of K for different ROI atlases. For instance, with the CC200, an increased K tends to improve the accuracy, whereas an increased K in the AAL results in a subtle decrease in the test accuracy followed by an increase after $K = 6$. Overall, all four ROI atlases, begin to converge to a median of performance when K approaches 7. Throughout the tests, the results indicate only small dependencies contingent on the values of K.

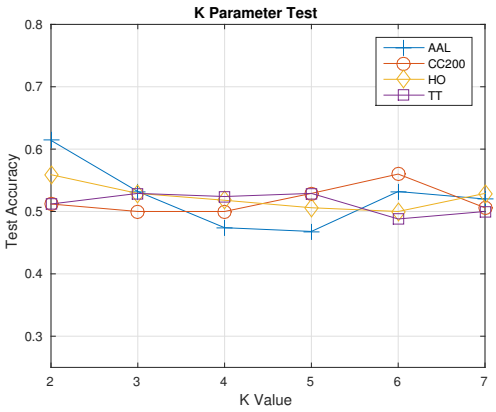


Figure 2.7: Test accuracy across different ROIs, using different K values

Considering K is the number of patterns involved in generating a hash code, assessing its role offers a way to tune the amount of discriminant information available at any given comparison. From Figure 2.7, it seems that for slight increases in K (e.g., from K = 4 to K = 6 for CC200) the amount of information increases, offering an increased test accuracy. However, there appears to be a point at which further increasing the value of K no longer improves the test accuracy (beyond $K = 6$). Ultimately, this leads to a decline in performance due to the redundancies in the comparison of a too large number of patterns.

2.3.5.2 Prediction Accuracy vs L

Now let us assess the code length parameter L. Fixed at $K = 2$, a variety of code lengths were tested, ranging from $L = 10$ to $L = 1000$. In Figure 2.8, it can be seen that L acts differently across different ROIs, especially between the range of 10 to 200 code length. Incidentally, it is shown that this range experiences the most abrupt shifts in performance, whilst the range 200-1000 shows either constant or steady trends. In the case of AAL and HO, they are particularly sensitive to

this parameter. Within the $L = 10$ to $L = 200$ range, the performance reaches its peak and then drops subsequently. Extending to the set of other ROIs, Figure 2.8 shows that this trend (to a lesser degree) is exhibited on CC200 and TT as well.

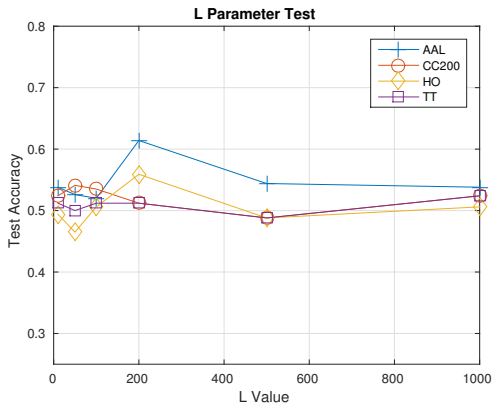


Figure 2.8: Test accuracy across different ROIs, using different L values

Resulting from this experiment, it can be shown that varying code lengths has impact on performance, with tuning required particularly in the range of compact codes from $L = 10$ and $L = 200$. Yet similar to K, simply increasing the value indefinitely does not result in a better test accuracy because of higher redundancies that might exist in longer codes.

2.4 FTA Summary

In this chapter, we propose a novel FTA algorithm to hash time series of varied length into fixed-size codes. We use the resultant hash codes to efficiently detect the ADHD subjects by fast retrieving the similar subjects. To maximize its performance, we design an effective algorithm to learn the optimal projections $\mathbf{W}$. The results of extensive experimental evaluations show the proposed FTA outperforms both unsupervised and supervised methods on the ADHD-200 dataset for identifying

ADHD subjects, beating the winning algorithm in the ADHD-200 Global Competition.

CHAPTER 3: IMPROVING VIDEO REPRESENTATION LEARNING THROUGH DYNAMIC ORDER ENCODING

In this chapter, we introduce a RNN-based recurrent model to encode even more complicated order patterns for video representation learning, which is a very active research area due to its fundamental role in many computer vision tasks. It attempts to close the huge performance gap between the state-of-the-art recognition systems and human beings. Inspired by the tremendous success of the Convolutional Neural Networks (CNN) in learning image representations [74, 116, 123, 63, 56], recent works focus on generalization the CNN architectures to learn high quality video representations [68, 126, 130, 117, 141]. Although these approaches make a significant progress in learning better video features, the performance on many tasks has a large gap to what has been achieved on the image-related tasks such as image classification [74, 116, 123, 56], human face recognition [112, 102] and human pose estimation [18, 139].

One of the possible reasons making video representation learning so challenging is video clips usually contain very rich global and local temporal information which is essential for distinguishing different videos with similar frame level information. For example, videos of opening/closing a door can be easily classified by the temporal information on when the door leaves/approaches the wall, while it is hard to classify them solely based on the visual information of individual frames. Most CNN-based video features handle the challenge by leveraging the local temporal information between consecutive frames [130, 68, 117, 38]. Some other methods employ Long-Short-Term-Memory (LSTM) [57] to capture the long-term dependencies between frames [31, 119]. However, the LSTM model encodes its output as a nonlinear function of frame-level image features, which still limits its capability to model how the temporal orders of action patterns would impact the recognition tasks.

This inspires us to encode global temporal information into video representations. Specifically, we present the Temporal Preserving Recurrent Neural Network (TPRNN) by generalizing the idea behind the First-Take-All framework [60], a novel representation that learns discrete-valued representations of sequential data by encoding the temporal orders of latent patterns. The proposed TPRNN architecture is designated to extract rich patterns over an entire video sequence and encode them as compact video features in ordered temporal structures. Different from the LSTM that is designed to memorize the long-term dependencies between frames, the proposed TPRNN generates video features directly from the ordinal relationships between action patterns in the temporal domain. Compared to frame-level features, the TPRNN features can be more discriminative in recognizing videos captured in the same context (e.g., background, objects and actors) but comprising different sequences of ordered action patterns.

To evaluate the proposed TPRNN model, we conduct extensive experiments on two action recognition datasets, UCF-101 [118] and Charades [115]. To verify the effectiveness of the model in encoding temporal orders of actions, we construct new action classes by reversing the original videos and test if the model can distinguish the reverse action from its original counterpart. Our experiment results show that the proposed TPRNN model outperforms the LSTM model on both datasets. Moreover, the TPRNN features also significantly improve the performance of the frame-level CNN features and LSTM on recognizing action classes that are only distinguishable by different temporal orders of action patterns.

The rest of this chapter is organized as follows. Section 3.1 reviews related literature on video representation learning. Then we introduce the proposed TPRNN architecture in Section 3.2. We show our experiment results on video action datasets in Section 3.3, and summarize the entire chapter in Section 3.4.

3.1 Related Works for Video Representation Learning

Deep features acquired by CNN-based architectures achieved great success in many computer vision applications such as image classification [74, 116, 123, 56], face recognition [112, 102], pose estimation [18, 139], etc. Due to its superior performance compared to conventional methods, recent works tend to expand the application of CNN features to a wider range of areas. In terms of video representation learning, several recent works have investigated the question of how to leverage temporal information in addition to the frame level spatial information. These efforts can be roughly divided into two different categories. One is to extend the convolution operation to the temporal axis with 3 dimensional filters learn spatiotemporal features. For example, [65] tries to stack consecutive video frames and extend 2D CNN into time axis. While [68] studies several approaches for temporal sampling shows they cannot encode the temporal information effectively as they only report a marginal improvement over a spatial CNN. Moreover, C3D method [130] introduces a architecture similar to [116] but allowing all filters to operate over spatial and temporal domain. [120] introduces another way to learn spatiotemporal features by factorizing 3D convolution into a 2D spatial and a 1D temporal convolution.

Another way to encode temporal information is represented by the two stream approach originally proposed by [117]. The method decomposes a video clip into spatial and temporal components and feeds them into two separated CNN architectures to learn spatial and temporal representations separately. Then the final prediction is based on the late fusion of the softmax scores from both streams. Rather than performing late fusion at softmax outputs, [38] studies several different fusion strategies including both spatial fusion (sum, concatenation, bilinear, etc.) temporal fusion (3D pooling). The fused features produced by the two stream approach is shown very effective in action recognition and has been deployed into several action recognition methods [17, 44].

Besides CNN-based features, some other methods employ LSTM [57] to encode the long-term

dependencies between frames into the video features. A typical way to do this is Long-Term Recurrent Convolutional Network (LRCN) [31], which is the most closely related work to ours. LRCN combines CNN structure and LSTM into a unified framework and can be trained in an end-to-end fashion. Similar works include [141] and [137]: [141] investigates ways to pool temporal features across longer time span, as well as sequential modeling with deeply stacked LSTM, while [137] fuses different types of features including two stream, stacked LSTM, spatial pooling and temporal pooling to make final prediction. Moreover, [119] treats LSTM layer as an autoencoder and learns video representations in an unsupervised fashion. Although LSTM can discover long-range temporal relationships between frames, the resultant video features are still from spatial feature space. On the contrary, First-Take-All hashing [60] encodes temporal order information directly from time space and can achieve good performance on time sensitive datasets.

3.2 Temporal Preserving Recurrent Network

In this section, we introduce Temporal Preserving Recurrent Network, a novel RNN-based model which is designed to encode temporal structure information between video frames. First, we explain the design principles for the proposed network along with the connection between the aforementioned First-Take-All Hashing [60], then we present the architecture of the proposed network including the definition of each layer and its functionality. Finally, we compare the proposed network with other recurrent networks, which share similar structures.

3.2.1 Intuition

We design the proposed Temporal Preserving Recurrent Network based on the inspiration from the First-Take-All (FTA) hashing [60], which employs the temporal order information to compactly

represent videos. In FTA, a multi-dimensional process is projected to a latent subspace at each time step, generating a set of 1D latent temporal signals. The occurrences of the maximal values are compared among all the latent signals, and the one with the first maximum occurrence is used to index the hash code. However, there are several drawbacks with the FTA formulation. (1) Only the index of the first-appearing patterns is used to represent the sequence. Others are ignored which may also contain useful information. (2) In the FTA comparison, The only learnable parameter is the linear projection. The number of projections and the dimension of each projection are also determined heuristically. Therefore, the learning capability and scalability of FTA is limited. (3) Since FTA is an hashing algorithm, the binary nature of the FTA codes prevents them from representing input sequences more accurate than those floating-point features.

To address the weaknesses of FTA hashing, we reformulate the FTA comparison in a recursive fashion, so that it can be implemented with a computational model such as recurrent neural network. Denote the original multi-dimensional process as $\{\mathbf{x}_t\}$, and the linear projection as $\mathbf{Z}$, the running maximum $\mathbf{m}_t$ of projected latent signals can be obtained by

$$\mathbf{m}_t = \max(\mathbf{Z}\mathbf{x}_t, \mathbf{m}_{t-1}). \tag{3.1}$$

All scalar functions such as $\max(\cdot)$ are applied in an element-wise way unless otherwise stated. The time steps $\mathbf{s}_t$ when the maximal values $\mathbf{m}_t$ first occurred so far can be recorded as

$$\mathbf{s}_t = \begin{cases} \tilde{t}, & \text{if } \mathbf{Z}\mathbf{x}_t > \mathbf{m}_{t-1}, \\ \mathbf{s}_{t-1}, & \text{otherwise.} \end{cases} \tag{3.2}$$

where $\tilde{t}$ stands for the normalized version of time index in $[0, 1]$. After recursive updating with equations 3.1 and 3.2, at the final step T, FTA calculates a binary hash code based on the index $\arg\min_i \mathbf{s}_T(i)$. Note that the conditional expression of $\mathbf{s}_t$ can be controlled by a logic gate $\sigma(\mathbf{Z}\mathbf{x}_t -$

$\mathbf{m}_{t-1}$), where $\sigma(\cdot)$ is a sigmoid function approximating the hard boolean operation.

3.2.2 Model Architecture

Based on the components of FTA, we propose a Temporal Preserving Recurrent Neural Network (TPRNN) with stronger temporal modeling capability. TPRNN is a RNN-based network with evolving memory cells connected in the same way at each time step. An illustration of the network structure at one time slice is shown in Figure 3.1. An input $\mathbf{x}_t$ (video frame feature) is fed into the network together with the current time step t, and the hidden states $\mathbf{m}_t$, $\mathbf{s}_t$ are updated with the control of gate $\mathbf{g}_t$. There are several components in the network with distinct functionality as described in below.

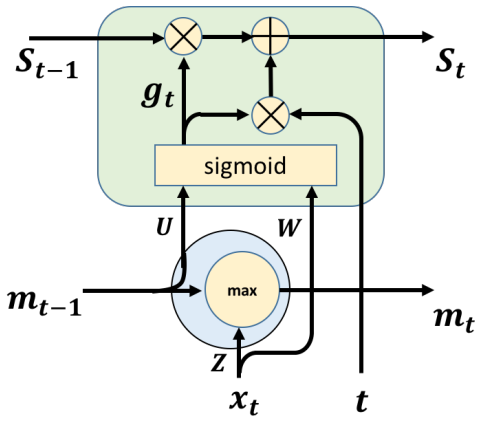


Figure 3.1: Structure of temporal preserving network, where the circle represents the max-pooling layer and the rounded corner rectangle represents the temporal preserving layer.

Max-pooling Layer The bottom circle in Figure 3.1 is a max-pooling layer over adjacent time steps. Same as equation 3.1 for FTA, this layer evaluates the latent patterns of input feature $\mathbf{x}_t$ with linear projection $\mathbf{Z}$, and stores the current maximal projected values in state $\mathbf{m}_t$ by comparing with

the previous state $\mathbf{m}_{t-1}$.

Temporal-preserving Layer This layer in the rounded corner rectangle of Figure 3.1 plays a key role in constructing the temporal preserving representation. It keeps track of important time moments t in state $\mathbf{s}_t$ with a control gate $\mathbf{g}_t$:

$$\mathbf{s}_t = (1 - \mathbf{g}_t) \cdot \mathbf{s}_{t-1} + \mathbf{g}_t \cdot \tilde{t}, \tag{3.3}$$

The gate $\mathbf{g}_t$ is a sigmoid function which is activated when the most salient moment of a latent event is detected:

$$\mathbf{g}_t = \sigma(\mathbf{W}\mathbf{x}_t - \mathbf{U}\mathbf{m}_{t-1}), \tag{3.4}$$

where the current input feature $\mathbf{x}_t$ is compared with the previous maximal response $\mathbf{m}_{t-1}$ in latent spaces spanned by matrices $\mathbf{W}$ and $\mathbf{U}$.

The temporal-preserving layer introduces a few extensions based on the FTA comparison in equation 3.2. First, rather than following a hard activation condition, we employ a soft activation function to monitor latent patterns, which is differential and more sensitive to subtle temporal changes. Similar to the gates in LSTM, $\mathbf{g}_t$ measures the likelihood of a latent pattern occurrence. It controls how much to forget about the previous state $\mathbf{s}_{t-1}$ and how much to remember for the current time step t. Such soft updating scheme enables multiple time steps to contribute as the occurrences of salient events. As a result, the temporal-preserving layer is capable of leveraging more high-order temporal information than solely considering the maximal response moments.

More importantly, two additional parameters $\mathbf{W}$ and $\mathbf{U}$ are introduced in equation 3.4 for better modeling of complex dynamic visual events. The projection $\mathbf{U}$ is applied on the max-pooling state $\mathbf{m}_{t-1}$ to extract more semantic information from input feature. The projection $\mathbf{W}$ together with

the original parameter $\mathbf{Z}$ provide two different latent spaces of $\mathbf{x}_t$ for activating gate and detecting maximal response, sharing similar motivation as the key/value addressing mechanism in Neural Turing Machine [54]. If we set $\mathbf{U} = \mathbf{I}$ and $\mathbf{W} = \mathbf{Z}$, the temporal-preserving layer will reduce to FTA comparison.

Output Layer At the last time step T, the hidden state $\mathbf{s}_T$ accumulates the occurrence information of latent patterns in all previous steps. Thus, each dimension of $\mathbf{s}_T$ represents the expected time step when a latent pattern occurs. To encode the temporal order relationship among different visual patterns, we apply a weighted softmax layer to $\mathbf{s}_T$ to get the output representation $\mathbf{o}$ of video sequence:

$$\mathbf{o} = softmax(\mathbf{Y}\mathbf{s}_T) = \frac{\exp(\mathbf{Y}\mathbf{s}_T)}{\sum_i \exp(\mathbf{Y}(i)\mathbf{s}_T)}, \tag{3.5}$$

where $\mathbf{Y}$ is output weight matrix and $\mathbf{Y}(i)$ is the i-th row of $\mathbf{Y}$. Each row of $\mathbf{Y}$ selects two or more latent patterns and compares their relative temporal order through a learned linear combination. The strength of all the ordinary relationships is normalized to a probability vector by a softmax function. In contrast to FTA, equation 3.5 gives us more flexibility to encode temporal information. The total number of ordinal relationships to encode is controlled by the number of rows in $\mathbf{Y}$, and the number of latent patterns involved in each comparison is controlled by the number of non-zero entries in the rows of $\mathbf{Y}$. An ℓ_1 regularization can be used to control the sparsity of $\mathbf{Y}$.

For a further understanding of the proposed model, we give an intuitive example of how each component works in TPRNN. In the max-pooling layer, the projected value of $\mathbf{x}_t$ indicates the likelihood of visual concepts, such as arm and forearm, appearing in current video frame. The most prominent responses of visual concepts are kept in the state $\mathbf{m}_t$. The temporal-preserving layer projects the visual concepts into some higher-order subspace with $\mathbf{W}$ and $\mathbf{U}$, capturing in-

formation such as the pose of elbow (angle between arm and forearm). When the elbow pose changes significantly (with arms stretching or folding), the gate $\mathbf{g}_t$ will be activated and the current event moment will be memorized in $\mathbf{s}_t$. By aggregating all the time steps when elbow pose changes and comparing with other correlated poses such as shoulder movement, the output representation $\mathbf{o}$ can be useful to characterize videos containing boxing activity which requires joint elbow and shoulder motion. The proposed model mainly relies on dynamic order information to represent and distinguish videos, which is why we call it temporal-preserving network.

3.2.3 Comparison with other RNNs

We compare the proposed network with the other variants in the RNN, which include the conventional LSTM [57] and Gated Recurrent Unit (GRU) [19]. Similar to these models, the proposed TPRNN also employs the activation gate to forget and store useful information in the hidden states. However, compared to both LSTM and GRU, there are several differences in the proposed TPRNN model. A major difference in the proposed TPRNN model is the encoding space. Rather than learning temporal dependencies from the frame-level (spatial) feature space, TPRNN intends to capture temporal order structures directly from the time space. This allows video sequences to be represented from a new perspective that totally different from using the spatial features. If two video sequences contain the same visual concepts but only with different orders (for example, open/close a door), the temporal order information is very useful to distinguish their differences. In such cases, the feature generated by TPRNN can be a great complement for spatial features.

Another straightforward difference is the proposed TPRNN has a simpler structure than LSTM and GRU. Table 3.1 summarizes the distinction between three structures in terms of the number of activation gates and the number of parameters. We can see that LSTM has the most complicated structure with the most activation gates and parameters, while TPRNN only contains less than half

of its parameters with only one gate. This makes TPRNN more invulnerable to overfitting.

Table 3.1: Structural differences between LSTM, GRU and TPRNN. Here n represents hidden states dimension and d represents input dimension.

	# of gates	# of parameters
LSTM	4	$4 * (n * d + n^2)$
GRU	2	$3 * (n * d + n^2)$
TPRNN	1	$2 * n * d + n^2$

3.3 Evaluations for TPRNN

We evaluate the proposed TPRNN representations by performing video classification tasks on two public available datasets: UCF-101 [118] and Charades [115]. The evaluation aims to validate the properties of TPRNN from three aspects. First, we compare TPRNN with conventional LSTM structure to demonstrate the advantage of TPRNN encoding temporal order structures. Then, we analyze the performance of TPRNN on time-sensitive classes. At last, we prove that the TPRNN feature can be a good complement of the spatial features by fusing TPRNN features with frame level features.

3.3.1 Datasets

We employ two video benchmarks to evaluate the proposed TPRNN model: UCF-101 [118] and Charades [115]. UCF-101 dataset includes 13320 videos from 101 action categories with average over 150 frames per video. Each video clip in dataset is segmented to exactly contain one of 101 categories. Charades dataset contains 9848 videos of daily indoors activities with temporal annotations for 157 action classes. Unlike UCF-101, each video clip in Charades dataset may contain multiple action classes in different temporal locations. For UCF-101, We use split-1 with 9537

and 3783 videos as training and testing samples, to conduct our experiments. And for Charades, we exclude videos without any action labels so the final version of the dataset in our experiments contains 7811 training and 1814 testing samples, respectively.

3.3.2 Implementation and Training

We implement both the conventional LSTM and the proposed TPRNN with Theano [128] python math library. Note that the time scale is normalized to 1 for all video clips such that the occurrences of all visual concepts are in $[0, 1]$. The deep features used in the experiments come from several different Convolutional Neural Network (CNN) models including AlexNet [74], VGG [116] and LRCN-single-frame [31]. It is worth mentioning that the VGG models we use to compute RGB (spatial) and flow (temporal) features are provided by [38] which is also fine-tuned on UCF-101, while the adopted AlexNet is pre-trained on ImageNet [107]. All these features are computed with Caffe [66] framework.

We compare the TPRNN features with another two baselines. One is using frame features to represent video clips by averaging across all frames. Another is feeding frame features to LSTM to learn long-term frame dependencies, then average the outputs across all time steps to get video representations. We feed these features into a linear classifier and produce the softmax score for each class. The weights of linear classifier can be learned along with TPRNN by minimizing the cross-entropy loss.

Although TPRNN can be trained along with the CNN in an end-to-end fashion, we fix the CNN weights to compute spatial features and only update weights of LSTM and TPRNN to focus our experiments on the recurrent module. This makes sure we evaluate the impact of encoding temporal order information without changing the spatial inputs. We follow the different frame sampling protocols specified by [38] and [115] on UCF-101 and Charades respectively. During the training

phase, the first frame of each video is randomly cropped with an input size of the CNN networks then the same spatial crop is applied to all frames. Specifically, for Charades dataset, we follow the setup in [115] that only train the models with untrimmed intervals which don't include action localization information for multi-labeled video clips. Unless otherwise specified, we employ the central crops to do the testing and the number of hidden states and batch size are always set to 200 and 16 videos per batch, respectively.

At last, we adopt prediction accuracy as evaluation metric for the single label cases of UCF-101 while we adopt mean average precision to handle multi-label cases of Charades.

3.3.3 LSTM vs. TPRNN

We compare TPRNN with the conventional LSTM by generating video representations with various frame level spatial features. Table 3.2 and 3.3 demonstrate the comparison results on both datasets. From tables we can see the proposed TPRNN outperforms the conventional LSTM with most input spatial features. On UCF-101, we also test both methods additionally with flow image inputs computed by [38]. Table 3.2 shows that TPRNN achieves around 1% better performance than LSTM with all input features from different fully connected layers and RGB/flow images. We can also observe that the improvement from LSTM to TPRNN is more obvious when using RGB frames. This is reasonable since RGB features only contain static spatial information, while flow features already have some local motion information. So TPRNN seems producing less benefit to the flow inputs. During the training, we also find LSTM more sensitive to the overfitting since there are much more weights in LSTM as discussed in section 3.2.3.

Similar results can be observed on Charades dataset, where TPRNN can also outperform LSTM with both fc6 and fc7 features from AlexNet. However, compared to UCF-101, the gap between LSTM and TPRNN is much smaller with features from fc6 than fc7. Note that during the training

phases on the Charades dataset, we perform untrimmed training which doesn't utilize any action localization information provided by training set. Thus learning long-term dependencies between input frames may be not as effective as encoding visual concept occurrences along the time domain, since visual concepts occurrences can be served as some boundary points for trimmed interval presenting interested actions.

Table 3.2: Recognition Accuracy on UCF-101 dataset

UCF-101	LSTM	TPRNN
VGG-16-fc6	0.7766	**0.7861**
VGG-16-fc7	0.7769	**0.7938**
VGG-16-flow-fc6	0.8039	**0.8118**
VGG-16-flow-fc7	0.8007	**0.8107**

Table 3.3: Mean Average Precision on Charades dataset

Charades	LSTM	TPRNN
AlexNet-fc6	0.1027	0.1061
AlexNet-fc7	0.0996	0.1119

3.3.4 Analysis on Subsets

Although we demonstrate the TPRNN can achieve better performance on both datasets, it is beneficial to analyze which video classes can benefit more from temporal features of TPRNN. Based on [38] which achieves over 80% recognition accuracy with only spatial VGG features on UCF-101, we can see most of action types can be discriminated well with only spatial context (e.g. background). In order to evaluate the discriminability based on temporal differences without too much interference of spatial context information, we first reverse the frame orders for all video clips to add another 101 classes to the original dataset (total 202 classes after reverse). So the reversed

classes can only be distinguished by their temporal order differences. Then we train both LSTM and TPRNN plus using only spatial features as baseline on the dataset with fine-tuned VGG-16 model used in section 3.3.3, and test with fc7 layer features for each of the original classes by predicting whether a sample from that class is reversed or not.

Table 3.4: Accuracy of predicting reverse/unreverse on 10 classes of UCF-101

Class	LSTM	TPRNN	Performance Gain
Cliffdiving	0.6410	0.9359	+0.2949
HighJump	0.5676	0.8378	+0.2702
CleanAndJerk	0.5000	0.8182	+0.3182
BalanceBeam	0.5697	0.7742	+0.2045
PoleVault	0.7125	0.7125	+0.0000
CricketBowling	0.5417	0.6667	+0.1240
LongJump	0.6154	0.6538	+0.0384
BlowingCandles	0.5000	0.5152	+0.0152
TennisSwing	0.5000	0.5102	+0.0102
Rowing	0.5000	0.5000	+0.0000
Overall	0.5697	0.6877	+0.1180

We test and report the original/reversed prediction results for 10 representative UCF-101 classes whose video clips are with different level of foreground, background changes and camera variations. For example, video clips in CliffDiving class begin with a background of cliff but end with a background of water, while video clips of BlowingCandles usually have a static background with relatively small region of interest (candle fire, etc.). As it is a binary classification problem, using only spatial features output =0.5 accuracy for all 10 classes, as the spatial features are exactly symmetric for test samples. This indicates it is impossible to distinguish the original/reversed clips without any long-term or temporal information. Other results are summarized in Table 3.4, which includes prediction accuracy and performance gain of TPRNN over LSTM for individual and the overall classes. We can see that LSTM performs obviously better than random guess on around half of classes but still produces poor results for another half. In contrast, TPRNN outperforms

LSTM with a significant margin on most of classes. However, on some classes it performs close to a random guess.

(a) CliffDiving

(b) TennisSwing

(c) Rowing

Figure 3.2: Examples of classes with different foreground and background variations.

We notice that in the classes where TPRNN and LSTM are significantly better than random, video clips usually contain distinct background or camera variations throughout the frames. For Instance, as shown in Figure 3.2a, video clips of CliffDiving class always start with a person standing height. After the action begins, the camera will track the person dropping from height until he/she gets into water. So there will be notable background order difference between original and reversed classes, making them much easier to be classified by TPRNN. In such cases, the action region is very small and is not a crucial factor to classify the video. Besides, for those classes which TPRNN performs similarly poor as TennisSwing (Figure 3.2b), we can see that such actions often take place at some static locations such as tennis ground, etc. Thus background variations contribute very little in temporal order differences and TPRNN will rely on the variations of much smaller action regions

(poses), making it less beneficial for these classes. What's more, Rowing class represents classes whose video clips are temporally symmetric. As shown in Figure 3.2c, the background is almost static and the action region varies periodically. In such cases, encoding the temporal differences may not characterize video clips well and let the TPRNN features perform like random guess. Similar behavior also can be observed on the remaining classes.

Table 3.5: Average Precision (AP) and Mean Average Precision (MAP) of each class pairs and action type group. *sth.* indicates different visual objects

Group 1– Opening/Closing *sth.*

	Spatial	LSTM	TPRNN
Door	0.2671	0.2662	0.3261
Box	0.0410	0.0446	0.0357
Laptop	0.0186	0.0252	0.0440
Closet/Cabinet	0.1612	0.1704	0.1660
Refrigerator	0.1673	0.0984	0.1712
MAP	0.1311	0.1209	0.1486

Group 2– Taking/Putting *sth.* somewhere

	Spatial	LSTM	TPRNN
Bag	0.0441	0.0457	0.0530
Shoes	0.0379	0.0421	0.0525
Sandwich	0.0305	0.0304	0.0367
Blanket	0.0857	0.0806	0.0972
Broom	0.0416	0.0381	0.0482
MAP	0.0480	0.0474	0.0575
Overall MAP	0.0896	0.0842	0.1031

Unlike UCF-101, Charades dataset contains many class pairs that naturally with forward and backward orders, e.g. closing/opening a door. Moreover, the classes in such pairs can be further aggregated into some action types like putting something somewhere and take something from somewhere, etc. These action types share similar temporal order patterns only with different visual concepts. We train all three methods (spatial classifier, LSTM, TPRNN) on entire dataset and

then calculate the Average Precision (AP) for each class. We report testing results on two different pair groups. Each group contains 5 class pairs in forms of the same action types but with different visual objects. For example, action type of group 1 is opening/closing something while the one of group 2 is putting/taking something somewhere. Table 3.5 demonstrates the AP for each pair as well as their Mean Average Precision (MAP) for each group, where the AP of each pair is the average of two classes. As we expected, TPRNN works better in most of class pairs and clearly boost the overall performance.

3.3.5 Fuse with Spatial Features

The TPRNN representation is designed to capture more temporal information and therefore is expected to be a good complement for spatial feature. To verify this argument, we combine it with spatial features and test the performance boost on the same recognition tasks. We compare the spatial, TPRNN features and their late fusion results in Table 3.6 with different frame feature inputs. Note that for spatial results, we average all frame features of each video clip to generate a fixed-size video feature and then perform the classification, while for late fusion, we follow the fusion setting of [117] by averaging the prediction score of Spatial features and TPRNN features to get the final prediction score. For each CNN architecture, we experiment with features from both fc6 and fc7 layer but only report fc7 results since fc6 results are quite similar. We can see that on UCF-101, late fusion with spatial features and TPRNN features achieves different level of boosting. In cases that TPRNN achieves same level performance as spatial features (VGG-16-RGB and flow), late fusion improves about 1% recognition accuracy than single feature, while when TPRNN performs better than spatial ones such as using LRCN-single frame inputs, fusion results are less boosted because of the discrimative gap between two types of features. Similar results can be observed on Charades dataset. Such results coincide with the expectation that the fusion with two features can achieve better performance.

Table 3.6: Late fusion results on UCF101 and Charades

UCF-101	Spatial	TPRNN	Late Fusion
LRCN[31]-single-fc7	0.6952	0.7112	0.7187
VGG-16-RGB-fc7	0.7893	0.7938	0.8057
VGG-16-flow-fc7	0.8096	0.8107	0.8197
Charades	Spatial	TPRNN	Late Fusion
AlexNet-fc7	0.1034	0.1119	0.1136

3.4 TPRNN Summary

This chapter presents a novel Temporal Preserving Recurrent Network (TPRNN) that aims to learn video representation directly from the temporal domain. The proposed network architecture models the temporal order relationships between visual concepts by leveraging their occurrences from spatial feature inputs. The resultant video features provide a new way to characterize video clips with temporal information only. Compared to other RNN structure such as LSTM [57] and GRU [19], TPRNN has simpler inner structure with less parameters which makes it more invunerable to overfitting. The structure design also let the TPRNN overcome the shortcomings that First-Take-All [60] hashing suffers and be able to leverage much more temporal order information in the video representation. We evaluate the proposed TPRNN model on UCF-101 and Charades dataset for action recognition with extensive experiments. The results indicate the proposed TPRNN model outperforms the conventional LSTM and can further improve by combining the spatial features. In particular, significant performance boost is achieved for action classes only are distinguishable by temporal orders.

CHAPTER 4: LEARNING LONG-TERM DEPENDENCIES IN FREQUENCY DOMAIN

Other than order patterns, this chapter focuses on a new pattern type by proposing a novel design for capturing long-term dependencies in frequency domain. Generally speaking, 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 [82]. 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 [88, 5, 52]. Although some sophisticated RNN models such as Long Short-term Memory (LSTM) [57] have been proven as powerful tools for modeling the sequences, there are some cases that are hard to handle by RNNs. For instance, [90] 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 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 chapter is organized as follows. Section 4.1 reviews relevant literature on different RNN-based architectures and their applications. Then we introduce the proposed SFM model in Section 4.2 with its formal definitions and mathematical analysis. Section 4.3 demonstrates the experiment results for different evaluation tasks. Finally, we conclude the chapter in section 4.4.

4.1 Recent Progress for RNN Research

Recurrent Neural Networks (RNNs), which is initially proposed by [35, 67], 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 [7], 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 [9], [96] and [85], 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 [57]. 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 [50, 49, 109]. Recently, [19] 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 [33] employs multiple RNN layers to model the sequence in different time scale. Moreover, [113] and [48] 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 [73] and Phased LSTM [92], 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, [5] and [122] have reached the same level performance as the well-developed systems in machine translation with RNN-encoder-decoder framework; [39] proposes the hierarchical Connectionist Temporal Classification (CTC) [51] network and its deep variants has achieved the state-of-the-art performance for phoneme recognition on TIMIT database [52]. Lastly, [12] reports that RNN yields a better prior for the polyphonic music modeling and transcription.

4.2 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 sequences.

4.2.1 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 4.2.2 for a detailed analysis) as:

$$\mathbf{S}_t = \mathbf{f}_t \circ \mathbf{S}_{t-1} + (\mathbf{g}_t \circ \mathbf{i}_t) \begin{bmatrix} e^{j\omega_1 t} \\ \cdots \\ e^{j\omega_K t} \end{bmatrix}^T \in \mathbb{C}^{D \times K} \tag{4.1}$$

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.

$$\mathrm{Re}\, \mathbf{S}_t = \mathbf{f}_t \circ \mathrm{Re}\, \mathbf{S}_{t-1} + (\mathbf{g}_t \circ \mathbf{i}_t) [\cos \omega_1 t, \cdots, \cos \omega_K t] \tag{4.2}$$

and

$$\mathrm{Im}\, \mathbf{S}_t = \mathbf{f}_t \circ \mathrm{Im}\, \mathbf{S}_{t-1} + (\mathbf{g}_t \circ \mathbf{i}_t) [\sin \omega_1 t, \cdots, \sin \omega_K t] \tag{4.3}$$

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

$$\mathbf{A}_t = |\mathbf{S}_t| = \sqrt{(\mathrm{Re}\,\mathbf{S}_t)^2 + (\mathrm{Im}\,\mathbf{S}_t)^2} \in \mathbb{R}^{D \times K} \tag{4.4}$$

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

$$\angle\mathbf{S}_t = \arctan(\frac{\mathrm{Im}\,\mathbf{S}_t}{\mathrm{Re}\,\mathbf{S}_t}) \in \left[-\frac{\pi}{2}, \frac{\pi}{2}\right]^{D \times K} \tag{4.5}$$

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.

4.2.1.1 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

$$\mathbf{f}_t^{\text{ste}} = \sigma(\mathbf{W}^{\text{ste}}\mathbf{z}_{t-1} + \mathbf{V}^{\text{ste}}\mathbf{x}_t + \mathbf{b}^{\text{ste}}) \in \mathbb{R}^D \tag{4.6}$$

and the frequency forget gate

$$\mathbf{f}_t^{\text{fre}} = \sigma(\mathbf{W}^{\text{fre}}\mathbf{z}_{t-1} + \mathbf{V}^{\text{fre}}\mathbf{x}_t + \mathbf{b}^{\text{fre}}) \in \mathbb{R}^K \tag{4.7}$$

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 $\mathbf{f}_t^{\text{fre}}$

$$\mathbf{f}_t = \mathbf{f}_t^{\text{ste}} \otimes \mathbf{f}_t^{\text{fre}} = \mathbf{f}_t^{\text{ste}} \cdot \mathbf{f}_t^{\text{fre}'} \in \mathbb{R}^{D \times K} \tag{4.8}$$

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

4.2.1.2 Input Gates and Modulations

The input gate can be defined in the similar fashion as

$$\mathbf{g}_t = \sigma(\mathbf{W}_g \mathbf{z}_{t-1} + \mathbf{V}_g \mathbf{x}_t + \mathbf{b}_g) \in \mathbb{R}^D \tag{4.9}$$

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

$$\mathbf{i}_t = \tanh(\mathbf{W}_i \mathbf{z}_{t-1} + \mathbf{V}_i \mathbf{x}_t + \mathbf{b}_i) \in \mathbb{R}^D \tag{4.10}$$

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

4.2.1.3 *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

$$\mathbf{z}_t^k = \mathbf{o}_t^k \circ f_o(\mathbf{W}_\mathbf{z}^k \mathbf{A}_\mathbf{z}^k + \mathbf{b}_\mathbf{z}^k), \text{for } k = 1, \cdots, K \tag{4.11}$$

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,

$$\mathbf{o}_t^k = \sigma(\mathbf{U}_\mathbf{o}^k \mathbf{A}_t^k + \mathbf{W}_\mathbf{o}^k \mathbf{z}_{t-1}^k + \mathbf{V}_\mathbf{o}^k \mathbf{x}_t^k + \mathbf{b}_\mathbf{o}^k) \in \mathbb{R}^M \tag{4.12}$$

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

$$\mathbf{z}_t = \sum_{k=1}^{K} \mathbf{z}_t^k = \sum_{k=1}^{K} \mathbf{o}_t^k \circ f_o(\mathbf{W}_\mathbf{z}^k \mathbf{A}_\mathbf{z}^k + \mathbf{b}_\mathbf{z}^k) \in \mathbb{R}^M \tag{4.13}$$

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.

4.2.2 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. 4.1 can be written into the following final formulation

$$\mathbf{S}_t = (\mathbf{f}_t \circ \mathbf{f}_{t-1} \circ \cdots \circ \mathbf{f}_1) \circ \mathbf{S}_0 + (\mathbf{g}_t \circ \mathbf{i}_t) \begin{bmatrix} e^{j\omega_1 t} \\ \cdots \\ e^{j\omega_K t} \end{bmatrix}^T + \sum_{t'=2}^{t} \mathbf{f}_t \circ \cdots \circ \mathbf{f}_{t'} \circ \mathbf{g}_{t'-1} \circ \mathbf{i}_{t'-1} \begin{bmatrix} e^{j\omega_1 (t'-1)} \\ \cdots \\ e^{j\omega_K (t'-1)} \end{bmatrix}^T \tag{4.14}$$

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

$$\{(\mathbf{f}_t \circ \mathbf{f}_{t-1} \circ \cdots \circ \mathbf{f}_1) \circ \mathbf{S}_0\} \cup \{\mathbf{g}_t \circ \mathbf{i}_t\} \cup \{\mathbf{f}_t \circ \cdots \circ \mathbf{f}_{t'} \circ \mathbf{g}_{t'-1} \circ \mathbf{i}_{t'-1} | t' = 2, \cdots, t\} \tag{4.15}$$

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.

4.2.3 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. 4.14, 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,

$$\boldsymbol{\omega} = \mathbf{W}_{\omega x}\mathbf{x}_t + \mathbf{W}_{\omega z}\mathbf{z}_{t-1} + \mathbf{b}_\omega \tag{4.16}$$

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.

4.3 Evaluations for SFM

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 [57] and the Gated Recurrent Unit (GRU) [19]; while the second group

(BG2) includes latest models like Clockwork RNN (CW-RNN) [73], Recurrent Highway Network (RHN) [146], Adaptive Computation Time RNNs (ACT-RNN) [47], Associative LSTM (A-LSTM) [27] and Phased LSTM (P-LSTM) [92]. 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 input sequences (CW-RNN, P-LSTM).

Table 4.1: 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 4.3.1), polyphonic music prediction (sec 4.3.2) and phone classification (sec 4.3.3), 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	-

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 4.2, 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 4.1. We implement

[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.

the proposed SFM model using Theano Python Math Library [127].

Unless otherwise specified, we train all the networks through the BPTT algorithm with the AdaDelta optimizer [142], 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.

4.3.1 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.

$$y_{square}(t) = \frac{4A}{\pi} \sum_{n=1,3,5,\cdots}^{\infty} \frac{1}{n} \sin(\frac{2n\pi}{T}(t+P)) + V, t \in [0, L] \tag{4.17}$$

$$y_{sawtooth}(t) = \frac{A}{2} - \frac{A}{\pi} \sum_{n=1}^{\infty} \frac{1}{n} \sin(\frac{2n\pi}{T}(t+P)) + V, t \in [0, L] \tag{4.18}$$

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 4.1 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.

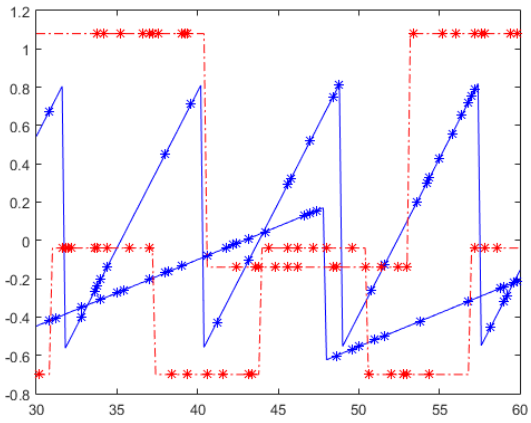


Figure 4.1: Several examples of the generated waves on the interval $[30, 60]$ with different periods, amplitudes, and phases. The red dash lines represent the square waves while the blue solid lines represent square waves. The '$*$' markers indicate the sampled data points that are used for training and testing.

We compare the proposed SFM with all BG1 and BG2 baselines and summarize the prediction results in Figure 4.2. The result shows by explicitly modeling the frequency patterns of these two types of sequences, both SFM and Adaptive SFM achieve best performance. In particular, the Adaptive SFM has achieved 0.9975 in accuracy – almost every signal has been classified correctly.

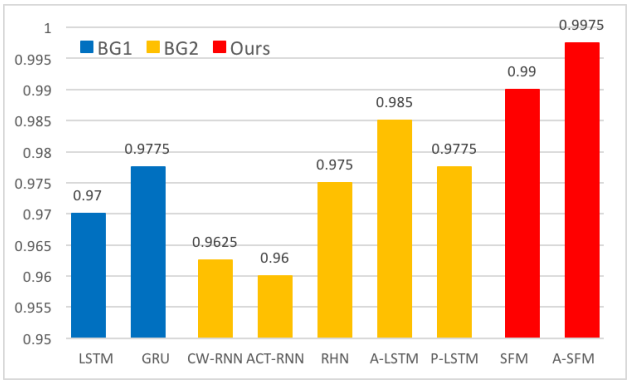


Figure 4.2: Signal type prediction accuracy of each model.

4.3.2 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 [12]. 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 [12]: MuseData, JSB chorales [2], Piano-midi.de [98] 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 [12].

- **RNN-RBM:** Proposed by [12], RNN-RBM is a modification of RTRBM [121] by combining

a full RNN with the Restrict Boltzmann Machine.

- **RNN-NADE-HF:** RNN-NADE [76] model with the RNN layer pretrained by the Hessian-free (HF) [85]

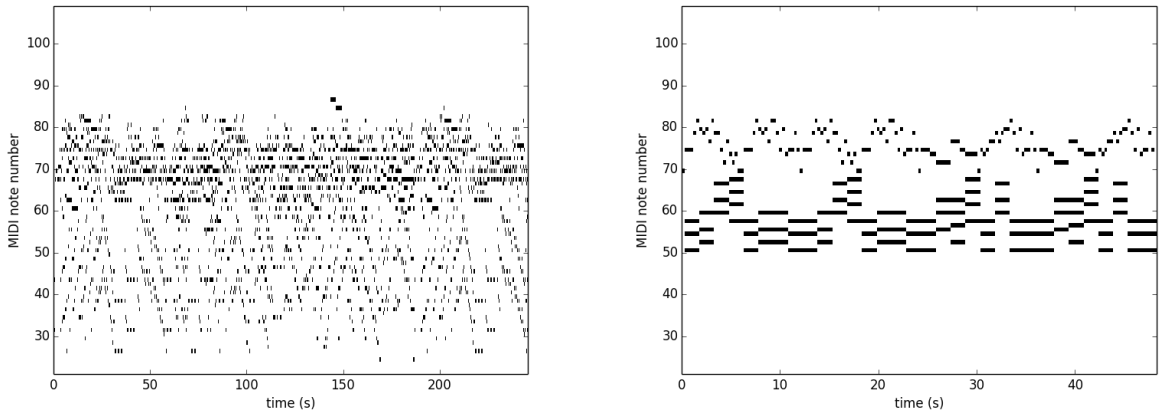
Table 4.2: Log-likelihood on the four music datasets. The last two columns contain the results from the proposed SFM models.

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	$\mathbf{-4.80}$
JSB chorales	-6.24	-6.14	-6.04	-5.89	-5.74	-5.63	-5.65	-6.27	-5.56	-5.47	$\mathbf{-5.45}$
Piano-midi.de	-7.27	-7.14	-7.83	-7.41	-7.58	-6.96	-7.02	-7.09	-7.05	$\mathbf{-6.76}$	-6.80
Nottingham	-5.60	-5.63	-5.90	-5.82	-5.60	-5.64	-5.70	-2.39	$\mathbf{-2.31}$	-5.67	-5.63

We directly report the results of these methods from [12]. We follow the same protocol [12] 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 [12], and we adopt it to make a direct comparison across the models. The results are reported in the Table 4.2. 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.

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



(a) MuseData (b) Nottingham

Figure 4.3: 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.

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 4.3 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.

4.3.3 Phoneme Classification

Finally, we evaluate the proposed SFM model by conducting the frame level phoneme classification task introduced by [48]. 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.

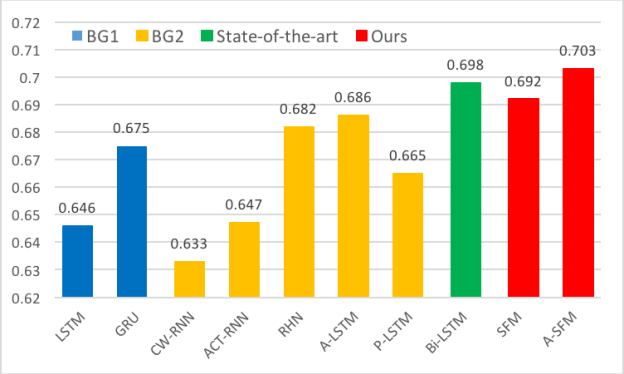


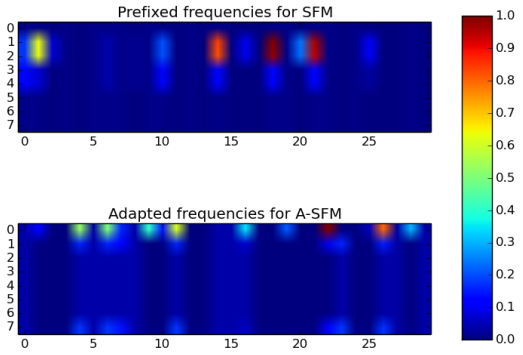
Figure 4.4: Accuracy for frame-level phoneme classification on TIMIT dataset.

We perform the classification task on TIMIT speech corpus [43]. We preprocess the dataset in the same way as [48]. 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 [48], we use the original phone set of 61 phonemes instead of mapping them into a smaller set [106].

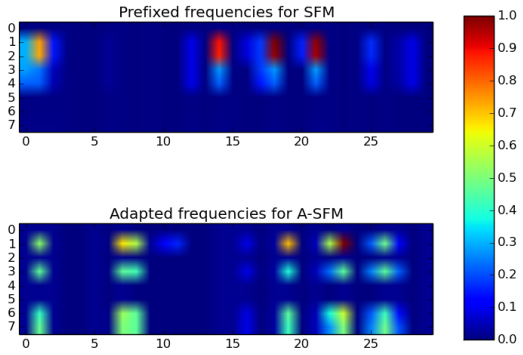
We train the proposed and compared models by following the standard splitting of training and test sets for the TIMIT dataset [43]. In addition, we randomly select 184 utterances from the training set as the validation set and keep the rest for training.

Figure 4.4 compares the classification accuracy by different models. In addition, we include the result of the bidirectional LSTM (Bi-LSTM) [48] for comparison. From the figure, we can see that both SFM and Adaptive SFM outperform the BG1 and BG2 baselines with approximately the

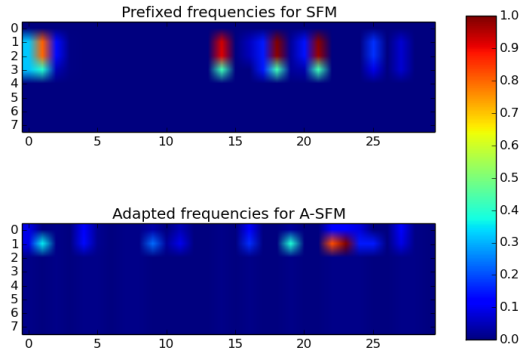
same number of parameters. Especially when compared with the conventional LSTM, CW-RNN and ACT-RNN, the proposed SFM models have significantly improved the performance by more than 5%. This demonstrates the advantages on explicitly modeling the frequency patterns in short-range windows, which plays a 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.



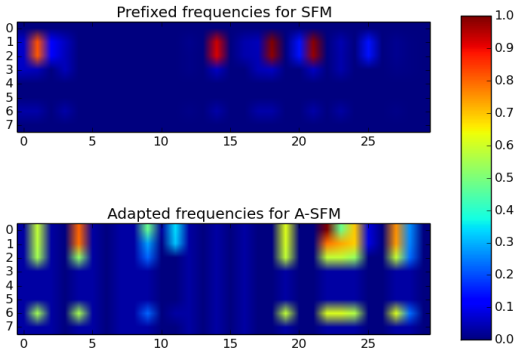
(a) At $\frac{T}{4}$



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



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



(d) At T

Figure 4.5: The amplitudes of SFM matrices for both the prefixed (SFM) and adaptive (A-SFM) frequencies. For all subfigures, each row represents a frequency component.

Additionally, we find adapting frequencies, which is the main difference between the Adaptive SFM and SFM, 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 4.5 illustrates the matrix amplitudes of both the SFM and Adaptive SFM at time $\frac{T}{4}$, $\frac{T}{2}$, $\frac{3T}{4}$ and T, from which the two networks demonstrate distinct ways to model the sequence. Based on section 4.2.3, the frequency set of the SFM keeps the same across the time. And in all subfigures of Figure 4.5, 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 step. Under such conditions, modeling the sequence calls for more frequency components, which are varied dramatically across the time as shown in Figure 4.5. Compared with the SFM, the Adaptive SFM is able to model richer frequency patterns.

4.4 Summarization for SFM

In this chapter, 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 out-

perform various classic and latest LSTM models as well as reaching the competitive performance compared to the state-of-the-art methods on each benchmark.

CHAPTER 5: LEARNING TO ADAPTIVELY ADJUST RECURRENT NEURAL NETS

In this final chapter for techniques, we further explore the way of modeling frequency patterns by proposing a novel paradigm to filter out the noisy frequency information based on dynamic temporal contexts. As we know, Recurrent Neural Networks (RNNs) play a critical role in sequential modeling as they have achieved impressive performances in various tasks [14][16][20][93]. Yet learning long-term dependencies from long sequences still remains a very difficult task [8] [58][140][61]. Among various ways that try to handle this problem, modeling multiscale patterns seem to be a promising strategy since many multiscale RNN structures perform better than other non-scale modeling RNNs in multiple applications [72][93][20][16][14][15]. Multiscale RNNs can be roughly divided into two groups based on their design philosophies. The first group trends to modeling scale patterns with the hierarchical architectures and prefixed scales for different layers. This may lead to at least two disadvantages. First, the prefixed scale can not be adjusted to fit the temporal dynamics throughout the time. Although patterns in different scale levels require distinct frequencies to update, they do not always stick to a certain scale and could vary at different time steps. For example, in polyphonic music modeling, distinguishing different music styles demands RNNs to model various emotion changes throughout music pieces. While emotion changes are usually controlled by the lasting time of notes, it is insufficient to model such patterns using only fixed scales as the notes last differently at different time. Secondly, stacking multiple RNN layers greatly increases the complexity of the entire model, which makes RNNs even harder to train. Unlike this, another group of multiscale RNNs models scale patterns through gate structures [93][14][99]. In such cases, additional control gates are learned to optionally update hidden for each time step, resulting in a more flexible sequential representations. Yet such modeling strategy may not remember information which is more important for future outputs but less related to

current states.

In this chapter, we aim to model the underlying multiscale temporal patterns for time sequences while avoiding all the weaknesses mentioned above. To do so, we present Adaptively Scaled Recurrent Neural Networks (ASRNNs), a simple extension for existing RNN structures, which allows them to adaptively adjust the scale based on temporal contexts at different time steps. Using the causal convolution proposed by [133], ASRNNs model scale patterns by firstly convolving input sequences with wavelet kernels, resulting in scale-related inputs that parameterized by the scale coefficients from kernels. After that, scale coefficients are sampled from categorical distributions determined by different temporal contexts. This is achieved by sampling Gumbel-Softmax (GM) distributions instead, which are able to approximate true categorical distributions through the re-parameterization trick. Due to the differentiable nature of GM, ASRNNs could learn to flexibly determine which scale is most important to target outputs according to temporal contents at each time step. Compared with other multiscale architectures, the proposed ASRNNs have several advantages. First, there is no fixed scale in the model. The subroutine for scale sampling can be trained to select proper scales to dynamically model the temporal scale patterns. Second, ASRNNs can model multiscale patterns within a single RNN layer, resulting in a much simpler structure and easier optimization process. Besides, ASRNNs do not use gates to control the updates of hidden states. Thus there is no risk of missing information for future outputs.

To verify the effectiveness of ASRNNs, we conduct extensive experiments on various sequence modeling tasks, including low density signal identification, long-term memorization, pixel-to-pixel image classification, music genre recognition and language modeling. Our results suggest that ASRNNs can achieve better performances than their non-adaptively scaled counterparts and are able to adjust scales according to various temporal contents. We organize the rest chapter like this: the first following section reviews relative literatures, then we introduce ASRNNs with details in next section; after that the results for all evaluations are presented, and the last section concludes

the chapter.

5.1 Literature Study for Multiscale RNNs

As a long-lasting research topic, the difficulties of training RNNs to learn long-term dependencies are considered to be caused by several reasons. First, the gradient exploding and vanishing problems during back propagation make training RNNs very tough [8] [58]. Secondly, RNN memory cells usually need to keep both long-term dependencies and short-term memories simultaneously, which means there should always be trade-offs between two types of information. To overcome such problems, some efforts aim to design more sophisticated memory cell structures. For example, Long-short term memory (LSTM) [57] and gated recurrent unit (GRU) [19], are able to capture more temporal information; while some others attempt to develop better training algorithms and initialization strategies such as gradient clipping [95], orthogonal and unitary weight optimization [3][77] [136][100][135][101] etc. These techniques can alleviate the problem to some extent [125][80][134].

Meanwhile, previous works like [72] [93] [20] [124] [62] suggest learning temporal scale structures is also the key to this problem. This stands upon the fact that temporal data usually contains rich underlying multiscale patterns [111][91] [34] [81] [59]. To model multiscale patterns, a popular strategy is to build hierarchical architectures. These RNNs such as hierarchical RNNs [34], clockwork RNNs [72] and Dilated RNNs [16] etc, contain hierarchical architectures whose neurons in high-level layers are less frequently updated than those in low-level layers. Such properties fit the natures of many latent multiscale temporal patterns where low-level patterns are sensitive to local changes while high-level patterns are more coherent with the temporal consistencies. Instead of considering hierarchical architectures, some multiscale RNNs model scale patterns using control gates to decide whether to update hidden states or not at a certain time step. Such struc-

tures like phased LSTMs [93] and skip RNNs [14], are able to adjust their modeling scales based on current temporal contexts, leading to more reasonable and flexible sequential representations. Recently, some multiscale RNNs like hierarchical multi-scale RNNs [20], manage to combine the gate-controlling updating mechanism into hierarchical architectures and has made impressive progress in language modeling tasks. Yet they still employ multi-layer structures which make the optimization not be easy.

5.2 Adaptively Scaled Recurrent Neural Networks

In this section we introduce Adaptively Scaled Recurrent Neural Networks (ASRNNs), a simple but useful extension for various RNN cells that allows to dynamically adjust scales at each time step. An ASRNNs is consist of three components: scale parameterization, adaptive scale learning and RNN cell integration, which will be covered in following subsections.

5.2.1 Scale Parameterization

We begin our introduction for ASRNNs with scale parameterization. Suppose $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2 \cdots, \mathbf{x}_T]$ is an input sequence where $\mathbf{x}_t \in \mathcal{R}^n$. At time t, instead of taking only the current frame $\mathbf{x}_t$ as input, ASRNNs compute an alternative scale-related input $\tilde{\mathbf{x}}_t$, which can be obtained by taking a causal convolution between the original input sequence $\mathbf{X}$ and a scaled wavelet kernel function ϕ_{j_t}.

More specifically, let J be the number of considered scales. Consider a wavelet kernel ϕ of size K. At any time t, given a scale $j_t \in \{0, \cdots, J-1\}$, the input sequence $\mathbf{X}$ is convolved with a

scaled wavelet kernel $\phi_{j_t} = \phi(\frac{i}{2^{j_t}})$. This yields the following scaled-related input $\tilde{\mathbf{x}}_t$ at time t

$$\tilde{\mathbf{x}}_t = (\mathbf{X} * \phi_{j_t})_t = \sum_{i=0}^{2^{j_t}K-1} \mathbf{x}_{t-i}\phi(\frac{i}{2^{j_t}}) \in \mathcal{R}^n \tag{5.1}$$

where for any $i \in \{t - 2^{j_t}K + 1, \cdots, t-1\}$, we manually set $\mathbf{x}_i = \mathbf{0}$ iff $i \leq 0$. And the causal convolution operator $*$ [133] is defined to avoid the resultant $\tilde{\mathbf{x}}_t$ depending on future inputs. We also let $\phi(\frac{i}{2^{j_t}}) = 0$ iff $2^{j_t} \nmid i$. It is easy to see that $\tilde{\mathbf{x}}_t$ can only contain information from $\mathbf{x}_{t-i}$ when $i = 2^{j_t}k, k \in \{1, \cdots, K\}$. In other words, there are skip connections between $\mathbf{x}_{t-2^{j_t}(k-1)}$ and $\mathbf{x}_{t-2^{j_t}k}$ in the scale j_t. While j_t becomes larger, the connections skip further.

It is worth mentioning that the progress for obtaining scale-related input $\tilde{\mathbf{x}}_t$ is quite similar as the convolutions with the real waveforms in [133]. By stacking several causal convolutional layers, [133] is able to model temporal patterns in multiple scale levels with its exponential-growing receptive field. However, such abilities are achieved through a hierarchical structure where each layer is given a fixed dilation factor that does not change through out time. To avoid this, we replace the usual convolution kernels with wavelet kernels, which come with scaling coefficients just like j_t in equation 5.1. By varying j_t, $\tilde{\mathbf{x}}_t$ is allowed to contain information from different scale levels. Thus we call it scale parameterization. We further demonstrate it's possible to adaptively control j_t based on temporal contexts through learning, which will be discussed in subsection 5.2.2.

5.2.2 Adaptive Scale Learning

To adjust scale j_t at different time t, we need to sample j_t from a categorical distribution where each class probability is implicitly determined by temporal contexts. However, it is impossible to directly train such distributions along with deep neural networks because of the non-differentiable nature of their discrete variables. Fortunately, [64] [83] propose Gumbel-Softmax (GM) distribution,

a differentiable approximation for a categorical distribution that allow gradients to be back propagated through its samples. Moreover, GM employs the re-parameterization trick, which divides the distribution into a basic independent random variable and a deterministic function. Thus, by learning the function only, we can bridge the categorical sampling with temporal contexts through a differentiable process.

Now we introduce the process of learning to sample scale j_t with more details. Suppose $\boldsymbol{\pi}_t = [\pi_0^t, \cdots, \pi_{J-1}^t] \in [0,1]^J$ are class probabilities for scale set $\{0, \cdots, J-1\}$ and $\mathbf{z}_t = [z_0^t, \cdots, z_{J-1}^t] \in \mathcal{R}^J$ are some logits related to temporal contexts at time t. The relationship between $\boldsymbol{\pi}_t$ and $\mathbf{z}_t$ can be written as

$$\pi_i^t = \frac{\exp(z_i^t)}{\sum_{i'=0}^{J-1} \exp(z_{i'}^t)} \tag{5.2}$$

where $i \in \{0, \cdots, J-1\}$. Let $\mathbf{y}_t = [y_0^t, \cdots, y_{J-1}^t] \in [0,1]^J$ be a sample from GM. Based on [64], y_i^t for $i = 0, \cdots, J-1$ can be calculated as

$$y_i^t = \frac{\exp((\log \pi_i^t + g_i)/\tau)}{\sum_{i'=0}^{J-1} \exp((\log \pi_{i'}^t + g_{i'})/\tau)} \tag{5.3}$$

where $g_0, \cdots, g_{J-1}$ are i.i.d. samples drawn from the basic Gumbel$(0,1)$ distribution and τ controls how much the GM is close to a true categorical distribution. In other words, as τ goes to 0, $\mathbf{y}_t$ would become $\mathbf{j}_t$, the one-hot vector whose j_tth value is 1.

Thus with GM, it is clear that the sampled $\mathbf{j}_t$ is approximated by a differentiable function of $\mathbf{z}_t$. We further define $\mathbf{z}_t$ with the hidden states $\mathbf{h}_{t-1} \in \mathcal{R}^m$ and input $\mathbf{x}_t \in \mathcal{R}^n$ as

$$\mathbf{z}_t = \mathbf{W}_z \mathbf{h}_{t-1} + \mathbf{U}_z \mathbf{x}_t + \mathbf{b}_z \in \mathcal{R}^J \tag{5.4}$$

where $\mathbf{W}_z, \mathbf{U}_z$ are weight matrices and $\mathbf{b}_z$ is bias vector. Combing equations 5.2, 5.3 and 5.4, we achieve our goal of dynamically changing j_t by sampling from GM distributions that parameterized

by $\mathbf{h}_{t-1}$ and $\mathbf{x}_t$. Since the entire procedure is differentiable, matrices $\mathbf{W}_z$ and $\mathbf{U}_z$ can be optimized along with other parameters of ASRNNs during the training.

5.2.3 Integrating with Different RNN Cells

With both the techniques introduced in previously introduced two subsections, we are ready to incorporate the proposed adaptive scaling mechanism with different RNN cells, resulting in various forms of ASRNNs. Since both $\tilde{\mathbf{x}}_t$ and sampling for j_t don't rely on any specific memory cell designs, it's straightforward to do so by replacing original input frames $\mathbf{x}_t$ with $\tilde{\mathbf{x}}_t$. For example, a ASRNN with LSTM cells can be represented as

$$\mathbf{f}_t, \mathbf{i}_t, \mathbf{o}_t = \text{sigmoid}(\mathbf{W}_{f,i,o}\mathbf{h}_{t-1} + \mathbf{U}_{f,i,o}\tilde{\mathbf{x}}_t + \mathbf{b}_{f,i,o}) \in \mathcal{R}^m \tag{5.5}$$

$$\mathbf{g}_t = \tanh(\mathbf{W}_g\mathbf{h}_{t-1} + \mathbf{U}_g\tilde{\mathbf{x}}_t + \mathbf{b}_g) \in \mathcal{R}^m \tag{5.6}$$

$$\mathbf{c}_t = \mathbf{f}_t \circ \mathbf{c}_{t-1} + \mathbf{i}_t \circ \mathbf{g}_t \tag{5.7}$$

$$\mathbf{h}_t = \mathbf{o}_t \circ \tanh(\mathbf{c}_t) \tag{5.8}$$

while a ASRNN with GRU cells can be written as

$$\mathbf{z}_t, \mathbf{r}_t = \text{sigmoid}(\mathbf{W}_{z,r}\mathbf{h}_{t-1} + \mathbf{U}_{z,r}\tilde{\mathbf{x}}_t + \mathbf{b}_{z,r}) \in \mathcal{R}^m \tag{5.9}$$

$$\mathbf{g}_t = \tanh(\mathbf{W}_g(\mathbf{r}_t \circ \mathbf{h}_{t-1}) + \mathbf{U}_g\tilde{\mathbf{x}}_t + \mathbf{b}_g) \in \mathcal{R}^m \tag{5.10}$$

$$\mathbf{h}_t = \mathbf{z}_t \circ \mathbf{h}_{t-1} + (1 - \mathbf{z}_t) \circ \mathbf{g}_t \tag{5.11}$$

where $\mathbf{W}_*, \mathbf{U}_*$ are weight matrices and $\mathbf{b}_*$ are bias vectors, and $\circ$ means element-wise multiplication. For rest of this chapter, we use ASLSTMs to refer those integrated with LSTM cells, ASGRUs for those integrated with GRU cells and so on and so forth. We still call them ASRNNs when there is no specified cell types. It is also worth mentioning that a conventional RNN cell is the special case of its ASRNN counterpart when $J = K = 1$.

5.2.4 Discussion

Finally, we briefly analyze the advantages of ASRNNs over other multiscale RNN structures. As mentioned at the beginning of this chapter, there are many RNNs, including hierarchical RNNs [34] and Dilated RNNs [16] etc, that apply hierarchical architectures to model multiscale patterns. Compared to them, the advantages of ASRNNs are clear. First, ASRNNs are able to model patterns with multiple scale levels within a single layer, making their spatial complexity much lower than hierarchical structures. Although hierarchical models may reduce the neuron numbers for each layer to have the similar size as single layer ASRNNs, they are harder to train with deeper structures. What's more, compared with the fixed scales for different layers, adapted scales are easier to capture underlying patterns as they can be adjusted based on temporal contexts at different time steps.

Besides, other multiscale RNN models like phased LSTMs [93] and skip RNNs [14] etc, build gate structures to manage scales. Such gates are learned to determine whether to remember the incoming information at each time. However, this may lose information which is important for future time but not for current time. This problem would never happen to ASRNNs as according to

equation 5.1, the current input $\mathbf{x}_t$ will always be included in $\tilde{\mathbf{x}}$ and $\mathbf{h}_t$ is updated every step. Thus there is no risk for ASRNNs to lose critical information. This is an important property especially for tasks with frame labels. In such cases previously irrelevant information may become necessary for later frame outputs. Thus information from every frame should be leveraged to get correct outputs at different time.

5.3 Experiments for Evaluating ASRNNs

In this section, we evaluate the proposed ASRNNs with five sequence modeling tasks: low density signal type identification, copy memory problem, pixel-to-pixel image classification, music genre recognition and word level language modeling. We also explore how the scales would be adapted along time. Unless specified otherwise, all the models are implemented using Tensorflow library [1]. We train all the models with the RMSProp optimizer [129] and set learning rate and decay rate to 0.001 and 0.9, respectively. It is worth mentioning that there is no techniques such as recurrent batch norm [114] and gradient clipping [95] applied during the training. All the weight matrices are initialized with glorot uniform initialization [45]. For ASRNNs, we choose Haar wavelet as default wavelet kernels, and set τ of Gumbel-Softmax to 0.1. We integrate ASRNNs with two popular RNN cells, LSTM [57] and GRU [19] and use their conventional counterparts as common baselines. Besides, the baselines also include scaled RNNs (SRNNs), a simplified version that every j_t is set to $J-1$. Additional baselines for individual tasks will be stated in the corresponding subsections if there are. For both SRNNs and ASRNNs, The maximal considered scale J and wavelet kernel size K are set to 4 and 8, respectively.

5.3.1 Low Density Signal Type Identification

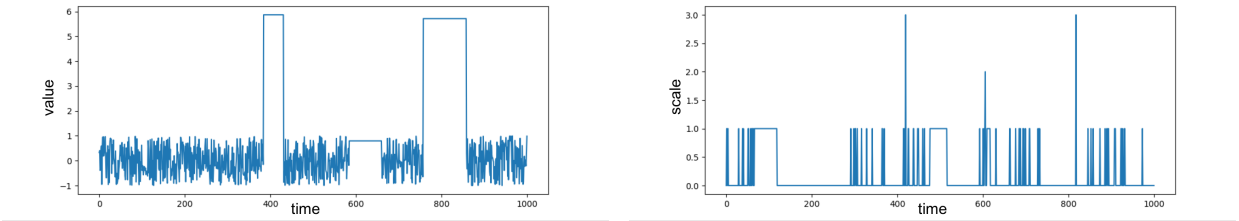
We begin our evaluation for ASRNNs with some synthetic data. The first task is low density signal type identification, which demands RNNs to distinguish the type of a long sequence that only contains limited useful information. More specifically, consider a sequence with length of 1000, first we randomly choose p subsequences at arbitrary locations of the sequence where $p \in \{3, 4, 5\}$. Each subsequence has different length T where $T \in \mathbb{Z}^+ \cap [20, 100]$ and we make sure that subsequences don't overlap with each other. For one sequence, all of its subsequences belong to one of the three types of waves: square wave, saw-tooth wave and sine wave, but with different amplitude A sampled from $[-7, 7]$. The rests of the sequence are filled with random noises sampled from $(-1, 1)$. The target is to identify which type of wave a sequence contains. Apparently, a sequence carries only $6\% \sim 50\%$ useful information, requiring RNNs capable of locating it efficiently.

Table 5.1: Accuracies for ASRNNs and baselines .

ACCURACY (%)	RNN	SRNN	ASRNN
LSTM	81.3	83.6	97.7
GRU	84.1	88.1	98.0

Following above criterion, we randomly generate 2000 low density sequences for each type. We choose 1600 sequences per type for training and the remaining are for testing. Table 5.1 demonstrates the identification accuracies for baselines and ASRNNs. We can see the accuracies of both ASLSTM and ASGRU are over 97.5%, meaning they have correctly identified the types for most of sequences without being moderated by noise. Considering the much lower performance of baselines, it's confident to say that ASRNNs are able to efficiently locate useful information with adapted scales. Besides, we also observe there are similar patterns among some waves and their

scale variation sequences. Figure 5.1 gives such an example, from which we see the scale 0 and 1 are more related to noises while the scale 2 and 3 only appear in the region with square form information. Moreover, the subsequence where the scale 2 is located is harder to identify as its values are too close to the noise. We believe such phenomena implies the scale variations could reflect some certain aspects that are helpful for understanding underlying temporal patterns.



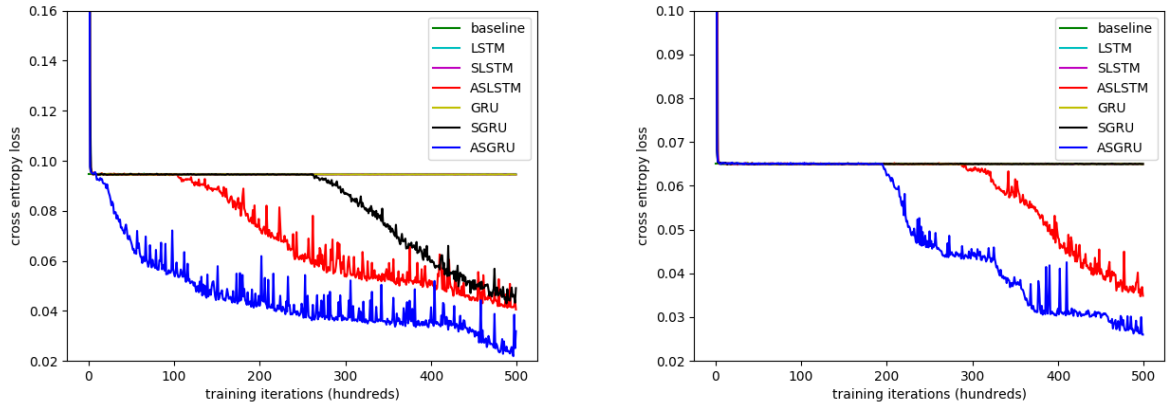
(a) A square wave sample.

(b) The corresponding scale variations.

Figure 5.1: The similar patterns between a raw square wave and its scale variations.

5.3.2 Copy Memory Problem

Next we revisit the copy memory problem, one of the original LSTM tasks proposed by [57] to test the long-term dependency memorization abilities for RNNs. We closely follow the experimental setups used in [3] [136]. For each input sequence with $T + 20$ elements, The first ten are randomly sampled from integers 0 to 7. Then the rest of elements are all set to 8 except the $T + 10$th to 9, indicating RNNs should begin to replicate the first 10 elements from now on. The last ten values of output sequence should be exactly the same as the first ten of the input. Cross entropy loss is applied for each time step. In addition to common baselines, we also adopt the memoryless baseline proposed by [3]. The cross entropy of this baseline is $\frac{10 \log(8)}{T+20}$, which means it always predict 8 for first $T + 10$ steps while give a random guess of 0 to 7 for last 10 steps. For each T, we generate 10000 samples to train all RNN models.



(a) $T = 200$. (b) $T = 300$.

Figure 5.2: Cross entropies for copy memory problem. Best viewed in colors.

Figure 5.2 demonstrates the cross entropy curves of baselines and ASRNNs. We notice that both LSTM and GRU get stuck at the same cross entropy level with the memoryless baseline during the entire training process for both $T = 200$ and $T = 300$, indicating both LSTM and GRU are incapable of solving the problem with long time delays. This also agrees with the results reported in [53] and [3]. For SRNNs, it seems like fixed scales are little helpful since only the SGRU at $T = 200$ can have a lower entropy after 250 hundred steps. Unlike them, cross entropies of ASRNNs are observed to further decrease after certain steps of staying with baselines. Especially for $T = 200$, ASGRU almost immediately gets the entropy below the baseline with only a few hundreds of iterations passed. Besides, comparing figure 5.2a and 5.2b, ASGRUs are more resistant to the increasing of T as ASLSTMs need more time to wait before they can further reduce cross entropies. Overall, such behaviors prove ASRNNs have stronger abilities for memorizing long-term dependencies than baselines.

Table 5.2: Classification accuracies for pixel-to-pixel MNIST. N stands for the number of hidden states. Italic numbers are results reported in the original papers. Bold numbers are best results for each part. ACC=accuracy, UNP/PER means the unpermuted/permuted cases respectively.

RNN	N	# OF WEIGHTS	MIN. SCALE	MAX. SCALE	AVG. SCALE	UNP ACC(%)	PER ACC(%)
LSTM	129	$\approx$ 68K	0	0	0	97.1	89.3
SLSTM	129	$\approx$ 68K	3	3	3	97.4	87.7
ASLSTM	128	$\approx$ 68K	0	3	0.92	**98.3**	**90.8**
GRU	129	$\approx$ 51K	0	0	0	96.4	90.1
SGRU	129	$\approx$ 51K	3	3	3	97.0	89.8
ASGRU	128	$\approx$ 51K	0	3	0.75	**98.1**	**91.2**
TANH-RNN [77]	100	-	-	-	-	*35.0*	*33.0*
URNN [3]	512	$\approx$ 16K	-	-	-	*95.1*	*91.4*
FULL-CAPACITY URNN [136]	512	$\approx$ 270K	-	-	-	*96.9*	*94.1*
IRNN [77]	100	-	-	-	-	*97.0*	*82.0*
SKIP-LSTM [14]	110	-	-	-	-	*97.3*	-
SKIP-GRU [14]	110	-	-	-	-	*97.6*	-
STANH-RNN [143]	64	-	-	-	-	*98.1*	*94.0*
RECURRENT BN-RNN [21]	100	-	-	-	-	**99.0**	**95.4**

5.3.3 Pixel-to-Pixel Image Classification

Now we proceed our evaluation for ASRNNs with real world data. In this subsection, we study the pixel-to-pixel image classification problem using MNIST benchmark [78]. Initially proposed by [77], it reshapes all 28×28 images into pixel sequences with length of 784 before fed into RNN models, resulting in a challenge task where capturing long term dependencies is critical. We follow the standard data split settings and only feed outputs from the last hidden state to a linear classifier [138]. We conduct experiments for both unpermuted and permuted settings.

Table 5.2 summarizes results of all experiments for pixel-to-pixel MNIST classifications. The first two blocks are the comparisons between common baselines and ASRNNs with different cell struc-

tures. Their numbers of weights are adjusted to keep approximately same in order to be compared fairly. We also include other state-of-the-art results of single layer RNNs in the third block. It is easy to see that both SRNNs and ASRNNs achieve better performances than conventional RNNs with scale-related inputs on both settings. This is probably because causal convolutions between inputs and wavelet kernels can be treated as a spatial convolutional layer, allowing SRNNs and ASRNNs to leverage information that is spatially local but temporally remote. Moreover, the adapted scales help ASRNNs further reach the state-of-the-art performances by taking dilated convolutions with those pixels that more spatially related to the current position. It is also worth mentioning the proposed dynamical scaling is totally compatible with the techniques from the third part of the table 5.2 such as recurrent batch normalization [21] and recurrent skip coefficients [143]. Thus ASRNNs can also benefit from them as well.

5.3.4 Music Genre Recognition

The next evaluation mission for ASRNNs is music genre recognition (MGR), a critical problem in the music information retrieval (MIR) [86] which requires RNNs to characterize the similarities between music tracks across many aspects such as cultures, artists and ages. Compared to other acoustic modeling tasks like speech recognition, MGR is considered to be more difficult as the boundaries between genres are hard to distinguish due to different subjective feelings among people [110]. We choose free music archive (FMA) dataset [28] to conduct our experiments. More specifically, we use the FMA-small, a balanced FMA subset containing 8000 music clips that distributed across 8 genres, where each clip lasts 30 seconds with sampling rate of 44100 Hz. We follow the standard $80/10/10\%$ data splitting protocols to get training, validation and test sets. We compute 13-dimensional log-mel frequency features (MFCC) with 25ms windows and 10ms frame steps for each clip, resulting in very long sequences with about 3000 entries. Besides, inspired by recent success of [133] and [108], we are also encouraged to directly employ raw audio waves as

inputs. Due to limited computational resources, we have to reduce the sampling rate to 200 Hz for raw music clips while resultant sequences are still two times longer than MFCC sequences.

Table 5.3: Music genre recognition on FMA-small. N stands for the number of hidden states. ACC=accuracy.

FEATURES	METHODS	N	# OF WEIGHTS	MIN. SCALE	MAX. SCALE	AVG. SCALE	ACC(%)
MFCC	LSTM	129	$\approx$ 74K	0	0	0	37.1
	SLSTM	129	$\approx$ 74K	3	3	3	37.7
	ASLSTM	128	$\approx$ 74K	0	3	1.34	**40.9**
	GRU	129	$\approx$ 56K	0	0	0	38.2
	SGRU	129	$\approx$ 56K	3	3	3	38.5
	ASGRU	128	$\approx$ 56K	0	3	1.39	**42.4**
	MFCC+GMM [4]	-	-	-	-	-	21.3
RAW	LSTM	129	$\approx$ 68K	0	0	0	18.5
	SLSTM	129	$\approx$ 68K	3	3	3	18.9
	ASLSTM	128	$\approx$ 68K	0	3	1.47	**20.1**
	GRU	129	$\approx$ 51K	0	0	0	18.8
	SGRU	129	$\approx$ 51K	3	3	3	18.4
	ASGRU	128	$\approx$ 51K	0	3	1.59	**19.5**
	RAW+CNN [30]	-	-	-	-	-	17.5

We demonstrate all the MGR results on FMA-small in the Table 5.3. Besides RNN models, we also include two baselines without temporal modeling abilities (GMM for MFCC and CNN for raw). We can see when using MFCC features, both the ASLSTM and ASGRU can outperform SRNNs and their conventional counterparts with about $3 \sim 4\%$ improvements. This is an encouraging evidence to show how adapted scales can boost the modeling capabilities of RNNs for MGR. However, the recognition accuracies drop significantly for all models when applying raw audio waves as inputs. In such cases, the gains from adapted scales are marginal for both the ASLSTM and ASGRU. We believe it is due to the low sampling rate for raw music clips since too much information is lost. However, increasing sampling rate will significantly rise the computational costs and make it eventually prohibitive for training RNNs.

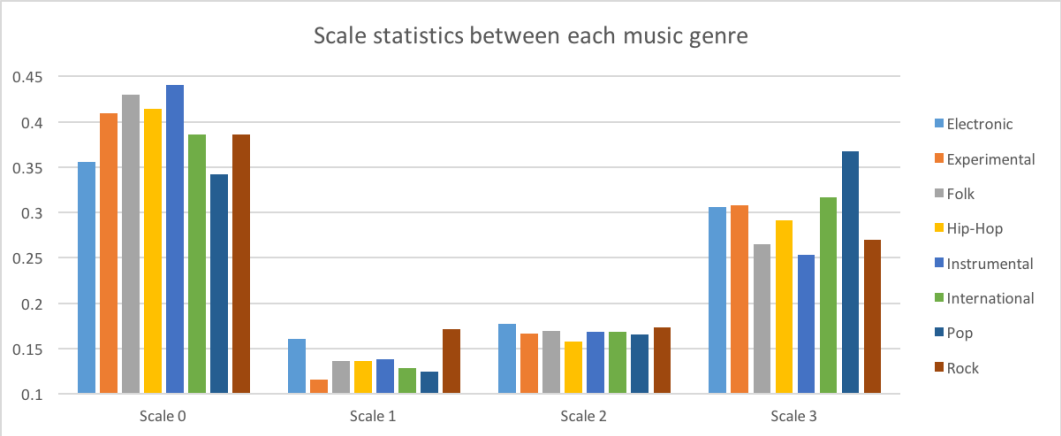


Figure 5.3: Statistics of scale selections between each music genre. The height of each bar indicates the ratio of how much times the scale is selected in the corresponding genre. Best viewed in colors.

To further understand the patterns behind such variations, we do statistics on how many times a scale has been selected for each genre, which is normalized and illustrated in figure 5.3. In general, all genres prefer to choose scale 0 and 3 since their ratio values are significantly higher than the other two. However, there are also obvious differences between genres within the same scale. For example, instrumental music tracks have more steps with scale 0 than Pop musics, while it's completely opposite for scale 3.

5.3.5 Word Level Language Modeling

Finally, we evaluate ASRNNs for the word level language modeling (WLLM) task on the WikiText-2 [87] dataset, which contains 2M training tokens with a vocabulary size of 33k. We use perplexity as the evaluation metric and the results are summarized in the Table 5.4, which shows ASRNNs can also outperform their regular counterparts. Besides, Figure 5.4 further visualizes captured scale variations for a sampled sentence. It indicates scales are usually changed at some special tokens (like semicolon and clause), which comfirms the flexibility of modeling dynamic scale pat-

terns with ASRNNs. What's more, although state-of-the-art models [87] [46] perform better, their techniques are orthogonal to our scaling mechanism so ASRNNs can still benefit from them.

Table 5.4: Perplexities for word level language modeling on WikiText-2 dataset. Italic numbers are reported by original papers.

METHODS	N	# OF WEIGHTS	MIN. SCALE	MAX. SCALE	AVG. SCALE	PPL
LSTM	1024	$\approx$ 10M	0	0	0	101.1
SLSTM	1024	$\approx$ 10M	3	3	3	97.7
ASLSTM	1024	$\approx$ 10M	0	3	1.51	**93.8**
GRU	1024	$\approx$ 7.8M	0	0	0	99.7
SGRU	1024	$\approx$ 7.8M	3	3	3	95.4
ASGRU	1024	$\approx$ 7.8M	0	3	1.38	**92.6**
ZONEOUT + VARIATIONAL LSTM [87]	-	-	-	-	-	100.9
POINTER SENTINEL LSTM [87]	-	-	-	-	-	80.8
NEURAL CACHE MODEL [46]	1024	-	-	-	-	**68.9**

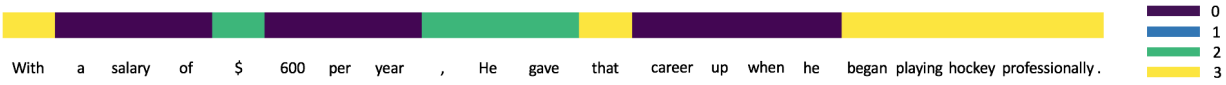


Figure 5.4: Visualized scale variations for a sampled sentence form WikiText-2 dataset.

5.4 Summarization for ASRNN

We present Adaptively Scaled Recurrent Neural Networks (ASRNNs), a simple yet useful extension that brings dynamical scale modeling abilities to existing RNN structures. At each time step, ASRNNs model the scale patterns by taking causal convolutions between wavelet kernels and input sequences such that the scale can be represented by wavelet scale coefficients. These coefficients are sampled from Gumbel-Softmax (GM) distributions which are parameterized by previous hidden states and current inputs. The differentiable nature of GM allows ASRNNs to learn to adjust scales based on different temporal contexts. Compared with other multiscale RNN models,

ASRNNs don't rely on hierarchical architectures and prefixed scale factors, making them simple and easy to train. Evaluations on various sequence modeling tasks indicate ASRNNs can outperform those non-dynamically scaled baselines by adjusting scales according to different temporal information.

CHAPTER 6: CONCLUSION AND FUTURE WORK

In this dissertation, we have explored a possible strategy to scale up the application of deep learning to a wider range of time series modeling tasks. Due to lack of well-labeled data, it is difficult to solve these problems via state-of-the-art deep models. We address the challenge by demonstrating the plausibility of extending the pure data-driven deep models with the domain-specific knowledge, which can provide priors to explicitly model the task-related temporal patterns for compensating the data shortage. We propose several designs and algorithms to mainly focus on the order patterns and frequency patterns modeling. For order patterns, we propose FTA, a hashing-based algorithm for brain disorder diagnosis. It projects the time courses of brain activities onto a set of latent patterns, and uses the index of the first coming pattern to globally represent the original time courses. The learning objectives of FTA is differentiable, thus it can be optimized as a layer of deep neural networks. We further extend FTA with RNNs for more complicated order patern-related modeling tasks such as video representation learning, resulting in the TPRNN, which employs gates to manage the ordinal relationships between latent patterns. For frequency patterns, we design SFM to explicitly model the long-term dependencies on the frequency domain. SFM decomposes its memory states into multiple frequency components and learns to control the information flows on individual component only, and the representative frequencies of each component can also be regarded as learnable weights and optimized along with other network weights. On the other hand, we develop the ASRNN for dynamically filtering out unnecessary frequency bands based on its temporal context. To do so, ASRNN convolves the input with a wavelet kernel, which provides a natural coefficient to adjust the scale of the convolution. The coefficient can be further sampled from a distribution which is conditioned by the temporal contexts.

Since most of our evaluation results have achieved better performances than baselines across various time series modeling tasks, explicitly integrating domain-specific temporal pattern modeling

with RNNs can be considered as a promising way for occasions with limited well-labeled data. We hope this dissertation could serve as a cornerstone for extending the applications of the deep learning techniques for more time series-related tasks. We also expect it can provide inspirations for other researchers to develop more hybrid solutions that combines the data-driven deep models with their own domain-specific knowledge. In the future, we plan to explore and leverage more types of temporal patterns that play a key role to characterize modeling tasks. Meanwhile, we are also interested in utilizing massive unlabeled data to boost the feature learning for sequences. For this purpose, designing novel network structure and learning objectives based on certain domain-related knowledge could be a feasible option for further investigation.

APPENDIX : EQUATIONS FOR LEARNING OPTIMAL PROJECTIONS

This section contains the equations required to derive the gradient of $\mathcal{F}$ w.r.t. $\mathbf{W}$ in learning the optimal projections (section 2.2.3.4).

Here, we use $\mathcal{L}^{ij}$ to denote the logarithmic training loss for a pair of TCs $\mathbf{X}^{(i)}$ and $\mathbf{X}^{(j)}$, i.e.,

$$\mathcal{L}^{ij} = s_{ij} \log\left(1 - h^{ij}\right) + (1 - s_{ij}) \log\left(h^{ij}\right) \tag{.1}$$

The following equations can be calculated by applying the chain rule of the derivatives on $\mathcal{F}$ of Eq. (2.10).

$$\frac{\partial \mathcal{F}}{\partial \mathbf{w}_k} = \sum_{i,j=1}^{N} \frac{\partial \mathcal{L}^{ij}}{\partial \mathbf{w}_k} + \gamma \frac{\partial \Omega}{\partial \mathbf{w}_k} + \eta \frac{\partial \mathcal{V}}{\partial \mathbf{w}_k} \tag{.2}$$

$$\frac{\partial \Omega}{\partial \mathbf{w}_k} = \sum_{k \neq k'=1}^{K} \frac{2\mathbf{w}_k^\intercal \mathbf{w}_{k'}}{\|\mathbf{w}_k\|^4 \|\mathbf{w}_{k'}\|^2} \left[\left(\mathbf{w}_k^\intercal \mathbf{w}_k\right) \mathbf{w}_{k'} - \left(\mathbf{w}_k^\intercal \mathbf{w}_{k'}\right) \mathbf{w}_k\right] \tag{.3}$$

$$\begin{aligned} \frac{\partial \mathcal{V}}{\partial \mathbf{w}_k} = \sum_{i,k=1}^{N,K} \sum_{t=1}^{T} \frac{1}{T^2} \Bigg[& 2\left(t - m_k\right)\left(-\frac{\partial m_k^{(i)}}{\partial \mathbf{w}_k}\right) p_{k,t}^{(i)} \\ & + (t - m_k)^2 \left(p_{k,t}^{(i)} \mathbf{x}_t^{(i)} - p_{k,t}^{(i)} \left(\sum_{t'=1}^{T} p_{k,t'}^{(i)} \mathbf{x}_{t'}^{(i)}\right)\right) \Bigg] \end{aligned} \tag{.4}$$

$$\frac{\partial \mathcal{L}^{ij}}{\partial \mathbf{w}_k} = \begin{cases} -\frac{1}{1-h^{ij}} \frac{\partial h^{ij}}{\partial \mathbf{w}_k} & s_i = s_j \\ \frac{1}{h^{ij}} \frac{\partial h^{ij}}{\partial \mathbf{w}_k} & otherwise \end{cases} \tag{.5}$$

$$\frac{\partial h^{ij}}{\partial \mathbf{w}_k} = \sum_{k'=1}^{K} h_{k'}^{(i)} \frac{\partial h_{k'}^{(j)}}{\partial \mathbf{w}_k} + \sum_{k'=1}^{K} h_{k'}^{(j)} \frac{\partial h_{k'}^{(i)}}{\partial \mathbf{w}_k} \tag{.6}$$

$$\frac{\partial h_k^{(i)}}{\partial \mathbf{w}_k} = h_k^{(i)}(-\frac{\partial m_k^{(i)}}{\partial \mathbf{w}_k}) - (h_k^{(i)})^2(-\frac{\partial m_k^{(i)}}{\partial \mathbf{w}_k}) \tag{.7}$$

$$\frac{\partial h_l^{(i)}}{\partial \mathbf{w}_k} = -h_l^{(i)} h_k^{(i)}(-\frac{\partial m_k^{(i)}}{\partial \mathbf{w}_k}), \textit{ when } l \neq k \tag{.8}$$

$$\frac{\partial m_k^{(i)}}{\partial \mathbf{w}_k} = \sum_{t=1}^{T} \frac{t}{T} \left(p_{k,t}^{(i)} \mathbf{x}_t^{(i)} - p_{k,t}^{(i)} \left(\sum_{t'=1}^{T} p_{k,t'}^{(i)} \mathbf{x}_{t'}^{(i)} \right) \right) \tag{.9}$$

LIST OF REFERENCES

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, et al. Tensorflow: A system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016.

[2] M. Allan and C. K. Williams. Harmonising chorales by probabilistic inference. In *NIPS*, pages 25–32, 2004.

[3] M. Arjovsky, A. Shah, and Y. Bengio. Unitary evolution recurrent neural networks. In *International Conference on Machine Learning*, pages 1120–1128, 2016.

[4] J.-J. Aucouturier and F. Pachet. Finding songs that sound the same. In *Proc. of IEEE Benelux Workshop on Model based Processing and Coding of Audio*, pages 1–8, 2002.

[5] D. Bahdanau, K. Cho, and Y. Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014.

[6] P. Bellec, C. Chu, F. Chouinard-Decorte, D. S. Margulies, and C. R. Craddock. The neuro bureau adhd-200 preprocessed repository. *bioRxiv*, page 037044, 2016.

[7] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[8] Y. Bengio, P. Simard, and P. Frasconi. Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166, 1994.

[9] Y. Bengio, N. Boulanger-Lewandowski, and R. Pascanu. Advances in optimizing recurrent networks. In *Acoustics, Speech and Signal Processing (ICASSP), 2013 IEEE International Conference on*, pages 8624–8628. IEEE, 2013.

[10] D. J. Berndt and J. Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

[11] D. M. Blei, A. Y. Ng, and M. I. Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[12] N. Boulanger-lewandowski, Y. Bengio, and P. Vincent. 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)*, pages 1159–1166, 2012.

[13] G. Buzsaki. *Rhythms of the Brain*. Oxford University Press, 2006.

[14] V. Campos, B. Jou, X. Giró-i Nieto, J. Torres, and S.-F. Chang. Skip rnn: Learning to skip state updates in recurrent neural networks. *arXiv preprint arXiv:1708.06834*, 2017.

[15] S. Chang, G.-J. Qi, C. C. Aggarwal, J. Zhou, M. Wang, and T. S. Huang. Factorized similarity learning in networks. In *Data Mining (ICDM), 2014 IEEE International Conference on*, pages 60–69. IEEE, 2014.

[16] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 76–86, 2017.

[17] G. Chéron, I. Laptev, and C. Schmid. P-cnn: Pose-based cnn features for action recognition. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3218–3226, 2015.

[18] X. Chu, W. Ouyang, H. Li, and X. Wang. Structured feature learning for pose estimation. *arXiv preprint arXiv:1603.09065*, 2016.

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

[20] J. Chung, S. Ahn, and Y. Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

[21] T. Cooijmans, N. Ballas, C. Laurent, Ç. Gülçehre, and A. Courville. Recurrent batch normalization. *arXiv preprint arXiv:1603.09025*, 2016.

[22] C. Cortes and V. Vapnik. Support-vector networks. *Machine learning*, 20(3):273–297, 1995.

[23] S. Cortese, C. Kelly, C. Chabernaud, E. Proal, A. Di Martino, M. P. Milham, and F. X. Castellanos. Toward systems neuroscience of adhd: a meta-analysis of 55 fmri studies. *American Journal of Psychiatry*, 2012.

[24] D. D. Cox and R. L. Savoy. Functional magnetic resonance imaging (fmri)"brain reading": detecting and classifying distributed patterns of fmri activity in human visual cortex. *Neuroimage*, 19(2):261–270, 2003.

[25] R. C. Craddock, G. A. James, P. E. Holtzheimer, X. P. Hu, and H. S. Mayberg. A whole brain fmri atlas generated via spatially constrained spectral clustering. *Human brain mapping*, 33 (8):1914–1928, 2012.

[26] J. Damoiseaux, S. Rombouts, F. Barkhof, P. Scheltens, C. Stam, S. M. Smith, and C. Beckmann. Consistent resting-state networks across healthy subjects. *Proceedings of the national academy of sciences*, 103(37):13848–13853, 2006.

[27] I. Danihelka, G. Wayne, B. Uria, N. Kalchbrenner, and A. Graves. Associative long short-term memory. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 1986–1994, 2016.

[28] M. Defferrard, K. Benzi, P. Vandergheynst, and X. Bresson. Fma: A dataset for music analysis. In *18th International Society for Music Information Retrieval Conference*, 2017. URL `https://arxiv.org/abs/1612.01840`.

[29] S. Dey, A. R. Rao, and M. Shah. Attributed graph distance measure for automatic detection of attention deficit hyperactive disordered subjects. *Frontiers in neural circuits*, 8, 2014.

[30] S. Dieleman and B. Schrauwen. End-to-end learning for music audio. In *Acoustics, Speech and Signal Processing (ICASSP), 2014 IEEE International Conference on*, pages 6964–6968. IEEE, 2014.

[31] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko, and T. Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *CVPR*, 2015.

[32] S. B. Eickhoff, K. E. Stephan, H. Mohlberg, C. Grefkes, G. R. Fink, K. Amunts, and K. Zilles. A new spm toolbox for combining probabilistic cytoarchitectonic maps and functional imaging data. *Neuroimage*, 25(4):1325–1335, 2005.

[33] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Nips*, volume 409, 1995.

[34] S. El Hihi and Y. Bengio. Hierarchical recurrent neural networks for long-term dependencies. In *Advances in neural information processing systems*, pages 493–499, 1996.

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

[36] A. Eloyan, J. Muschelli, M. B. Nebel, H. Liu, F. Han, T. Zhao, A. Barber, S. Joel, J. J. Pekar, S. Mostofsky, et al. Automated diagnoses of attention deficit hyperactive disorder using magnetic resonance imaging. 2012.

[37] J. A. Etzel, V. Gazzola, and C. Keysers. An introduction to anatomical roi-based fmri classification analysis. *Brain research*, 1282:114–125, 2009.

[38] C. Feichtenhofer, A. Pinz, and A. Zisserman. Convolutional two-stream network fusion for video action recognition. *arXiv preprint arXiv:1604.06573*, 2016.

[39] S. Fernández, A. Graves, and J. Schmidhuber. Sequence labelling in structured domains with hierarchical recurrent neural networks. In *IJCAI*, pages 774–779, 2007.

[40] M. D. Fox, A. Z. Snyder, J. L. Vincent, M. Corbetta, D. C. Van Essen, and M. E. Raichle. The human brain is intrinsically organized into dynamic, anticorrelated functional networks. *Proceedings of the National Academy of Sciences of the United States of America*, 102(27): 9673–9678, 2005.

[41] J. A. Frazier, S. Chiu, J. L. Breeze, N. Makris, N. Lange, D. N. Kennedy, M. R. Herbert, E. K. Bent, V. K. Koneru, M. E. Dieterich, et al. Structural brain magnetic resonance imaging of limbic and thalamic volumes in pediatric bipolar disorder. *American Journal of Psychiatry*, 2005.

[42] T.-c. Fu. A review on time series data mining. *Engineering Applications of Artificial Intelligence*, 24(1):164–181, 2011.

[43] J. S. Garofolo, L. F. Lamel, W. M. Fisher, J. G. Fiscus, and D. S. Pallett. Darpa timit acoustic-phonetic continous speech corpus cd-rom. *NASA STI/Recon technical report n*, 93, 1993.

[44] G. Gkioxari and J. Malik. Finding action tubes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 759–768, 2015.

[45] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural

networks. In *Proceedings of the thirteenth international conference on artificial intelligence and statistics*, pages 249–256, 2010.

[46] E. Grave, A. Joulin, and N. Usunier. Improving neural language models with a continuous cache. *arXiv preprint arXiv:1612.04426*, 2016.

[47] A. Graves. Adaptive computation time for recurrent neural networks. *arXiv preprint arXiv:1603.08983*, 2016.

[48] A. Graves and J. Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18(5):602–610, 2005.

[49] A. Graves and J. Schmidhuber. Offline handwriting recognition with multidimensional recurrent neural networks. In *Advances in neural information processing systems*, pages 545–552, 2009.

[50] A. Graves, N. Beringer, and J. Schmidhuber. Rapid retraining on speech data with lstm recurrent networks. *Technical Report IDSIA-09-05, IDSIA*, 2005.

[51] A. Graves, S. Fernández, F. Gomez, and J. Schmidhuber. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on Machine learning*, pages 369–376. ACM, 2006.

[52] A. Graves, A.-r. Mohamed, and G. Hinton. Speech recognition with deep recurrent neural networks. In *Acoustics, speech and signal processing (icassp), 2013 ieee international conference on*, pages 6645–6649. IEEE, 2013.

[53] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[54] A. Graves, G. Wayne, and I. Danihelka. Neural turing machines. *arXiv preprint arXiv:1410.5401*, 2014.

[55] J.-D. Haynes and G. Rees. Decoding mental states from brain activity in humans. *Nature Reviews Neuroscience*, 7(7):523–534, 2006.

[56] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385*, 2015.

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

[58] S. Hochreiter, Y. Bengio, P. Frasconi, J. Schmidhuber, et al. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies, 2001.

[59] H. Hu and G.-J. Qi. State-frequency memory recurrent neural networks. In *International Conference on Machine Learning*, pages 1568–1577, 2017.

[60] H. Hu, J. Velez-Ginorio, and G.-J. Qi. Temporal order-based first-take-all hashing for fast attention-deficit-hyperactive-disorder detection. In *Proceedings of the 22Nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 905–914, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4232-2. doi: 10.1145/2939672.2939774. URL `http://doi.acm.org/10.1145/2939672.2939774`.

[61] H. Hu, Z. Wang, J.-Y. Lee, Z. Lin, and G.-J. Qi. Temporal domain neural encoder for video representation learning. In *Computer Vision and Pattern Recognition Workshops (CVPRW), 2017 IEEE Conference on*, pages 2192–2199. IEEE, 2017.

[62] X.-S. Hua and G.-J. Qi. Online multi-label active annotation: towards large-scale content-based video search. In *Proceedings of the 16th ACM international conference on Multimedia*, pages 141–150. ACM, 2008.

[63] S. Ioffe and C. Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

[64] E. Jang, S. Gu, and B. Poole. Categorical reparameterization with gumbel-softmax. *arXiv preprint arXiv:1611.01144*, 2016.

[65] S. Ji, W. Xu, M. Yang, and K. Yu. 3d convolutional neural networks for human action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 35(1):221–231, 2013.

[66] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

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

[68] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and L. Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[69] E. J. Keogh and M. J. Pazzani. Scaling up dynamic time warping for datamining applications. In *Proceedings of the sixth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 285–289. ACM, 2000.

[70] E. J. Keogh and M. J. Pazzani. Derivative dynamic time warping. In *Sdm*, volume 1, pages 5–7. SIAM, 2001.

[71] S. Klöppel, A. Abdulkadir, C. R. Jack, N. Koutsouleris, J. Mourão-Miranda, and P. Vemuri. Diagnostic neuroimaging across diseases. *Neuroimage*, 61(2):457–463, 2012.

[72] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber. A clockwork rnn. *arXiv preprint arXiv:1402.3511*, 2014.

[73] J. Koutnik, K. Greff, F. Gomez, and J. Schmidhuber. A clockwork rnn. In *Proceedings of The 31st International Conference on Machine Learning*, pages 1863–1871, 2014.

[74] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012. URL `http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf`.

[75] J. L. Lancaster, M. G. Woldorff, L. M. Parsons, M. Liotti, C. S. Freitas, L. Rainey, P. V. Kochunov, D. Nickerson, S. A. Mikiten, and P. T. Fox. Automated talairach atlas labels for functional brain mapping. *Human brain mapping*, 10(3):120–131, 2000.

[76] H. Larochelle and I. Murray. The neural autoregressive distribution estimator. In *AISTATS*, volume 1, page 2, 2011.

[77] Q. V. Le, N. Jaitly, and G. E. Hinton. A simple way to initialize recurrent networks of rectified linear units. *arXiv preprint arXiv:1504.00941*, 2015.

[78] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[79] S. Lemm, B. Blankertz, T. Dickhaus, and K.-R. Müller. Introduction to machine learning for brain imaging. *Neuroimage*, 56(2):387–399, 2011.

[80] K. Li, G.-J. Qi, J. Ye, and K. A. Hua. Linear subspace ranking hashing for cross-modal retrieval. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, (9):1825–1838, 2017.

[81] T. Lin, B. G. Horne, P. Tino, and C. L. Giles. Learning long-term dependencies in narx recurrent neural networks. *IEEE Transactions on Neural Networks*, 7(6):1329–1338, 1996.

[82] Z. C. Lipton, J. Berkowitz, and C. Elkan. A critical review of recurrent neural networks for sequence learning. *arXiv preprint arXiv:1506.00019*, 2015.

[83] C. J. Maddison, A. Mnih, and Y. W. Teh. The concrete distribution: A continuous relaxation of discrete random variables. *arXiv preprint arXiv:1611.00712*, 2016.

[84] M. Marcus, B. Santorini, and M. A. Marcinkiewicz. Building a large annotated corpus of english: The penn treebank. 1993.

[85] J. Martens and I. Sutskever. Learning recurrent neural networks with hessian-free optimization. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1033–1040, 2011.

[86] C. McKay and I. Fujinaga. Musical genre classification: Is it worth pursuing and how can it be improved? In *ISMIR*, pages 101–106, 2006.

[87] S. Merity, C. Xiong, J. Bradbury, and R. Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

[88] T. Mikolov, M. Karafiát, L. Burget, J. Cernockỳ, and S. Khudanpur. Recurrent neural network based language model. In *Interspeech*, volume 2, page 3, 2010.

[89] M. P. Milham, D. Fair, M. Mennes, S. H. Mostofsky, et al. The adhd-200 consortium: a model to advance the translational potential of neuroimaging in clinical neuroscience. *Frontiers in systems neuroscience*, 6:62, 2012.

[90] R. Mittelman. Time-series modeling with undecimated fully convolutional neural networks. *arXiv preprint arXiv:1508.00317*, 2015.

[91] M. C. Mozer. Induction of multiscale temporal structure. In *Advances in neural information processing systems*, pages 275–282, 1992.

[92] D. Neil, M. Pfeiffer, and S.-C. Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016.

[93] D. Neil, M. Pfeiffer, and S.-C. Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016.

[94] G. A. Ojemann, J. Ojemann, and N. F. Ramsey. Relation between functional magnetic resonance imaging (fmri) and single neuron, local field potential (lfp) and electrocorticography (ecog) activity in human cortex. *Front Hum Neurosci*, 7:34, 2013.

[95] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[96] R. Pascanu, T. Mikolov, and Y. Bengio. On the difficulty of training recurrent neural networks. *ICML (3)*, 28:1310–1318, 2013.

[97] R. A. Poldrack. Region of interest analysis for fmri. *Social cognitive and affective neuroscience*, 2(1):67–70, 2007.

[98] G. E. Poliner and D. P. Ellis. A discriminative model for polyphonic piano transcription. *EURASIP Journal on Applied Signal Processing*, 2007(1):154–154, 2007.

[99] G.-J. Qi. Hierarchically gated deep networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2267–2275, 2016.

[100] G.-J. Qi, X.-S. Hua, and H.-J. Zhang. Learning semantic distance from community-tagged media collection. In *Proceedings of the 17th ACM international conference on Multimedia*, pages 243–252. ACM, 2009.

[101] G.-J. Qi, C. C. Aggarwal, and T. S. Huang. On clustering heterogeneous social media objects with outlier links. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pages 553–562. ACM, 2012.

[102] H. Qin, J. Yan, X. Li, and X. Hu. Joint training of cascaded cnn for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3456–3465, 2016.

[103] L. Rabiner and B.-H. Juang. Fundamentals of speech recognition. 1993.

[104] R. P. N. Rao. *Brain-Computer Interfacing: An Introduction*. Cambridge University Press, New York, NY, USA, 2013. ISBN 1139032801, 9781139032803.

[105] C. A. Ratanamahatana and E. Keogh. Making time-series classification more accurate using learned constraints. SIAM, 2004.

[106] T. Robinson. *Several improvements to a recurrent error propagation network phone recognition system*. 1991.

[107] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015. doi: 10.1007/s11263-015-0816-y.

[108] T. N. Sainath, R. J. Weiss, A. Senior, K. W. Wilson, and O. Vinyals. Learning the speech front-end with raw waveform cldnns. In *Sixteenth Annual Conference of the International Speech Communication Association*, 2015.

[109] H. Sak, A. Senior, and F. Beaufays. Long short-term memory based recurrent neural network architectures for large vocabulary speech recognition. *arXiv preprint arXiv:1402.1128*, 2014.

[110] N. Scaringella, G. Zoia, and D. Mlynek. Automatic genre classification of music content: a survey. *IEEE Signal Processing Magazine*, 23(2):133–141, 2006.

[111] J. Schmidhuber. Neural sequence chunkers. 1991.

[112] F. Schroff, D. Kalenichenko, and J. Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 815–823, 2015.

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

[114] S. Semeniuta, A. Severyn, and E. Barth. Recurrent dropout without memory loss. *arXiv preprint arXiv:1603.05118*, 2016.

[115] G. A. Sigurdsson, G. Varol, X. Wang, A. Farhadi, I. Laptev, and A. Gupta. Hollywood in homes: Crowdsourcing data collection for activity understanding. In *European Conference on Computer Vision*, 2016.

[116] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR*, abs/1409.1556, 2014.

[117] K. Simonyan and A. Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in Neural Information Processing Systems*, pages 568–576, 2014.

[118] K. Soomro, A. R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[119] N. Srivastava, E. Mansimov, and R. Salakhutdinov. Unsupervised learning of video representations using lstms. *CoRR, abs/1502.04681*, 2, 2015.

[120] L. Sun, K. Jia, D.-Y. Yeung, and B. E. Shi. Human action recognition using factorized spatio-temporal convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 4597–4605, 2015.

[121] I. Sutskever, G. E. Hinton, and G. W. Taylor. The recurrent temporal restricted boltzmann machine. In *Advances in Neural Information Processing Systems*, pages 1601–1608, 2009.

[122] I. Sutskever, O. Vinyals, and Q. V. Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014.

[123] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[124] J. Tang, X.-S. Hua, G.-J. Qi, and X. Wu. Typicality ranking via semi-supervised multiple-instance learning. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 297–300. ACM, 2007.

[125] J. Tang, X. Shu, G.-J. Qi, Z. Li, M. Wang, S. Yan, and R. Jain. Tri-clustered tensor completion for social-aware image tag refinement. *IEEE transactions on pattern analysis and machine intelligence*, 39(8):1662–1674, 2017.

[126] G. W. Taylor, R. Fergus, Y. LeCun, and C. Bregler. Convolutional learning of spatio-temporal features. In *European conference on computer vision*, pages 140–153. Springer, 2010.

[127] T. T. D. Team, R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Bal-

las, F. Bastien, J. Bayer, A. Belikov, et al. Theano: A python framework for fast computation of mathematical expressions. *arXiv preprint arXiv:1605.02688*, 2016.

[128] Theano Development Team. Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688, May 2016. URL `http://arxiv.org/abs/1605.02688`.

[129] T. Tieleman and G. Hinton. Divide the gradient by a running average of its recent magnitude. coursera: Neural networks for machine learning. Technical report, Technical Report. Available online: https://zh. coursera. org/learn/neuralnetworks/lecture/YQHki/rmsprop-divide-the-gradient-by-a-running-average-of-its-recent-magnitude (accessed on 21 April 2017).

[130] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3d convolutional networks. In *2015 IEEE International Conference on Computer Vision (ICCV)*, pages 4489–4497. IEEE, 2015.

[131] G. Tzanetakis and P. Cook. Musical genre classification of audio signals. *IEEE Transactions on speech and audio processing*, 10(5):293–302, 2002.

[132] N. Tzourio-Mazoyer, B. Landeau, D. Papathanassiou, F. Crivello, O. Etard, N. Delcroix, B. Mazoyer, and M. Joliot. Automated anatomical labeling of activations in spm using a macroscopic anatomical parcellation of the mni mri single-subject brain. *Neuroimage*, 15 (1):273–289, 2002.

[133] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, and K. Kavukcuoglu. Wavenet: A generative model for raw audio. *arXiv preprint arXiv:1609.03499*, 2016.

[134] J. Wang, Z. Zhao, J. Zhou, H. Wang, B. Cui, and G. Qi. Recommending flickr groups with social topic model. *Information retrieval*, 15(3-4):278–295, 2012.

[135] X. Wang, T. Zhang, G.-J. Qi, J. Tang, and J. Wang. Supervised quantization for similarity search. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2018–2026, 2016.

[136] S. Wisdom, T. Powers, J. Hershey, J. Le Roux, and L. Atlas. Full-capacity unitary recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 4880–4888, 2016.

[137] Z. Wu, X. Wang, Y.-G. Jiang, H. Ye, and X. Xue. Modeling spatial-temporal clues in a hybrid deep learning framework for video classification. In *Proceedings of the 23rd ACM international conference on Multimedia*, pages 461–470. ACM, 2015.

[138] Z. Xing, J. Pei, and E. Keogh. A brief survey on sequence classification. *ACM Sigkdd Explorations Newsletter*, 12(1):40–48, 2010.

[139] W. Yang, W. Ouyang, H. Li, and X. Wang. End-to-end learning of deformable mixture of parts and deep convolutional neural networks for human pose estimation. CVPR, 2016.

[140] J. Ye, H. Hu, G.-J. Qi, and K. A. Hua. A temporal order modeling approach to human action recognition from multimodal sensor data. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 13(2):14, 2017.

[141] J. Yue-Hei Ng, M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga, and G. Toderici. Beyond short snippets: Deep networks for video classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4694–4702, 2015.

[142] M. D. Zeiler. Adadelta: an adaptive learning rate method. *arXiv preprint arXiv:1212.5701*, 2012.

[143] S. Zhang, Y. Wu, T. Che, Z. Lin, R. Memisevic, R. R. Salakhutdinov, and Y. Bengio. Architectural complexity measures of recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1822–1830, 2016.

[144] F. Zhou and F. De la Torre. Generalized time warping for multi-modal alignment of human motion. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2012.

[145] F. Zhou and F. Torre. Canonical time warping for alignment of human behavior. In *Advances in neural information processing systems*, pages 2286–2294, 2009.

[146] J. G. Zilly, R. K. Srivastava, J. Koutník, and J. Schmidhuber. Recurrent highway networks. *arXiv preprint arXiv:1607.03474*, 2016.