NOVEL GATING MECHANISMS FOR TEMPORAL CONVOLUTIONAL NETWORKS

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF ENGINEERING AND SCIENCE
OF BILKENT UNIVERSITY
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR
THE DEGREE OF
MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

By
Fatih Aslan
September 2021

Novel Gating Mechanisms for Temporal Convolutional Networks
By Fatih Aslan
September 2021

We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

Suleyman Serdar Kozat(Advisor)

Sinan Gezici

İsmail Uyanık

Approved for the Graduate School of Engineering and Science:

Ezhan Karaşan
Director of the Graduate School

ABSTRACT

NOVEL GATING MECHANISMS FOR TEMPORAL CONVOLUTIONAL NETWORKS

Fatih Aslan
M.S. in Electrical and Electronics Engineering
Advisor: Suleyman Serdar Kozat
September 2021

We investigate the sequential modeling problem and introduce a novel gating mechanism into the temporal convolutional network architectures. In particular, we propose the Gated Temporal Convolutional Network architecture with elaborately tailored gating mechanisms. In our implementation, we alter the way in which the gradients flow and avoid the vanishing or exploding gradient and the dead ReLU problems. The proposed GTCN architecture is able to model the irregularly sampled sequences as well. In our experiments, we show that the basic GTCN architecture is superior to the generic TCN architectures in various benchmark tasks requiring the modeling of long-term dependencies and irregular sampling intervals. Moreover, we achieve the state-of-the-art results on the permuted sequential MNIST and the sequential CIFAR10 benchmarks with the basic structure.

Keywords: sequential learning, temporal convolutional networks.

ÖZET

ZAMANSAL EVRİŞİMLİ SİNİRSEL AĞLAR İÇİN ÖZGÜN BİR GEÇİT MEKANİZMASI

Fatih Aslan
Elektrik Elektronik Mühendisliği, Yüksek Lisans
Tez Danışmanı: Süleyman Serdar Kozat
Eylül 2021

Sıralı modelleme problemini araştırıyoruz ve zamansal evrişimli ağ mimarilerine yeni bir geçit mekanizması sunuyoruz. Özellikle, özenle uyarlanmış geçit mekanizmalarına sahip Geçitli Zamansal Evrişimsel Ağ mimarisini öneriyoruz. Uygulamamızda, gradyanların akış şeklini değiştiriyoruz ve kaybolan veya patlayan gradyan ve ölü ReLU problemlerinden kaçınıyoruz. Önerilen GTCN mimarisi, düzensiz olarak örneklenen dizileri de modelleyebilir. Deneylerimizde, temel GTCN mimarisinin, uzun vadeli bağımlılıkların ve düzensiz örnekleme aralıklarının modellenmesini gerektiren çeşitli kıyaslama görevlerinde genel TCN mimarilerinden üstün olduğunu gösteriyoruz. Ayrıca, temel yapıyı kullanarak sıralı MNIST ve sıralı CIFAR10 kıyaslamalarında en gelişmiş sonuçları elde ediyoruz.

Anahtar sözcükler: sıralı öğrenme, zamansal evrişimli sinirsel ağlar.

Acknowledgement

I would first like to thank my supervisor, Prof. Suleyman Serdar Kozat, whose expertise was invaluable in formulating the research questions and methodology. Your insightful feedback pushed me to sharpen my thinking and brought my work to a higher level. I'm proud of, and grateful for, my time working with him.

I would like to thank Prof. Sinan Gezici and Prof. İsmail Uyanık for being in my thesis committee and their valuable time.

I would like to express my gratitude to my mother Ayşe, my father Muzaffer and my brother Emre for their unconditional support and encouragement throughout my life. I would also like to thank my future wife Şerife for her encouragement and devotion during this whole adventure.

I would like to acknowledge that this study is supported by Turk Telekom within the framework of 5G and Beyond Joint Graduate Support Programme coordinated by Information and Communication Technologies Authority.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Preliminaries

We study the classification and regression of the sequential data using deep learning architectures. This problem, sequential modeling, is extensively studied in the machine learning [1] literature since it arises in various real-life applications including finance [2], medical [3], and cyber-security [4]. In most applications, nonlinear approaches are preferred when the linear modeling is inadequate [5]. Deep learning architectures are shown to be capable of modeling nonlinear relations in many applications, such as object detection [6], image classification [7], and anomaly detection [8]. Until recent years, recurrent architectures such as Recurrent Neural Networks (RNN) [9] and their more advanced versions such as Long-Short Term Memory (LSTM) [10] and Gated Recurrent Unit (GRU) [11] were considered as the default architectures for sequence modeling tasks by deep learning researchers and practitioners [12]. Contrarily, the recent results on sequential data processing showed that convolutional architectures can beat recurrent architectures in modeling the long-term dependencies in sequential learning tasks [1]. However, despite their success at reaching the state-of-the-art results in sequential learning tasks, the ability of convolutional architectures to model the irregularly sampled sequences is not investigated apart from the continuous

kernel convolutions [13]. The sequences in most real-life applications are sampled irregularly or have missing values [14]. For instance, in medical applications, the condition of a patient is recorded at varying intervals [15]. In this thesis, we introduce a novel architecture capable of modeling long-term dependencies and the irregularities in the sampling intervals.

In particular, we introduce novel gating mechanisms to the generic Temporal Convolutional Networks (TCN) [1] to enable the architectures to model the sampling irregularities alongside long-term dependencies. We combine the powerful aspects of the gated architectures, such as LSTM [10] and GRU [11] with the temporal convolutional architectures [1]. We show that the proposed architecture, namely Gated Temporal Convolutional Networks (GTCN), outperforms the generic TCN architectures when the sampling intervals are regular or irregular. The results on real-world and artificial benchmark datasets indicate that the GTCNs overcome the state-of-the-art architectures in various sequential modeling tasks with regular and irregular sampling intervals.

1.2 Prior Art and Comparisons

1.2.1 Sequence Modeling

Sequence modeling using deep learning architectures can be divided into recurrent architectures and convolutional architectures. Recurrent architectures model the sequences using inherent recurrent connections and process the sequential data with arbitrary length, while convolutional architectures model the sequences using stacked causal convolutions. Mainly, in recurrent architectures inherent characteristics of the sequence are summarized in a state vector and this state vector is updated as the new observations arrive. Whereas, in convolutional architectures different parts of the sequence is passed through convolution filters, i.e., for feature extraction, and the output is produced at each time step. There also exist architectures that combine the recurrent and convolutional architectures

such as Convolutional LSTM [16] and Quasi-RNN [17]. Due to the sequential state update, the gradient flows in the same direction with time in recurrent architectures. Since the gradient flow of the recurrent architectures is in the same direction with time, the recurrent architectures suffer from vanishing or exploding gradients as the sequence length increases and fall short to model the long-term dependencies [1]. Extensive research is done on recurrent architectures to mitigate the effects of the vanishing or exploding gradients and architectures that use gated activations are proposed. On the other hand, convolutional architectures do not suffer from vanishing or exploding gradients as much as recurrent architectures due to their use of convolution filters and propagating the gradient through the deeper layers of the architecture, i.e., not sequentially in time. Convolutional architectures utilize causal convolutions to uphold the causality constraint of the sequential learning task and stack dilated convolutions on top of each other to cover more observations in the past and learn long-term dependencies better [1], [18], [19].

1.2.2 Irregular Sampling Intervals

Various sequential modeling applications in real-life contain irregular sampling intervals [15] either because of the nature of the application or missing values in regular sampling. In order to model the irregularities, the sampling interval information needs to be incorporated into the architecture [20]. The Time Gated LSTM (TG-LSTM) architecture [21] incorporates the sampling interval by concatenating the interval information to the input as an additional channel. Other recurrent architectures such as [14], [22], [23], and [24] utilize ordinary differential equations to learn the sampling intervals. Recently, an architecture that utilizes continuous kernel convolutions to model the irregularly sampled sequences is proposed in [13].

Extensive studies in recurrent architectures reduce the negative effects of vanishing or exploding gradients by adding gating mechanisms [10]. However, in

practice, generic recurrent architectures are not able to model the long-term dependencies due to their inherent recurrent connections [1]. In [1], it is shown that the generic TCNs outperform most generic recurrent architectures in modeling long-term dependencies. While extensions on RNNs such as Dilated RNN [25] reduce the effects of the vanishing or exploding gradients and beat the generic TCN on certain tasks, we focus on convolutional architectures and demonstrate that our architecture outperforms those extensions as well. By using ReLU [26] activation functions, TCNs alleviate the vanishing or exploding gradients. However, ReLU activation functions die in very deep architectures which is needed when the dependency is extensive in sequential modeling, e.g., when we have long period seasonalities [27]. To remedy, we introduce an elaborately tailored gating mechanism into the generic TCN architectures to mitigate both the vanishing or exploding gradients and the dead ReLU problems. Furthermore, this new gating mechanism enables the architecture to model the irregularly sampled sequences as well. To this end, we incorporate the gating mechanism into the TCN framework and increase the performance on various tasks, beating the state-of-the-art architectures.

1.3 Contributions

Our main contributions are as follows.

1. We introduce a novel gating mechanism into the generic TCN [1] architectures to model very long-term dependencies by altering the way of gradient flow and alleviating vanishing or exploding gradients and the dead ReLU problems altogether.

2. We demonstrate that the proposed GTCN architecture is robust to missing values and irregularly sampled sequential data.

3. Through an extensive set of experiments, we compare the proposed GTCN architectures with the generic TCN architectures [1] and show that the proposed architecture is superior to the generic TCNs in well-known benchmark

datasets.

4. We achieve the state-of-the-art results in the sequential CIFAR10 [28], permuted sequential MNIST [29], and the Speech Commands [30] datasets using our basic architecture.

1.4 Organization of This Thesis

The organization of this thesis is as follows. We define our problem setting in Chapter 2. In Chapter 3, we provide the generic TCN architectures, give the motivation behind the proposed architecture, and then introduce our GTCN architecture. Chapter 4 compares the performance of the GTCN architecture in modeling the long term dependencies with the state-of-the-art architectures under the regular sampling scenario. Chapter 5 demonstrates the GTCN architecture's ability to model the irregularly sampled sequences. Chapter 6 provides further experimental analysis including the ablation studies, gradient analysis, and visualisation of the gates. We conclude the thesis with final remarks in Chapter 7.

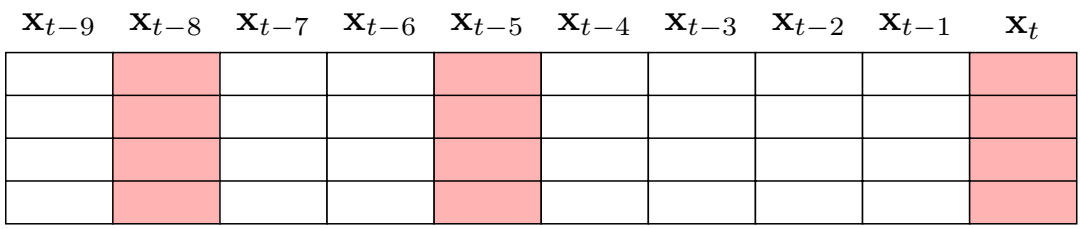
Chapter 2

Problem Description

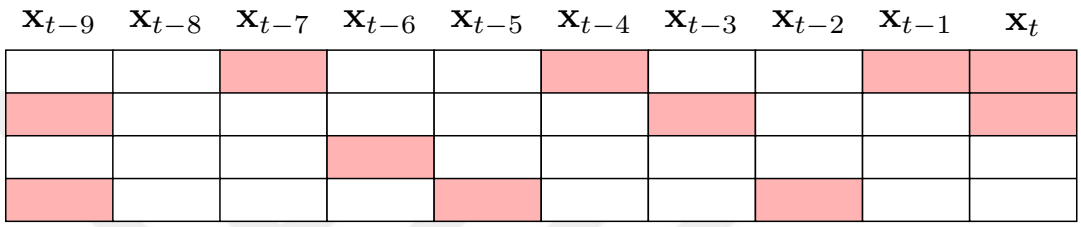
All vectors are column vectors and denoted by boldface lower case letters. Tensors are represented by boldface capital letters. For a vector $\mathbf{u}$, $\mathbf{u}^T$ is the ordinary transpose. The time index is given as subscript, e.g., $\mathbf{u}_{t_i}$ is the vector at time step t_i, and the layer index is given as superscript. $[\mathbf{U}, \mathbf{V}]$ denotes that the tensors $\mathbf{U}$ and $\mathbf{V}$ are concatenated on the temporal dimension. $[\mathbf{U}; \mathbf{V}]$ denotes that the tensors $\mathbf{U}$ and $\mathbf{V}$ are concatenated on the channel dimension. $\sigma(\mathbf{U})$ denotes that the sigmoid function is applied to the tensor $\mathbf{U}$ element-wise and $\otimes$ denotes the element-wise multiplication. $(\mathbf{U})^l$ denotes the Hadamard power and $f'(\mathbf{U})$ denotes the derivative of the function f.

2.1 Sequence Modeling

We study the sequence modeling task, where we observe an input sequence $\mathbf{X}_{t_k} = [\mathbf{x}_{t_0}, \mathbf{x}_{t_1}, \ldots, \mathbf{x}_{t_k}]$ and produce predictions $\hat{\mathbf{Y}}_{t_k} = [\hat{\mathbf{y}}_{t_0}, \hat{\mathbf{y}}_{t_1}, \ldots, \hat{\mathbf{y}}_{t_k}]$ on the output sequence $\mathbf{Y}_{t_k} = [\mathbf{y}_{t_0}, \mathbf{y}_{t_1}, \ldots, \mathbf{y}_{t_k}]$ at each step t_i where $i \leq k$ using the input sequence. While producing the outcome sequence $\hat{\mathbf{Y}}_{t_i}$, we only use the observed inputs up to step t_i, i.e., the goal is to estimate the possibly nonlinear mapping



(a) An example of irregularly sampled but fully observed sequence.



(b) An example of sporadically sampled sequence.

Figure 2.1. Examples of irregularly and sporadically sampled sequences. Red denotes the observed inputs.

from the input sequence to the output sequence:

$$\hat{\mathbf{Y}}_{t_i} = f(\mathbf{X}_{t_i}).$$

The nonlinear mapping is found by minimizing the accumulated loss between the actual output sequences $\mathbf{Y}_{t_i}$ and the predicted sequences $\hat{\mathbf{Y}}_{t_i}$:

$$\bar{L} = \frac{1}{k} \sum_{i=1}^{k} L(\mathbf{Y}_{t_i}, \hat{\mathbf{Y}}_{t_i}).$$

In some sequential modeling tasks, we do not use the predictions at each time step when calculating the loss. For instance, when doing sequential image classification, we only use the predictions at the last time step $\hat{\mathbf{Y}}_{t_k}$ and calculate the loss using $\bar{L} = L(\mathbf{Y}_{t_k}, \hat{\mathbf{Y}}_{t_k})$.

2.2 Irregularly Sampled Sequences

We also consider irregularly sampled sequences. Thus, the sampling intervals of the input sequence may not be regular and Δt_i may vary throughout the sequence. Hence, this implementation provides us flexibility when dealing with

the sequences with missing values. We can simply treat the regularly sampled sequences with missing values as irregularly sampled sequences. For instance, if some inputs from the regularly sampled sequence $\mathbf{X}_{t_k}$ are missing, we discard the missing values and rearrange the sampling intervals accordingly. In this thesis, we consider both the irregularly sampled but fully observed sequences and the sporadically sampled sequences. In the irregularly sampled but fully observed sequences, if the data is sampled at step t_i, all channels of $\mathbf{x}_{t_i}$ are observed. Whereas in the sporadically sampled sequences, some channels of $\mathbf{x}_{t_i}$ might not be observed. Examples to the irregularly sampled but fully observed sequences and the sporadically observed sequences are shown in Fig. 2.1.

Chapter 3

Novel Gating Mechanism for the TCN Architectures

We propose an elaborately tailored gating mechanism to be employed on the generic TCN architectures. Our novel algorithm allows us to incorporate the sampling interval information and keeps the gradients close to the desired values on all hidden layers. In section 3.1, we first give a brief introduction to the generic TCN architectures. After introducing the generic TCNs, we give the motivation behind the GTCN architectures in section 3.2 and describe the GTCN architectures in detail in section 3.3.

3.1 Generic TCN Architectures

The generic TCN architectures are first introduced in [1] and outperformed most generic recurrent architectures such as LSTM [10] and GRU [11] on sequence modeling tasks. TCNs utilize dilated and causal convolutions similar to the other convolutional sequence modeling architectures, such as Wavenet [18] and GLU [19].

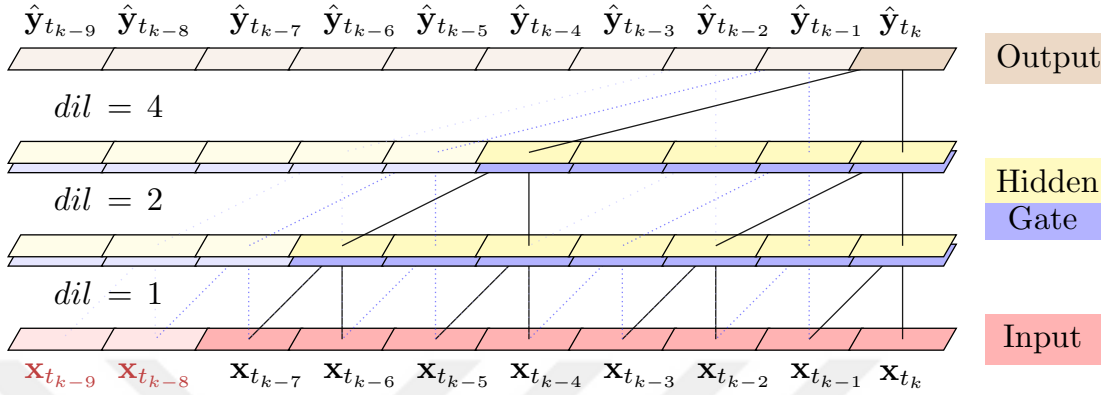


Figure 3.1. GTCN architecture with the dilation coefficient $d = 2$, the kernel size $k = 2$, the number of hidden layers $n = 2$, and the receptive field $h = 8$. *dil* denotes the dilation of the dilated convolutional filters in each layer.

Similar to the other convolutional architectures for sequence modeling, TCNs stack dilated convolutions on top of each other to increase the receptive field exponentially. The receptive field of the TCN is calculated using the number of hidden layers (n), the dilation coefficient (d), and the kernel size (k) of the architecture. The exact calculation of the receptive field (h) is given as:

$$h = k \times d^n. \tag{3.1}$$

TCNs do not alter the dilation size and keep it fixed, e.g., $d = 2$. Therefore, to increase the receptive field of the TCN, one can either increase the kernel size or the number of hidden layers. An increase in the kernel size increases the receptive field linearly, whereas an increase in the number of hidden layers increases the receptive field exponentially. In terms of computational efficiency, the number of parameters changes linearly, both with an increase in the number of hidden layers and the kernel size. Therefore, to build architectures with a reasonable receptive field while keeping the number of parameters low, one needs to increase the number of hidden layers in the architecture.

Contrary to the other convolutional architectures for sequence modeling, the generic TCN architecture does not use any gating mechanisms for nonlinear activations and uses ReLU [26] activations on hidden layers. In the supplementary material of [1], a temporal convolutional architecture with gated activations from [19] is reported to have no significant performance improvement over the

generic TCN architectures. Even though convolutional architectures decrease the negative effects of vanishing and exploding gradients by flowing the gradients in a different direction instead of temporal direction, i.e., the gradients flow just through layers, very deep architectures still suffer from vanishing or exploding gradients especially as we increase the number of stacked layers. By having ReLU activations rather than gating mechanisms, TCNs avoid the vanishing or exploding gradient problem and deeper architectures can be built using the generic TCNs. However, ReLU activation functions die in very deep architectures or under improper initializations [27].

We adopt the generic TCN [1] architecture and add gated activation functions to increase the performance. Therefore, our receptive field is also limited with equation 3.1. The summary of our proposed GTCN architecture can be seen in Fig. 3.1. The receptive field of the architecture shown in Fig. 3.1 is $h = 2 \times 2^2 = 8$. Thus, the architecture can learn up to 8 previous steps, e.g., $\hat{\mathbf{y}}_{t_k} = f([\mathbf{x}_{t_{k-8}}, \mathbf{x}_{t_{k-7}}, \ldots, \mathbf{x}_{t_k}])$. As shown in Fig. 3.1, we propagate the gate information alongside the hidden representations to the deeper layers of the architecture. TCNs use residual blocks as the building blocks. One residual block of the TCN architecture can be seen in Fig. 3.2a. The building block of our architecture, gated residual block, is provided in Fig. 3.2b.

3.2 Motivation

In this section, we provide the motivations behind the GTCN architecture. In particular, we discuss the requirements to model very long-term dependencies and the sampling irregularities while keeping the gradients reasonable. We also discuss the shortcomings of the existing architectures.

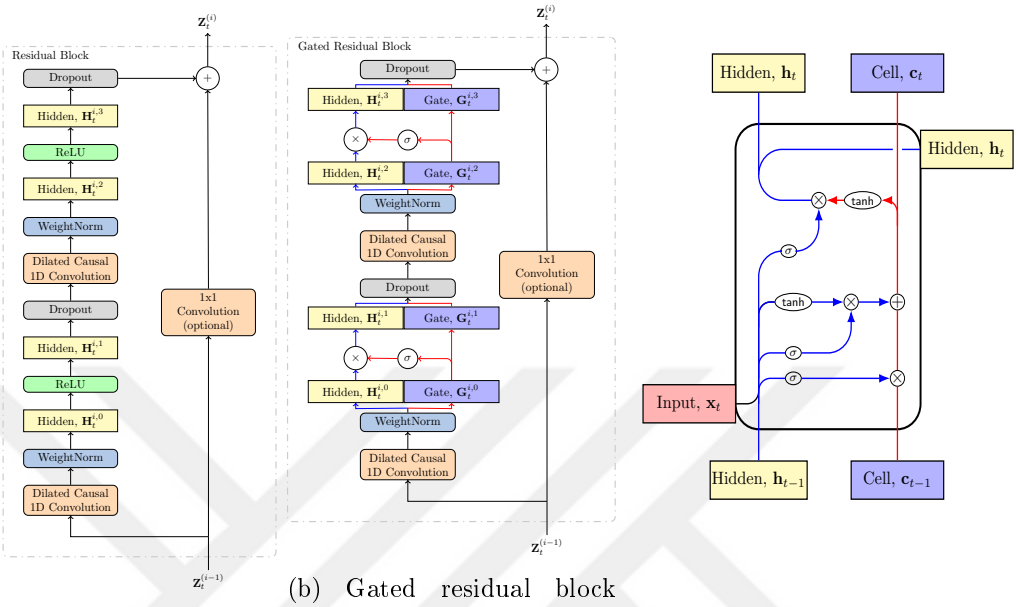
3.2.1 Modeling Very Long-term Dependencies

In order to model very long-term dependencies, the receptive field of the architecture needs to cover inputs from all previous steps that affect the output. The receptive field is increased by increasing the number of hidden layers. However, as the number of hidden layers is increased, generic TCN architectures suffer from dead ReLU, whereas gated convolutional architectures suffer from vanishing or exploding gradients. The direction of gradient flow is altered in our implementation to avoid both the dying of activation functions and the vanishing or exploding of the gradients.

3.2.2 Irregular Sampling or Missing Values

As shown in [20], one can incorporate the sampling interval information into the architecture to enhance the performance. In order to achieve this, one can directly concatenate the sampling intervals to the input tensors. In this direct implementation, sampling intervals have additive effects on the output of the architecture when the generic TCN architectures are used. Using Taylor series expansion, [21] proves that sampling intervals have a scaling effect on the actual output. In order to incorporate the scaling effects of the sampling intervals, TG-LSTM [21] are proposed. However, due to Taylor series approximation, proof in [21] only stands for sufficiently small sampling intervals. When the sampling intervals increase, sampling intervals can have both additive and scaling effects on the actual output. Our implementation models both additive and scaling effects of the sampling intervals by combining convolutional architectures with gated activations.

The proposed GTCN architecture inspires by the LSTMs and combines powerful aspects of the convolutional and gated architectures. LSTMs avoid the vanishing and exploding gradient problem by maintaining a cell state through the temporal direction. Similarly, GTCN architectures maintain the gate information through deeper layers of the architecture along with hidden representations and



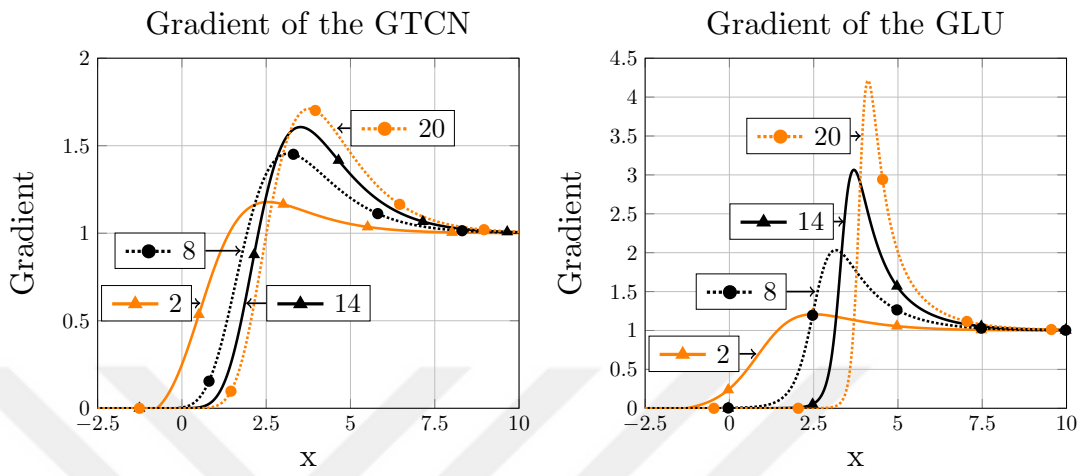
(a) Residual block [1] (b) Gated residual block (proposed architecture) (c) An LSTM cell [10]

Figure 3.2. Residual blocks used in building generic TCNs (a), and proposed GTCN architecture (b), along with a generic LSTM cell (c). Rectangles with sharp corners represent the hidden representations, gate information, hidden state, and cell state. Red arrows in 3.2b and 3.2c represent the propagation of the gate information and cell state, while blue arrows represent the propagation of the hidden representation and hidden state.

avoid the vanishing and exploding of the gradients even in very deep architectures. An LSTM cell is provided in Fig. 3.2c. The gate information of the GTCN architectures operate similarly to the cell states of the LSTM architectures. A gated residual block of the GTCN architecture is shown in Fig. 3.2b. In Fig. 3.2, the propagation of the hidden representation and hidden state is indicated with blue arrows. Whereas, the propagation of the gate information and cell state is shown as red arrows.

3.3 Gated Temporal Convolutional Network

We follow the implementation on [1] and build the architecture using the gated residual blocks. The residual block of [1] and the proposed gated residual block can be seen in Fig. 3.2a and in Fig. 3.2b, respectively. Contrary to the generic TCN architectures, our implementation contains gated activations instead of



(a) The gradient of the GTCN.

(b) The gradient of the GLU.

Figure 3.3. The gradients of the GTCN and GLU for the scalar case with alternating number of hidden layers. The number of hidden layers are provided as legend entries.

ReLU activations. We propagate both the hidden representation and the gate information to the deeper layers of the architecture. The proposed GTCN architecture differs from the existing convolutional sequence modeling architectures with the following properties:

1. The GTCNs differ from the GLU [19] and Wavenet [18] by propagating both the hidden representation and the gate information instead of propagating only the hidden representation.

2. The GTCNs differ from the TCNs [1] using gated activation functions instead of ReLU activation functions.

We alter **the way of gradient flow** and give the model **the ability to incorporate sampling interval information** as follows.

3.3.1 The Way of Gradient Flow

The gradient in our architecture flows similar to the GLU [19] and Wavenet [18], with the exception of propagating the gate information to the deeper layers. It is

proven that the GLU reduces the negative effects of the vanishing or exploding gradients compared to the Wavenet in [19]. The GTCN reduce the problem of vanishing or exploding gradients even further. As shown in [19], the gradient of the single layer gated linear unit

$$\nabla[\mathbf{X} \otimes \sigma(\mathbf{X})] = \nabla\mathbf{X} \otimes \sigma(\mathbf{X}) + \mathbf{X} \otimes \sigma'(\mathbf{X})\nabla\mathbf{X} \tag{3.2}$$

has a path $\nabla\mathbf{X} \otimes \sigma(\mathbf{X})$ without downscaling, i.e., this path avoids the vanishing or exploding gradients as more units are stacked. In equation 3.2, the one-dimensional convolution operations are omitted following the notation in [19]. The single unit of the GTCN acts similar to the GLU, with the difference of propagating the gate information to the deeper layers. Following the notation in [19], the gradient of the gated temporal convolutional network with l hidden layers

$$\begin{aligned}\nabla[\mathbf{X} \otimes (\sigma(\mathbf{X}))^l] = \nabla\mathbf{X} \otimes (\sigma(\mathbf{X}))^l \\ + l\mathbf{X} \otimes (\sigma(\mathbf{X}))^{l-1} \otimes \sigma'(\mathbf{X})\nabla\mathbf{X}\end{aligned}$$

has no path with downscaling, i.e., the gradients flow through both paths without vanishing or exploding. The gradients of the GTCN and GLU for the scalar case is provided in Fig. 3.3 with alternating number of hidden layers. As demonstrated in Fig. 3.3, the gradient of the GTCN remains stable as the number of hidden layers increase, which is needed when the sequences are long.

3.3.2 The Ability to Incorporate Sampling Interval Information

Gated activations also allow the architecture to model the scaling effects of the sampling intervals. Our implementation of the gated activations inspired by the GLUs [19] and TG-LSTMs [21]. In addition, the GTCN architectures model the additive effects of the sampling intervals by concatenating the hidden representation and the gate information before the one-dimensional convolution operation.

The one-dimensional convolution operation can be considered as a linear multiplication on the channel dimension and rewritten as

$$(\mathbf{w}_t)^T[\mathbf{h}_t; \mathbf{g}_t], \tag{3.3}$$

enabling $\mathbf{h}_t$ and $\mathbf{g}_t$ to additively affect each other. In equation 3.3, $\mathbf{w}_t$ denotes the column vector consisting of the weights of the one-dimensional convolutional filter along the channel dimension, we omit the layer superscript for notational simplicity.

Our implementation consists of the gated residual blocks, which are stacked on top of each other. The gated residual block on level i takes the input $\mathbf{Z}_t^{i-1}$ and propagates the output $\mathbf{Z}_t^i$ to the gated residual block at level $i+1$. In a gated residual block, the hidden representation and the gate information are chunked on the channel dimension to employ the gated activation and concatenated again before the dropout operation. We follow the generic TCN implementation of [1] and utilize weight normalizations, dropouts, and residual connections. The gated residual block on level i is shown in Fig. 3.2b.

The first gated residual block operates on the input sequence as follows:

$$\begin{aligned}
[\mathbf{H}_t^{1,0}; \mathbf{G}_t^{1,0}] &= Conv1D(\mathbf{X}_t), \\
[\mathbf{H}_t^{1,1}; \mathbf{G}_t^{1,1}] &= [\mathbf{H}_t^{1,0} \otimes \sigma(\mathbf{G}_t^{1,0}); \mathbf{G}_t^{1,0}], \\
[\mathbf{H}_t^{1,2}; \mathbf{G}_t^{1,2}] &= Conv1D([\mathbf{H}_t^{1,1}; \mathbf{G}_t^{1,1}]), \\
[\mathbf{H}_t^{1,3}; \mathbf{G}_t^{1,3}] &= [\mathbf{H}_t^{1,2} \otimes \sigma(\mathbf{G}_t^{1,2}); \mathbf{G}_t^{1,2}].
\end{aligned} \tag{3.4}$$

The mathematical operations in the gated residual block on level i are provided as:

$$\begin{aligned}
[\mathbf{H}_t^{i,0}; \mathbf{G}_t^{i,0}] &= Conv1D([\mathbf{H}_t^{i-1,3}; \mathbf{G}_t^{i-1,3}]), \\
[\mathbf{H}_t^{i,1}; \mathbf{G}_t^{i,1}] &= [\mathbf{H}_t^{i,0} \otimes \sigma(\mathbf{G}_t^{i,0}); \mathbf{G}_t^{i,0}], \\
[\mathbf{H}_t^{i,2}; \mathbf{G}_t^{i,2}] &= Conv1D([\mathbf{H}_t^{i,1}; \mathbf{G}_t^{i,1}]), \\
[\mathbf{H}_t^{i,3}; \mathbf{G}_t^{i,3}] &= [\mathbf{H}_t^{i,2} \otimes \sigma(\mathbf{G}_t^{i,2}); \mathbf{G}_t^{i,2}].
\end{aligned} \tag{3.5}$$

The last gated residual block operates on the hidden representation and gate

information and outputs a prediction as follows:

$$
\begin{aligned}
[\mathbf{H}_t^{l,0}; \mathbf{G}_t^{l,0}] &= Conv1D([\mathbf{H}_t^{l-1,3}; \mathbf{G}_t^{l-1,3}]), \\
[\mathbf{H}_t^{l,1}; \mathbf{G}_t^{l,1}] &= [\mathbf{H}_t^{l,0} \otimes \sigma(\mathbf{G}_t^{l,0}); \mathbf{G}_t^{l,0}], \\
[\mathbf{H}_t^{l,2}; \mathbf{G}_t^{l,2}] &= Conv1D([\mathbf{H}_t^{l,1}; \mathbf{G}_t^{l,1}]), \\
\hat{\mathbf{Y}}_t &= \mathbf{H}_t^{l,2} \otimes \sigma(\mathbf{G}_t^{l,2}).
\end{aligned} \tag{3.6}
$$

In equations 3.4, 3.5, and 3.6, we omit the weight normalizations, the dropouts, and the residual connection in the equations for notational simplicity. We denote the one-dimensional dilated causal convolution with $Conv1D(\cdot)$ and omit the weights of the convolutions as well.

Chapter 4

Experiments on Modeling Long-Term Dependencies

In this chapter, we provide experiments on different sequence modeling tasks with regular sampling and illustrate the performance of the proposed GTCN architecture by comparing it with the TCN [1] and other state-of-the-art architectures. We evaluate the ability to model the long-term dependencies of the architectures using widely used benchmarks, such as the sequential MNIST [29], permuted sequential MNIST [29], and sequential CIFAR10 [28]. For the experiments in this chapter, the hyperparameters of the architectures are provided in Table 4.1.

We compare the performance of the GTCN architecture with the TCN [1] architecture on benchmark datasets that are widely used for evaluating the ability to model the long-term dependencies of the architectures. We also report the results obtained with the state-of-the-art architectures for comparison. We set the kernel size and the number of layers as 8 to ensure that the receptive field covers the entire sequence, i.e., the receptive field is $8 \times 2^7 = 1024$. Since the sequences in the sequential MNIST datasets have the length of 784 and the sequences in the sequential CIFAR10 dataset have the length of 1024, the sequences are covered entirely.

Table 4.1. Hyperparameter selections for the experiments in Chapter 4 along with the number of parameters for each model.

Hyperparameter	Sequential MNIST		Permuted Sequential MNIST		Sequential CIFAR10	
	TCN	GTCN	TCN	GTCN	TCN	GTCN
Kernel Size	8	8	8	8	8	8
Number of Filters	50	25	50	25	50	25
Number of Layers	8	8	8	8	8	8
Gradient Clip	0.1	0.1	0.1	0.1	0.1	0.1
Learning Rate	0.002	0.002	0.002	0.01	0.001	0.01
Dropout	0.2	0.2	0.2	0.2	0.2	0.2
Number of Epochs	100	100	100	100	100	100
Scheduler Patience	5	5	5	5	5	5
Scheduler Factor	0.3	0.3	0.3	0.3	0.3	0.3
Scheduler Cooldown	10	10	5	5	10	10
Number of Parameters	302,610	302,310	302,610	302,310	303,510	303,160

The Speech Commands dataset [30] is also suitable for illustrating the architectures' ability to model very long term dependencies with $16,000$ samples for the regular sampling case, and $4,800$ samples for the irregular sampling case where the 70% of the observations are dropped randomly. We provide the results on the Speech Commands dataset in Chapter 5 alongside the irregular sampling scenario for brevity.

Table 4.2. Accuracy (%) scores on the sequential MNIST (sMNIST) and permuted sequential MNIST (pMNIST) datasets. The horizontal line separates the results copied from the existing literature from the results obtained in this paper.

Architecture (number of parameters)	Accuracy	
	sMNIST	pMNIST
LSTM (70k) [1]	87.2	85.7
GRU (70k) [1]	96.2	87.3
Dilated GRU (130k) [25]	99.2	94.6
TCN (70k) [1]	99.0	97.2
dense-IndRNN (257k) [31]	**99.48**	97.2
TrellisNet (8m) [32]	99.2	98.13
HiPPO (500k) [33]	-	98.3
UnICORNN (135k) [34]	-	98.4
CKCNN-Big (1m) [13]	99.32	98.54
TCN (300k) [1]	99.39	98.44
GTCN (300k)	99.42	**98.69**

4.1 Sequential MNIST

The sequential MNIST dataset is a well-established benchmark for testing the sequential architectures. The task is based on the classical well-known MNIST [35] dataset. In the sequential version of the MNIST dataset, grey-scale MNIST images are flattened and fed to the architectures sequentially. In order to have a fair comparison between the architectures, we set the number of parameters of both the GTCN and TCN to be around 300k. Accuracy scores on the test data are given in Table 4.2 under the column *sMNIST*. In Table 4.2, accuracy scores of the state-of-the-art algorithms from the existing literature also provided with the corresponding number of parameters for the sake of comparisons. Since the task is relatively simple, there is no significant difference between the performances of the TCN and GTCN architectures. However, both of the architectures perform near the state-of-the-art and overcome most of the existing architectures.

Table 4.3. Accuracy (%) scores on the sequential CIFAR10 dataset. The horizontal line separates the results copied from the existing literature from the results obtained in this paper.

Architecture (number of parameters)	Accuracy
	Sequential CIFAR10
Self-Attention (500k) [28]	62.2
CKCNN-Big (1m) [13]	63.74
r-LSTM (500k) [28]	72.2
TrellisNet (8m) [32]	73.42
TCN (300k) [1]	70.66
GTCN (300k)	**74.37**

4.2 Permuted Sequential MNIST

The permuted sequential MNIST dataset is a variant of the sequential MNIST dataset, where each flattened pixel array is permuted based on a fixed random permutation before fed to the architectures. Due to the random permutation, local properties in the image can not be captured, and the task requires the architectures to model the dependencies on the more distant past. Similar to the sequential MNIST, the TCN and GTCN architectures are fixed to have 300k parameters. Accuracy scores of the GTCN architecture and the state-of-the-art architectures are provided in Table 4.2, under the column *pMNIST*. In the permuted sequential MNIST dataset, GTCN outperforms the current state-of-the-art CKCNN-Big [13]. Table 4.2 shows that the performance difference between the TCN and the GTCN architectures increases as the task gets more complex. However, the range of the best accuracies in the sequential MNIST datasets is around 1% Due to the simpleness of the task, most of the algorithms model the sequences well enough. The GTCN and the TCN have the similar parameter size and computational cost with most of the architectures.

4.3 Sequential CIFAR10

A more challenging sequential modeling test is the sequential CIFAR10 [28] benchmark. Similar to the sequential MNIST dataset, images of the original CIFAR10 dataset [36] are flattened and then fed to the architectures sequentially. Since the original CIFAR10 data images are RGB-colored, sequences in the sequential CIFAR10 benchmark have 3 inputs, with a length of 1024. The CIFAR10 dataset is a much more difficult benchmark than the MNIST dataset. Thus, in the sequential CIFAR10 benchmark, all architectures suffer from reduced performance compared to the sequential MNIST benchmarks. Comparative results are shown in Table 4.3. The proposed GTCN architecture outperforms the current state-of-the-art with the improvement of 1%. The GTCN achieves this improvement with only having 300k parameters, where the state-of-the-art architecture TrellisNet has 8m parameters. Since the dataset is more complex than the sequential MNIST datasets, and due to the longer sequences, the difference between the performances of the GTCN architecture and other architectures becomes more significant. The evolution of the test accuracies for the TCN and the GTCN architectures with respect to the number of epochs is shown in Fig. 4.1 for 100 epochs.

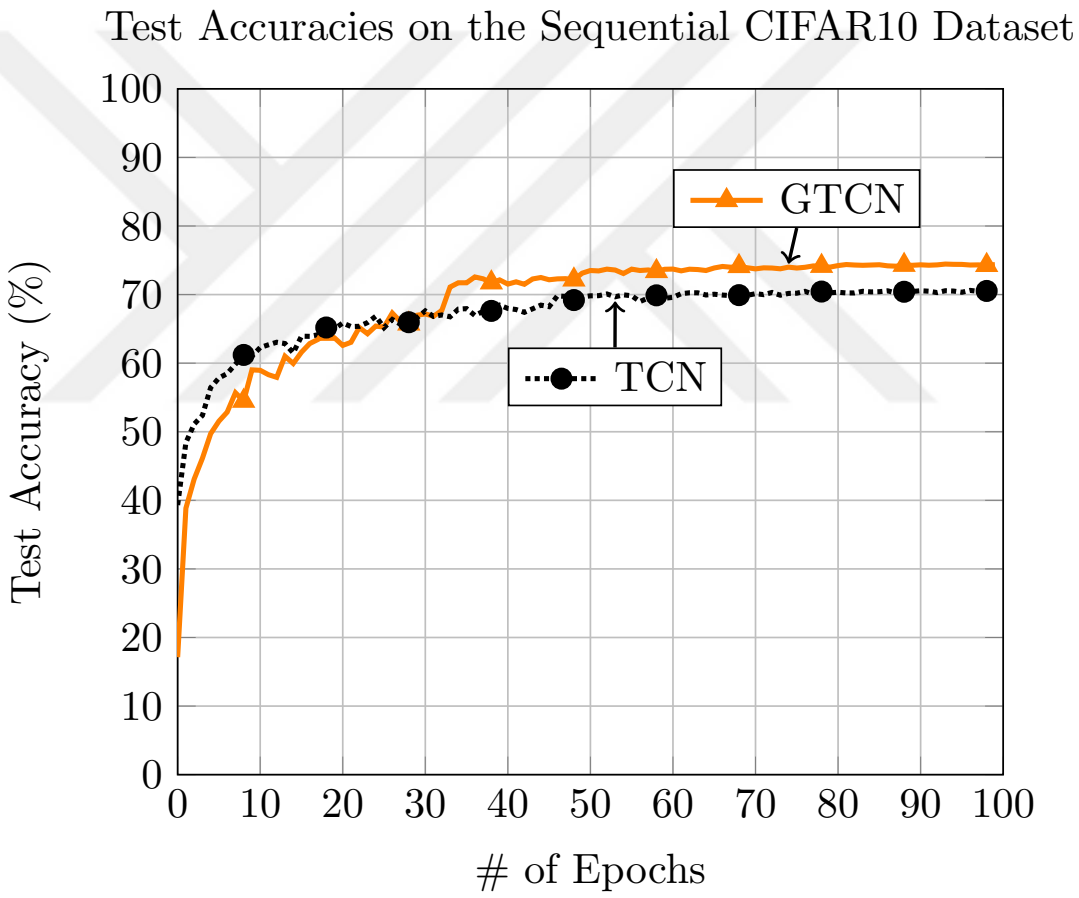


Figure 4.1. Test accuracies of the TCN and the GTCN architectures on the sequential CIFAR10 dataset with respect to the number of epochs.

Chapter 5

Experiments on Modeling Irregularly Sampled Sequences

In this chapter, we focus on the real-life problems with irregular sampling intervals, such as the Character Trajectories dataset [37] and the Speech Commands dataset [30]. We compare the TCN and the GTCN under an irregular sampling scenario. For comparisons, we provide the state-of-the-art results as well.

In order to model the irregularly sampled sequences, sampling information needs to be fed to the architecture. Sampling information is fed to the architecture using two different methods, namely the **interval appending** and **mask appending**. Methods are described as:

- **Interval Appending:** We drop the portion of the sequence randomly. Once the portion of the sequence is dropped, we append the sampling interval information as an additional channel to the input sequence. Once the sampling intervals are concatenated to the irregularly sampled input sequences as an additional channel, we feed the resulting sequences to the architectures. An example of dropping the observations and concatenating the sampling intervals is provided in Fig. 5.1. We normalize the sampling intervals using the mean and the standard deviation of the sampling intervals

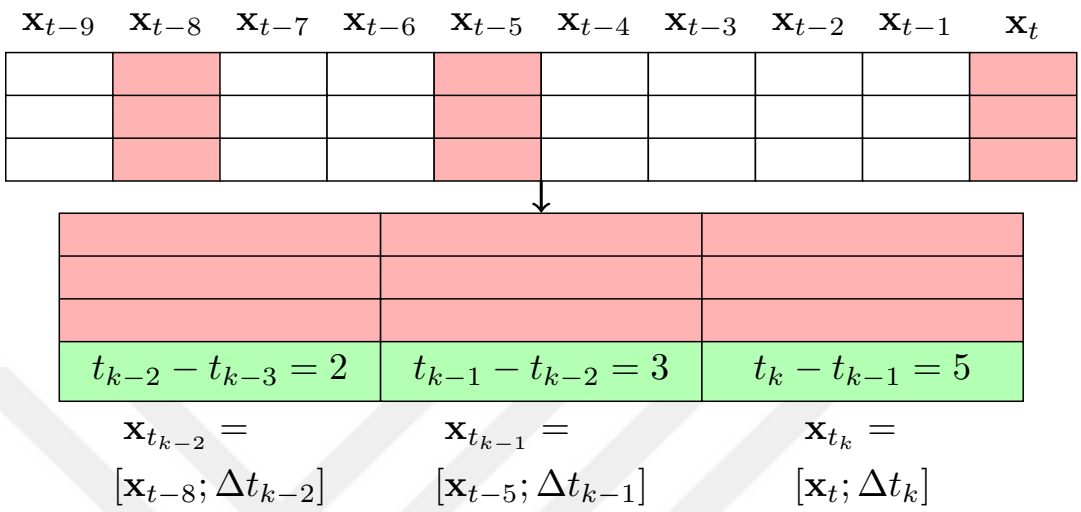


Figure 5.1. Demonstration of appending the sample intervals to the input sequences with irregular sampling. Observed inputs are denoted with red and the sampling intervals are denoted with green.

in the training set.

- **Mask Appending:** We fill the portion of the sequence using the mean of the training set. Once the portion of the sequence is filled with mean, we append a mask as an additional channel to the input sequence and feed this new sequence with an additional channel to the architectures. For the mask channel, we indicate the observed inputs with 1, and unobserved inputs with 0. Since we fill the inputs instead of dropping, length of the resulting sequence stays the same. An example of filling the observations and concatenating the mask information is provided in Fig. 5.2.

Since the sequence length is not reduced with the mask appending method, the interval appending method results in much more computational efficiency than the mask appending method. Therefore, we use the interval appending method in the experiments with the irregular sampling but fully observed sequences in this chapter. We use the mask appending method to visualise the gate information in section 6.5. In addition, we use a modified mask appending method to model the sporadically sampled sequences in section 6.3.

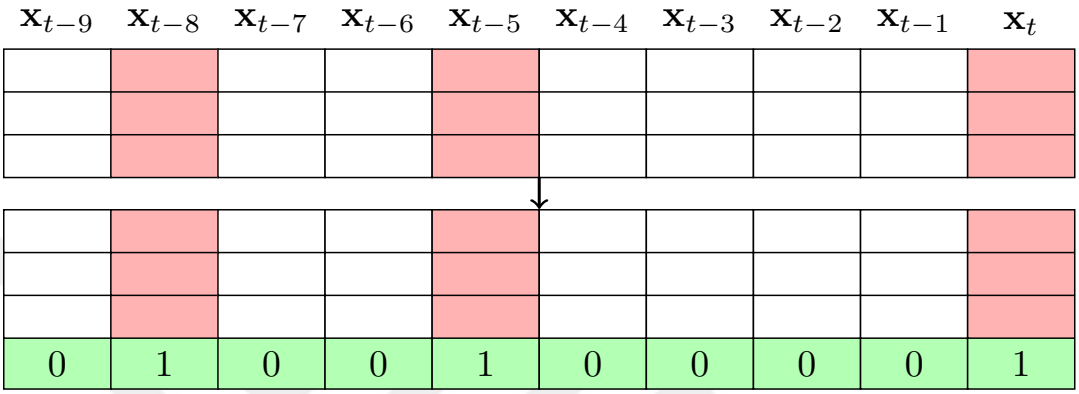


Figure 5.2. Demonstration of appending the mask information to the input sequences with irregular sampling. Observed inputs are denoted with red and the mask information is denoted with green.

Table 5.1. Accuracy (%) scores on the Character Trajectories dataset. The horizontal line separates the results copied from the existing literature from the results obtained in this paper.

Architecture	Drop Percentage			
	0%	30%	50%	70%
GRU-ODE [22]	-	92.6	86.7	89.9
GRU-Δt [23]	-	93.6	91.3	90.4
GRU-D [14]	-	94.2	90.2	91.9
ODE-RNN [38]	-	95.4	96.0	95.3
NCDE [23]	-	98.7	**98.8**	**98.6**
CKCNN [13]	**99.53**	**98.83**	98.6	98.14
TCN [1]	98.60	98.14	97.90	94.17
GTCN	98.37	98.14	96.97	94.17

5.1 Character Trajectories

We assess the ability to work with irregularly sampled sequences of the GTCN architecture using the Character Trajectories dataset from the UEA time series classification archive [37]. The dataset has 2858 sequences with 3 channels, where each of the sequences has a length of 182. We follow the methodology of [13] and [23] to compose the irregularly sampled sequences by dropping 30%, 50%, and 70% of each sequence randomly. The results are provided in Table 5.1, along with the state-of-the-art results. Since the task is relatively easy and sequences have lengths less than 200, the deepest GTCN and TCN architectures use 6 layers of residual blocks. Therefore, the TCNs do not suffer from the dead ReLU and perform better than the GTCNs. Generally, the TCN and the GTCN perform at par with the state-of-the-art algorithms when the sequences are not very long. For the experiments with the Character Trajectories dataset, the hyperparameters of the architectures are provided in Table 5.2.

Table 5.2. Hyperparameter selections for the experiments with the Character Trajectories dataset along with the number of parameters for each model.

Hyperparameter	Drop Rate 0%		Drop Rate 30%		Drop Rate 50%		Drop Rate 70%	
	TCN	GTCN	TCN	GTCN	TCN	GTCN	TCN	GTCN
Kernel Size	6	6	8	8	6	6	8	8
Number of Filters	40	20	40	20	40	20	40	20
Number of Layers	6	6	5	5	5	5	4	4
Gradient Clip	0.1	0.1	0.1	0.1	0.1	0.1	0.1	0.1
Learning Rate	0.001	0.001	0.001	0.001	0.001	0.001	0.001	0.001
Dropout	0.4	0.4	0.4	0.4	0.4	0.4	0.4	0.4
Number of Epochs	50	50	50	50	50	50	50	50
Scheduler Patience	3	3	3	3	3	3	3	3
Scheduler Factor	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Scheduler Cooldown	5	5	5	5	5	5	5	5
Number of Parameters	108,260	107,780	118,300	117,800	89,180	88,680	92,540	92,040

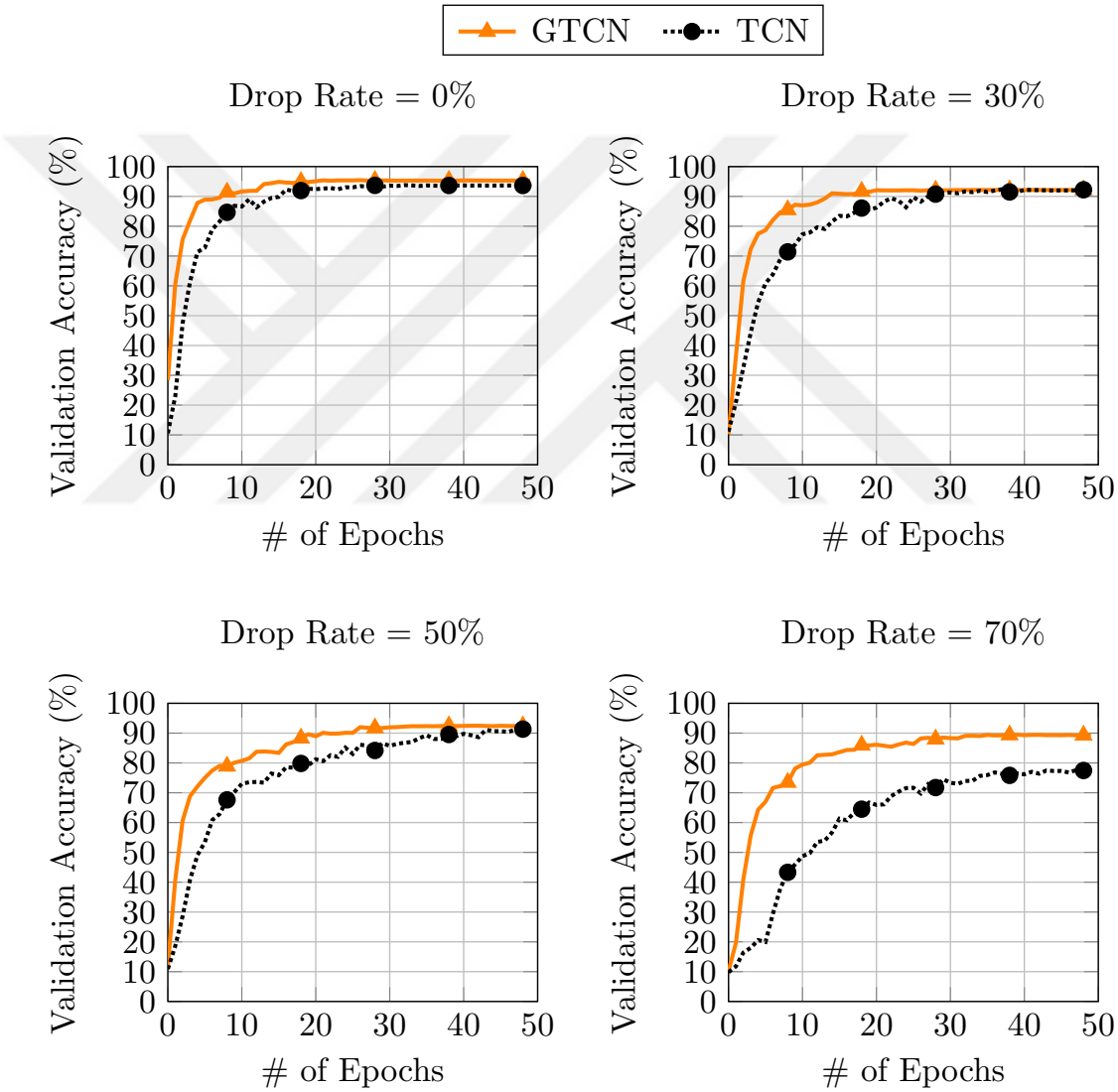


Figure 5.3. Validation accuracies of the GTCN and TCN architectures on the speech commands dataset. The percentage of dropped observations are shown in the title of each plot.

5.2 Speech Commands

The Speech Commands dataset [30] is a relatively challenging dataset compared to the Character Trajectories dataset. There are voice recordings of various short commands in the Speech Commands dataset with the length of 1 second each, sampled at 16 kHz. We follow the work of [13] and choose ten different commands to predict. Selected ten commands for this experiment are *"yes," "no," "up," "down," "left," "right," "on," "off," "stop,"* and *"go,"* same as [13]. Contrary to [23], we do not employ any preprocessing on the speech data other than normalization. This dataset is referred to as the Raw Speech Commands dataset in [13]. We follow the same methodology used in the Character Trajectories dataset to sample the sequences irregularly. Each sequence has 16000 points initially. Due to having longer sequences, the Speech Commands dataset is an excellent choice to assess the architectures based on the ability to model the long-term dependencies alongside the irregularities in the sampling intervals. The TCN and GTCN algorithms are set to have 12 layers and the kernel size of 8 when the sequences are observed entirely. For the experiments with the Speech Commands dataset, the hyperparameters of the architectures are provided in Table 5.3.

The results on the test dataset are provided in Table 5.4. In the regular sampling setting, where we do not drop observations from the sequences, the GTCN outperforms the state-of-the-art results on the preprocessed Speech Commands dataset while the TCN reaches a comparable accuracy. Both the GTCN and TCN outperform the state-of-the-art results on the raw Speech Commands dataset. The state-of-the-art result on the preprocessed Speech Commands dataset is 95.27%, obtained in [13] by the CKCNN architecture. TCN outperforms GTCN when the percentage of dropped observations is 30%. The most significant difference between the accuracies of the GTCN and TCN architectures occurs when the 70% of the observations are dropped with the difference being around 11%. Nevertheless, both the TCN and the GTCN outperform the CKCNN architecture under the regular and irregular sampling scenarios. Despite being outperformed by the GTCN, the TCN effectively models the long-term dependencies in the regularly sampled Speech Commands dataset. Improvement of the GTCN over the

Table 5.3. Hyperparameter selections for the experiments with the Speech Commands dataset along with the number of parameters for each model.

Hyperparameter	Original Dataset		Drop Rate 30%		Drop Rate 50%		Drop Rate 70%	
	TCN	GTCN	TCN	GTCN	TCN	GTCN	TCN	GTCN
Kernel Size	8	8	6	6	8	8	5	5
Number of Filters	50	25	50	25	50	25	50	25
Number of Layers	12	12	12	12	11	11	11	11
Gradient Clip	0.5	0.5	0.5	0.5	0.5	0.5	0.5	0.5
Learning Rate	0.0001	0.001	0.0001	0.001	0.0001	0.001	0.0001	0.001
Dropout	0.2	0.2	0.2	0.2	0.2	0.2	0.2	0.2
Number of Epochs	50	50	50	50	50	50	50	50
Scheduler Patience	3	3	3	3	3	3	3	3
Scheduler Factor	0.3	0.3	0.3	0.3	0.3	0.3	0.3	0.3
Scheduler Cooldown	0	0	0	0	0	0	0	0
Number of Parameters	463,410	463,110	348,660	348,335	423,660	423,335	265,860	265,535

Table 5.4. Accuracy (%) scores on the Speech Commands dataset. The horizontal line separates the results copied from the existing literature from the results obtained in this paper.

Architecture	Drop Percentage			
	0%	30%	50%	70%
CKCNN [13]	71.66	63.46	60.55	57.50
TCN [1]	94.15	**92.68**	91.35	77.55
GTCN	**95.54**	92.61	**92.74**	**89.29**

TCN is only 1% when the sampling is regular. However, the GTCN offers more balance in terms of modeling long term dependencies and sampling irregularities together. Interestingly, the poorest performance of the GTCN is on the dataset with 30% drop rate. The gradients are investigated thoroughly in section 6.4.

Chapter 6

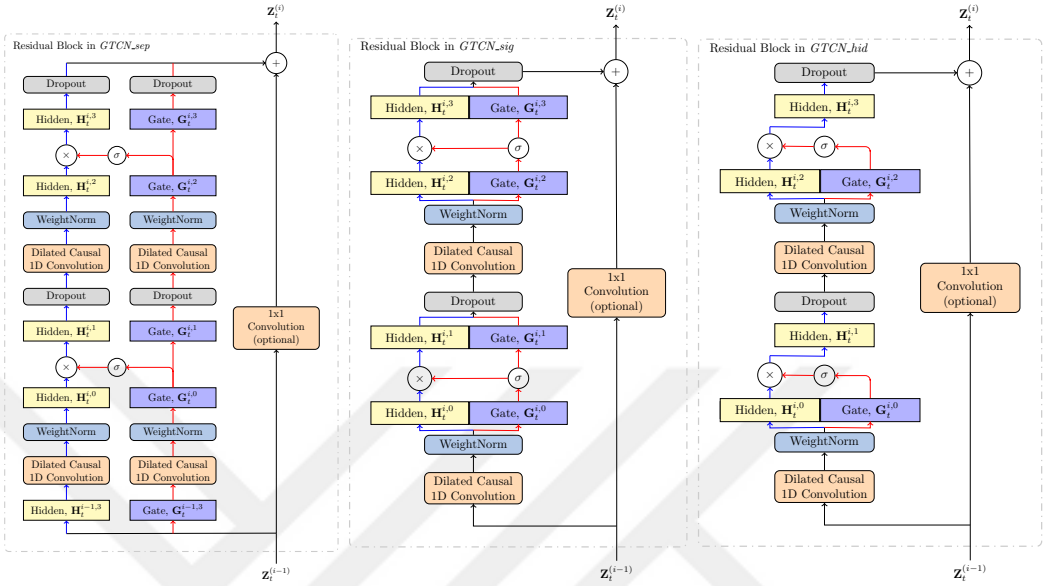
Other Experimental Analysis

In this chapter, we provide experimental analysis and further investigate the proposed GTCN architecture. First, we perform ablation studies using the Sequential CIFAR10 [28] and Speech Commands [30] datasets. Then we compare the sampling information appending methods using the Sequential CIFAR10 dataset and demonstrate the ability of the GTCN architecture to model sporadically sampled sequences. Next we provide further insights on the gradients and the gate information by performing gradient analysis and visualising the gate information. We perform an experiment by reducing the number of parameters on both the TCN and the GTCN architectures to compare the performance of the architectures with fewer parameters. Finally, we mention the hyperparameter selection processes and the limitations of the GTCN architecture.

6.1 Ablation Study

Architectures to be assessed in the ablation study are:

- $GTCN\_sep$, where the gate information and the hidden representation are not concatenated before the one-dimensional convolution operation.



(a) *GTCN_sep* (b) *GTCN_sig* (c) *GTCN_hid*

Figure 6.1. Residual blocks used in ablation studies. Rectangles with sharp corners represent the hidden representation and gate information. Red arrows represent the propagation of the gate information, while blue arrows represent the propagation of the hidden representation.

- *GTCN_sig*, where the gate information is propagated after the activation function is employed.
- *GTCN_hid*, where the gate information is not propagated to the deeper layers of the architecture. This case reduces to the TCN with gated activations algorithm in [1].

The designs of the Residual Blocks used in the ablation study are provided in Figure 6.1, where the mathematical operations on level i can be summarized

as follows.

$$
\begin{aligned}
&GTCN\_sep\text{:}\\
&\qquad \mathbf{H}_t^{i,0} = Conv1D(\mathbf{H}_t^{i-1,3}),\\
&\qquad \mathbf{G}_t^{i,0} = Conv1D(\mathbf{G}_t^{i-1,3}),\\
&\qquad \mathbf{H}_t^{i,1} = \mathbf{H}_t^{i,0} \otimes \sigma(\mathbf{G}_t^{i,0}),\\
&\qquad \mathbf{G}_t^{i,1} = \mathbf{G}_t^{i,0},\\
&\qquad \mathbf{H}_t^{i,2} = Conv1D(\mathbf{H}_t^{i,1}),\\
&\qquad \mathbf{G}_t^{i,2} = Conv1D(\mathbf{G}_t^{i,1}),\\
&\qquad \mathbf{H}_t^{i,3} = \mathbf{H}_t^{i,2} \otimes \sigma(\mathbf{G}_t^{i,2}),\\
&\qquad \mathbf{G}_t^{i,3} = \mathbf{G}_t^{i,2}.
\end{aligned}
$$

$$
\begin{aligned}
&GTCN\_sig\text{:}\\
&[\mathbf{H}_t^{i,0}; \mathbf{G}_t^{i,0}] = Conv1D([\mathbf{H}_t^{i-1,3}; \mathbf{G}_t^{i-1,3}]),\\
&[\mathbf{H}_t^{i,1}; \mathbf{G}_t^{i,1}] = [\mathbf{H}_t^{i,0} \otimes \sigma(\mathbf{G}_t^{i,0}); \sigma(\mathbf{G}_t^{i,0})],\\
&[\mathbf{H}_t^{i,2}; \mathbf{G}_t^{i,2}] = Conv1D([\mathbf{H}_t^{i,1}; \mathbf{G}_t^{i,1}]),\\
&[\mathbf{H}_t^{i,3}; \mathbf{G}_t^{i,3}] = [\mathbf{H}_t^{i,2} \otimes \sigma(\mathbf{G}_t^{i,2}); \sigma(\mathbf{G}_t^{i,2})].
\end{aligned}
$$

$$
\begin{aligned}
&GTCN\_hid\text{:}\\
&[\mathbf{H}_t^{i,0}; \mathbf{G}_t^{i,0}] = Conv1D([\mathbf{H}_t^{i-1,3}),\\
&[\mathbf{H}_t^{i,1}] = [\mathbf{H}_t^{i,0} \otimes \sigma(\mathbf{G}_t^{i,0})],\\
&[\mathbf{H}_t^{i,2}; \mathbf{G}_t^{i,2}] = Conv1D([\mathbf{H}_t^{i,1}]),\\
&[\mathbf{H}_t^{i,3}] = [\mathbf{H}_t^{i,2} \otimes \sigma(\mathbf{G}_t^{i,2})].
\end{aligned}
$$

In this section, we compare the proposed GTCN architecture with the *GTCN_sep*, *GTCN_sig*, and *GTCN_hid* architectures using the Sequential CIFAR10 [28] and Speech Commands [30] datasets under the regular and irregular sampling scenarios.

Table 6.1. Accuracy (%) scores on the sequential CIFAR10 dataset with random pixel removals.

Architecture	Number of Removed Pixels			
	0	256	512	768
GTCN_sep	54.89	55.05	51.56	50.04
GTCN_sig	68.99	62.21	60.21	54.63
GTCN_hid	68.87	64.95	**62.64**	56.80
GTCN	**73.37**	**66.04**	61.80	**56.81**

6.1.1 Sequential CIFAR10 Dataset with Missing Values

We employ the three architectures and the proposed GTCN architecture on the sequential CIFAR10 dataset with missing values. We randomly remove some pixels from the flattened CIFAR10 images and append the sampling interval information as an additional channel, similar to experiments in Chapter 5. Similar to the irregular sampling scenario, we normalize the sampling intervals using the mean and the standard deviation of the sampling intervals in the training set. We generate three different datasets by removing 256, 512, and 768 pixels randomly from each sequence. We provide the results on the three datasets with missing pixels alongside the original sequential CIFAR10 dataset and compare the proposed GTCN architecture with the *GTCN_sep*, *GTCN_sig*, and *GTCN_hid* architectures in Table 6.1. The hyperparameters of the architectures are provided in Table 6.2.

The evolution of the test accuracies of the architectures is provided in Fig. 6.2. In all of the scenarios, the *GTCN_sep* architecture has the poorest performance due to its lack of ability to model the additive effects of the sampling interval information. When there are no missing values in the dataset, the *GTCN_hid* architecture performs similar to the *GTCN_sig* architecture, and the GTCN architecture outperforms all architectures, as expected due to the longer sequences in the sequential CIFAR10 dataset. In order to learn the longer dependencies, architectures are set to be deeper in this scenario, which causes the vanishing or exploding of the gradients in the *GTCN_hid* and *GTCN_sig* architectures.

Table 6.2. Hyperparameter selections for the experiments in Section 6.1.1 along with the number of parameters for each model.

Hyperparameter	Number of Removed Pixels			
	0	256	512	768
Kernel Size	8	6	8	8
Number of Channels	25	25	25	25
Number of Layers	8	8	7	6
Gradient Clip	0.1	0.1	0.1	0.1
Learning Rate	0.01	0.01	0.01	0.01
Dropout	0.2	0.2	0.2	0.2
Number of Epochs	50	50	50	50
Scheduler Patience	3	3	3	3
Scheduler Factor	0.3	0.3	0.3	0.3
Scheduler Cooldown	0	0	0	0
Architecture	Number of Parameters			
GTCN	303,160	228,185	263,385	223,185
GTCN_hid	153,160	115,685	133,385	113,185
GTCN_sig	303,160	228,185	263,385	223,185
GTCN_sep	153,160	115,685	133,385	113,185

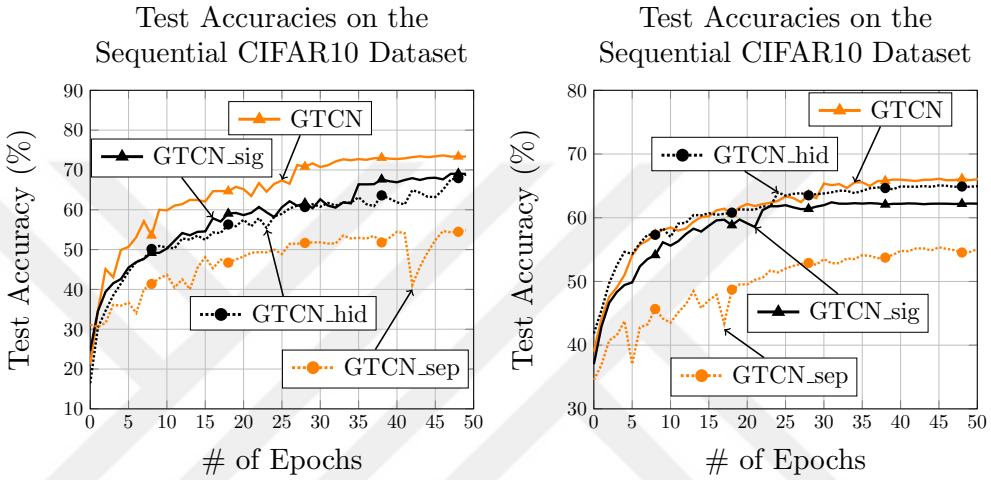
As the number of dropped pixels increases, the difference between the GTCN and *GTCN_hid* architectures becomes insignificant due to the decrease in the sequence lengths and the number of hidden layers of the architectures. We adjusted the number of hidden layers to be in the interval $[6, 8]$ according to the sequence lengths.

6.1.2 Irregularly Sampled Speech Commands Dataset

We employ the three architectures and the proposed GTCN architecture on the irregularly sampled speech commands dataset. For the irregular sampling scenario, we use the same dataset as in Chapter 5. Similar to the irregular sampling

scenario, we use the interval appending method and normalize the sampling intervals using the mean and the standard deviation of the sampling intervals in the training set. We provide the results on the three datasets with irregular sampling intervals alongside the regularly sampled speech commands dataset and compare the proposed GTCN architecture with the *GTCN_sep*, *GTCN_sig*, and *GTCN_hid* architectures in Table 6.3. The hyperparameters of the architectures are provided in Table 6.5.

The evolution of the validation accuracies of the architectures is provided in Fig. 6.3. *GTCN_hid* performs the best under the regular sampling scenario, where GTCN outperforms other architectures when the sampling intervals are irregular. When the drop rate is higher than 30%, *GTCN_sig* can not model the sequence and performs as good as random chance. As shown in Fig. 6.3 validation score of *GTCN_sig* does not increase through epochs when the drop rate is 50% or 70%. When the sampling intervals are regular GTCN, *GTCN_hid*, and *GTCN_sig* have similar results and validation trajectories.



(a) Sequential CIFAR10 dataset with no missing values.

(b) Sequential CIFAR10 dataset, 256 pixels are dropped.

(c) Sequential CIFAR10 dataset, 512 pixels are dropped.

(d) Sequential CIFAR10 dataset, 768 pixels are dropped.

Figure 6.2. Test accuracies of the GTCN, *GTCN_hid*, *GTCN_sig*, and *GTCN_sep* architectures on the sequential CIFAR10 dataset with missing values. The number of dropped pixels is shown below each plot.

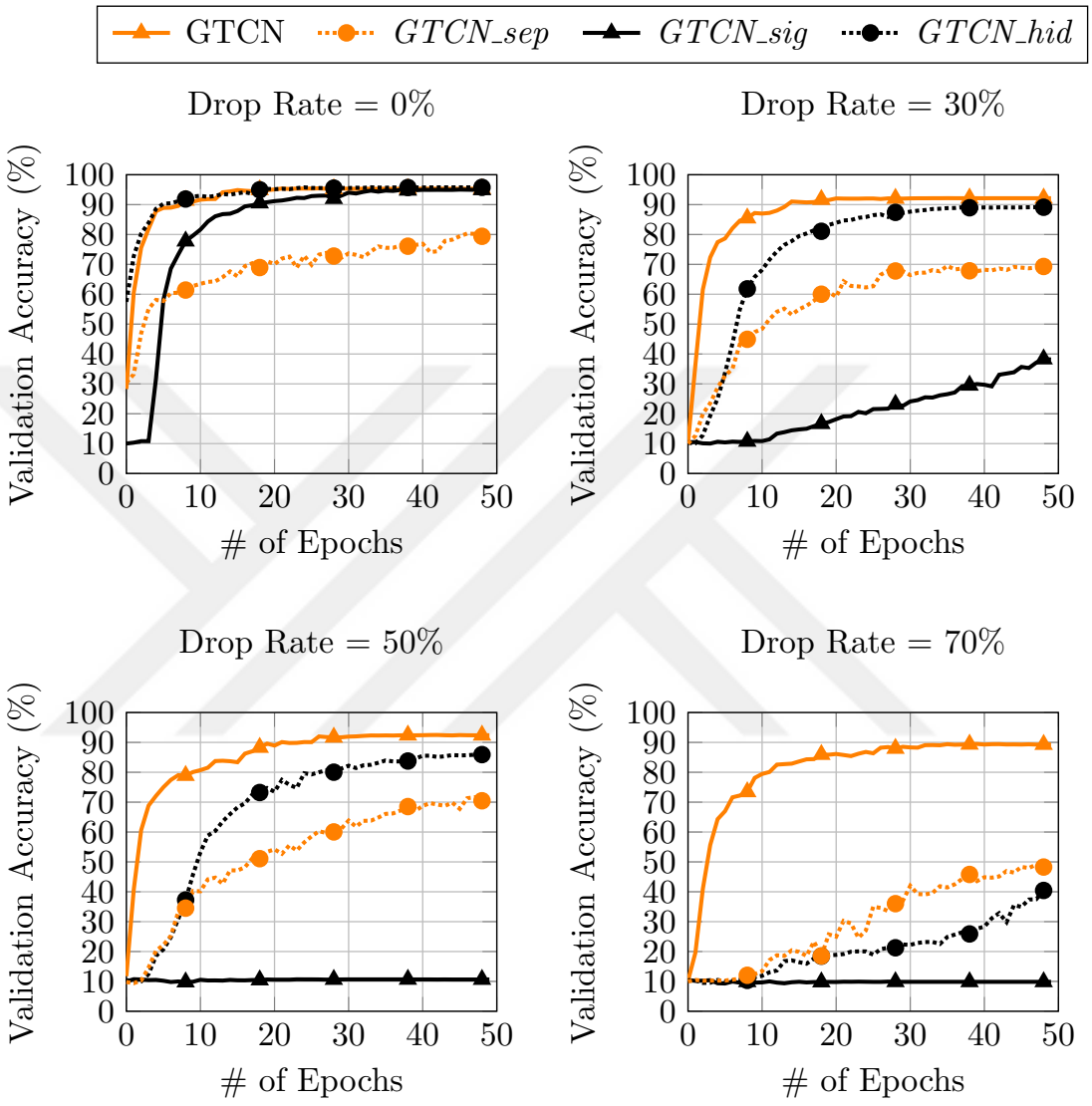


Figure 6.3. Validation accuracies of the GTCN, *GTCN_hid*, *GTCN_sig*, and *GTCN_sep* architectures on the speech commands dataset. The percentage of dropped observations are shown in the title of each plot.

Table 6.3. Accuracy (%) scores on the speech commands dataset.

Architecture	Drop Percentage			
	0%	30%	50%	70%
GTCN_sep	79.86	68.80	71.20	48.12
GTCN_sig	94.99	38.17	10.65	9.85
GTCN_hid	**95.79**	89.12	85.92	40.48
GTCN	95.54	**92.61**	**92.74**	**89.29**

6.2 Comparison between the Methods of Appending the Sampling Information

We demonstrate that mask appending is also a valid method to model the sampling irregularities in the sequential data using the sequential CIFAR10 dataset with the GTCN architecture. We undersample the sequences similar to the experiments done in Section 6.1.1, with the mask appending method instead of interval appending method. Thus, we don't remove the pixels but fill them with the mean value, i.e., 127.5 for the CIFAR10 images.

The results of the GTCN architecture using the mask appending method are provided in Table 6.4 where the validation trajectories are provided in Fig. 6.4 with respect to the number of filled pixels, in comparison with the interval appending method. In the experiments, hyperparameters of the GTCN architecture are set to be the same as the hyperparameters in Table 4.1, under the column of Sequential CIFAR10, GTCN.

As shown in Fig. 6.4 and Table 6.4, the mask appending method outperforms the interval appending method when the sequences are irregularly sampled but fully observed with the accuracy difference varying between 3% and 7%. However, since the sequence lengths are not reduced in the mask appending method, using the interval appending method reduces the computational time by almost 75% when the number of removed or filled pixels are 768.

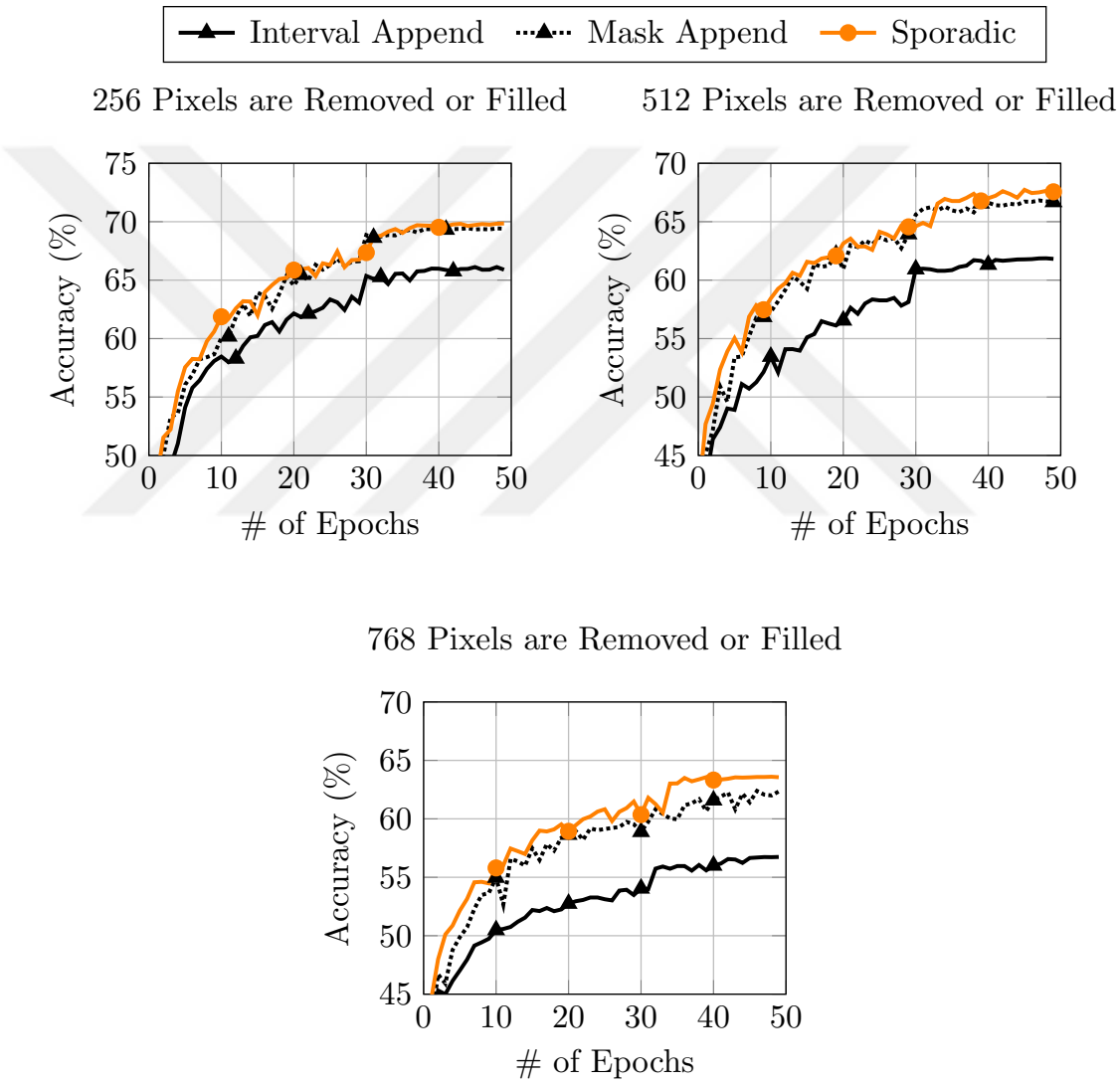


Figure 6.4. Validation accuracies of the GTCN architecture using interval append, mask append and sporadic sequences. The number of removed or filled pixels are shown in the title of each plot.

6.3 Sporadically Sampled Sequences

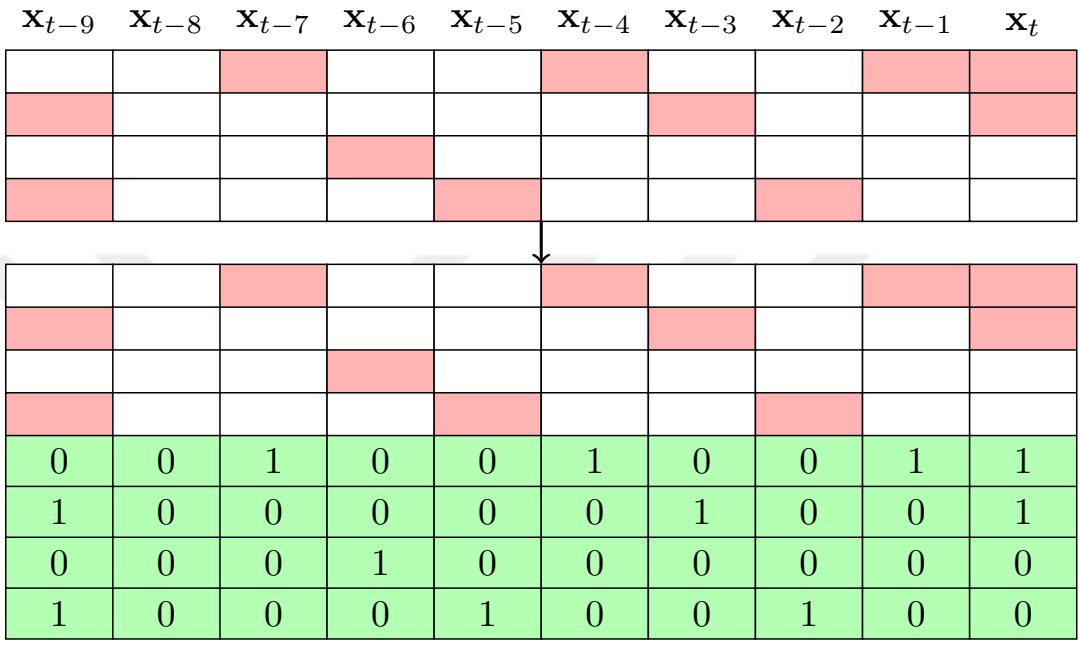


Figure 6.5. Demonstration of appending the mask information to the input sequences with sporadical sampling. Observed inputs are denoted with red and the mask information is denoted with green.

To demonstrate that the GTCN architecture is also capable of modelling the sporadically sampled sequences, we perform experiments using the sporadically sampled sequential CIFAR10 [28] dataset. In this section, we use a modified version of the mask appending method to feed the sampling information to the GTCN architecture. In this modified method, k additional channels are appended to the original input sequence, with k being the number of original input channels, i.e., sporadically sampled sequences in the sequential CIFAR10 dataset has 6 input channels, RGB and 3 channel sampling information. The modified mask appending method for sporadically sampled sequences is shown in Fig. 6.5.

The results of the GTCN architecture for the sporadically sampled scenario are given in Table 6.4 where the validation trajectories are provided in Fig. 6.4 with respect to the number of filled pixels, in comparison with the scenario of irregularly sampled but fully observed sequences. In the scenario of sporadically observed sequences, hyperparameters of the GTCN architecture are set to be same

Table 6.4. Accuracy (%) scores on the sequential CIFAR10 dataset with random pixel removals (interval append) or fillings (mask append and sporadic).

Scenario	Number of Filled Pixels		
	256	512	768
Irregular (interval append)	66.04	61.80	56.81
Irregular (mask append)	**69.85**	**67.56**	**63.56**
Sporadic	69.39	66.70	62.13

as the hyperparameters in Table 4.1, under the column of Sequential CIFAR10, GTCN.

As shown in Fig. 6.4 and Table 6.4, the GTCN architecture is capable of modeling the sporadically sampled sequences as well. When the sequence is sporadically sampled, the architecture suffers from the 1% accuracy loss with respect to irregularly sampled but fully observed sequences. Nevertheless, using the mask appending method with the sporadically sampled sequences results in higher accuracy than the interval appending method. As a result, the performance gain is achieved with the mask appending method by reducing the computational efficiency, even with the sporadically sampled sequences.

6.4 Analysis of the Gradients

In this section we provide an analysis of the gradients of the architectures used in Section 6.1 alongside the TCN architecture. The analysis is performed using the absolute values of all gradients. The maximum of the absolute gradients, the mean of the absolute gradients, and the percentage of the zero gradients are plotted with respect to the number of epochs. The maximum of the absolute gradients and the mean of the absolute gradients provides insights on exploding or vanishing gradients. The percentage of the zero gradients are investigated to determine if the TCN architecture suffer from dead ReLU [27]. The Speech Commands datasets [30] with regular and irregular sampling intervals are used

Table 6.5. Hyperparameter selections for the experiments in Sections 6.1.2 and 6.4 along with the number of parameters for each model. The learning rate of the TCN is 0.0001 while the number of channels is 50.(*)

Hyperparameter	Drop Rate			
	0%	30%	50%	70%
Kernel Size	8	6	8	5
Number of Channels*	25	25	25	25
Number of Layers	12	12	11	11
Gradient Clip	0.5	0.5	0.5	0.5
Learning Rate*	0.001	0.001	0.001	0.001
Dropout	0.2	0.2	0.2	0.2
Number of Epochs	50	50	50	50
Scheduler Patience	3	3	3	3
Scheduler Factor	0.3	0.3	0.3	0.3
Scheduler Cooldown	0	0	0	0
Architecture	Number of Parameters			
GTCN	463,110	348,335	423,335	265,535
GTCN_hid	233,110	175,835	213,335	134,285
GTCN_sig	463,110	348,335	423,335	265,535
GTCN_sep	233,110	175,835	213,335	134,285
TCN	463,410	348,660	423,660	265,860

to perform the analysis. Due to the very long sequences in the speech commands dataset, architectures are set to be deep, which enables us to analyse the vanishing or exploding gradients and dead ReLU issues thoroughly. The hyperparameters of the architectures are provided in Table 6.5.

6.4.1 Regularly Sampled Speech Commands Dataset

Gradient analysis for the regularly sampled speech commands dataset is provided in Fig. 6.6. The peaks in the maximum of the absolute gradient plots indicate that *GTCN_sep*, *GTCN_sig*, and *GTCN_hid* suffer from exploding gradients. *GTCN_sep* having the highest maximum gradients explains the performance loss

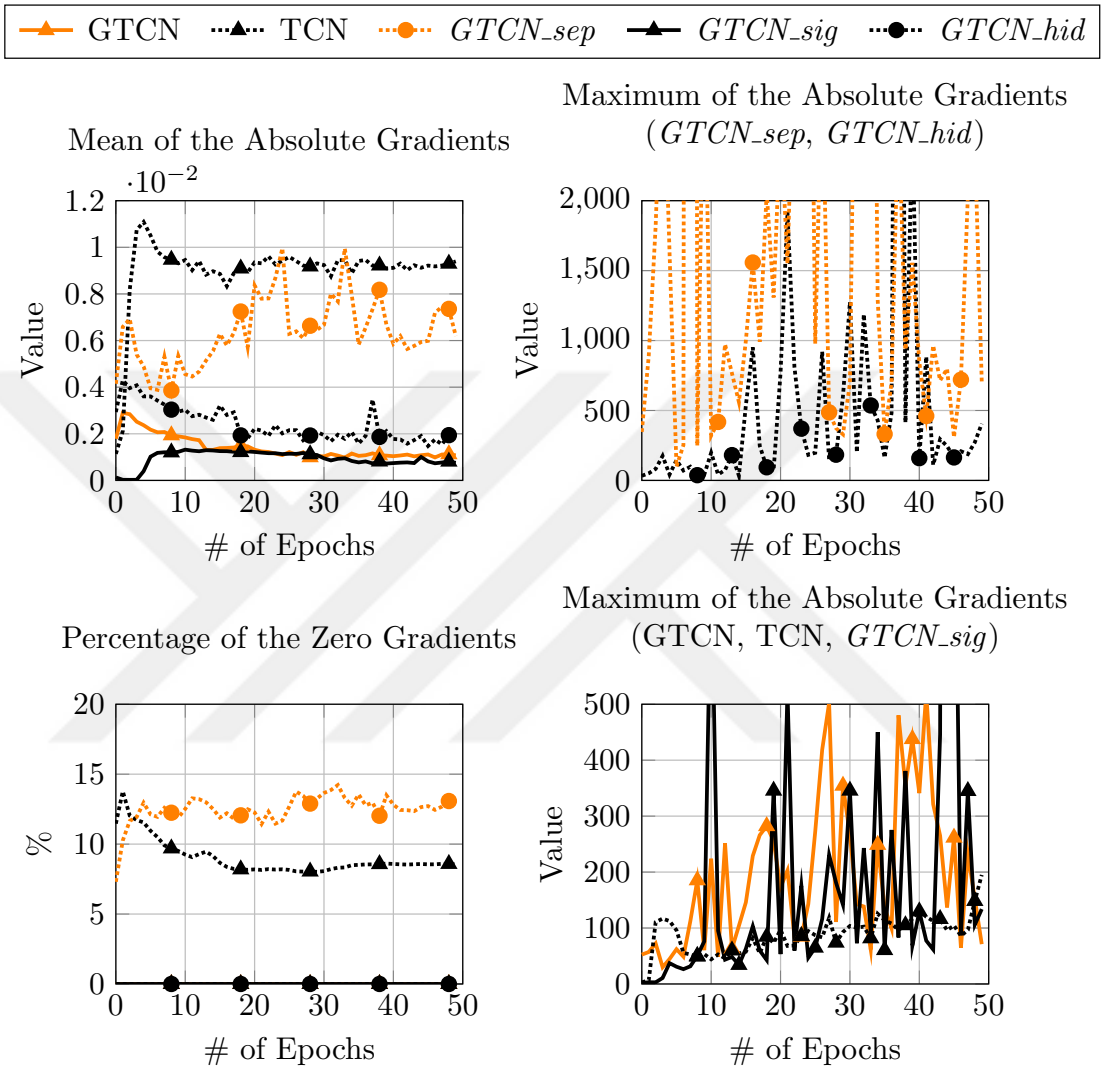


Figure 6.6. Gradient analysis for the TCN, GTCN, *GTCN_hid*, *GTCN_sig*, and *GTCN_sep* architectures on the speech commands dataset with regular sampling intervals.

in Fig. 6.3. The percentage of the zero gradients plot in Fig. 6.6 suggests that almost 10% of the neurons in TCN suffers from dead ReLU.

It is shown that *GTCN_hid* outperforms GTCN when the sampling intervals are regular in Table 6.3. However, gradient analysis shows that *GTCN_hid* is vulnerable to exploding gradients, which explains the performance loss as the drop rate increases.

6.4.2 Irregularly Sampled Speech Commands Dataset with 30% Drop Rate

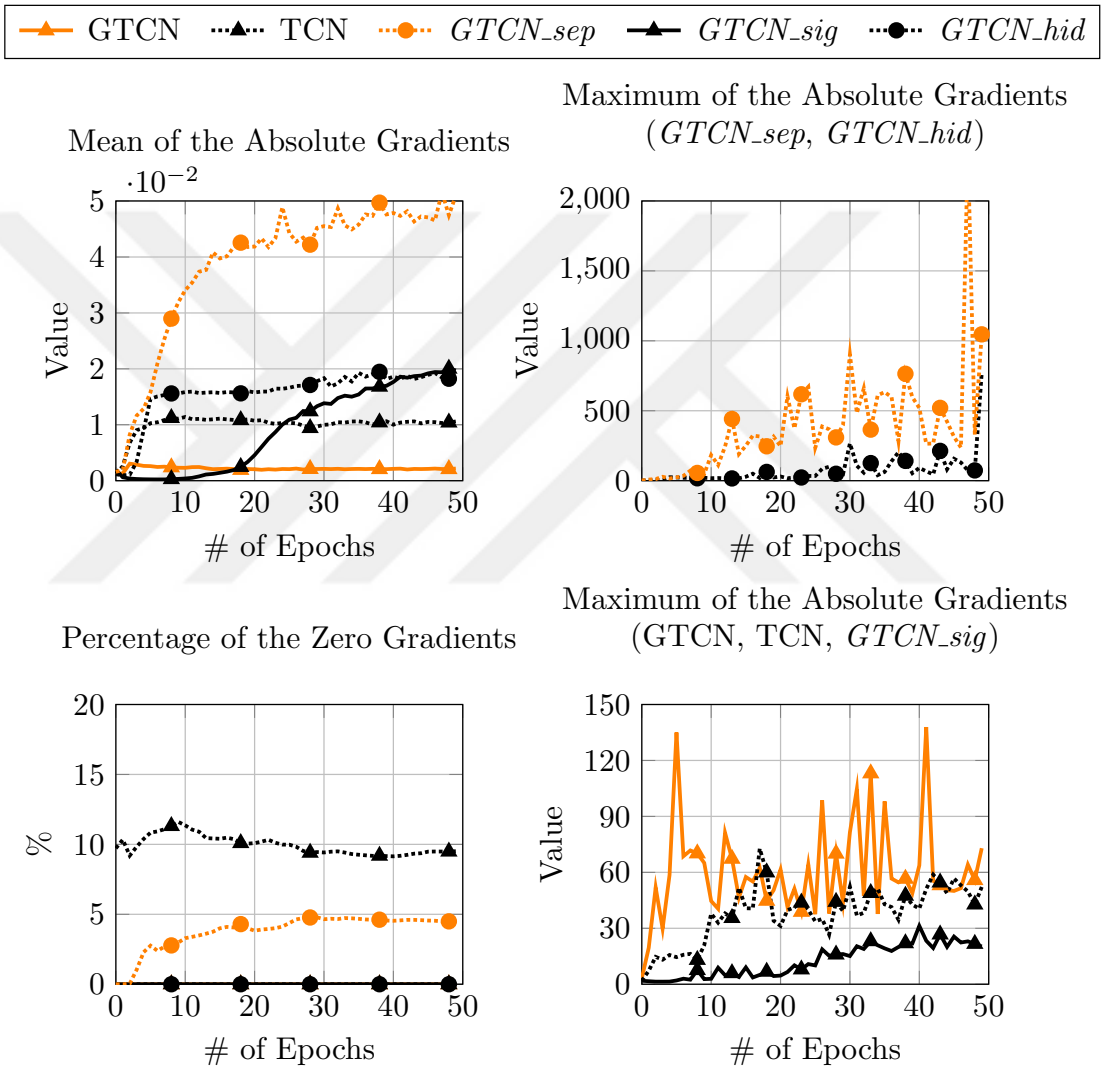


Figure 6.7. Gradient analysis for the TCN, GTCN, *GTCN_hid*, *GTCN_sig*, and *GTCN_sep* architectures on the speech commands dataset with 30% drop rate.

Gradient analysis for the irregularly sampled speech commands dataset with 30% drop rate is provided in Fig. 6.7. High value of both the maximum of the absolute gradients and the percentage of the zero gradients of *GTCN_sep* suggests that *GTCN_sep* architecture suffers from both vanishing and exploding gradients. Accuracy of *GTCN_sig* in Fig. 6.3 also indicates that *GTCN_sig* architecture is not able to model the speech commands sequences with irregular

sampling intervals. TCN, GTCN, and *GTCN_hid* have similar accuracies and gradients when the drop rate is 30%.

6.4.3 Irregularly Sampled Speech Commands Dataset with 50% Drop Rate

Gradient analysis for the irregularly sampled speech commands dataset with 50% drop rate is provided in Fig. 6.8. The maximum of the absolute gradients is lower for all the architectures, due to shorter sequences and shallow architectures. In addition, the percentage of the zero gradients of the TCN architecture is around 5%, which proves the vulnerability of ReLU activation functions in deeper networks. *GTCN_sep* and *GTCN_hid* have the highest value of the maximum of the absolute gradients. *GTCN_sig* having the least value of both the maximum of the absolute gradients and the mean of the absolute gradients suggest that *GTCN_sig* suffers from the vanishing gradients. Accuracy of *GTCN_sig* in Fig. 6.3 indicates that *GTCN_sig* is not able to learn when the drop rate is higher than 30%. Gradient analysis of *GTCN_sig* shows that the architecture is not able to learn due to the vanishing gradients.

6.4.4 Irregularly Sampled Speech Commands Dataset with 70% Drop Rate

Gradient analysis for the irregularly sampled speech commands dataset with 70% drop rate is provided in Fig. 6.9. It is evident that the negative effects of dead ReLU and vanishing or exploding gradients continue to reduce as the sequences get shorter except the *GTCN_sig* architecture. However, performance loss is inevitable due to the reduced information in the shorter sequences, as shown in Table 6.3. Gradients of the *GTCN_sig* architecture are very close to 0, which explains the accuracy of the *GTCN_sig* architecture in Fig. 6.3 and Table 6.3.

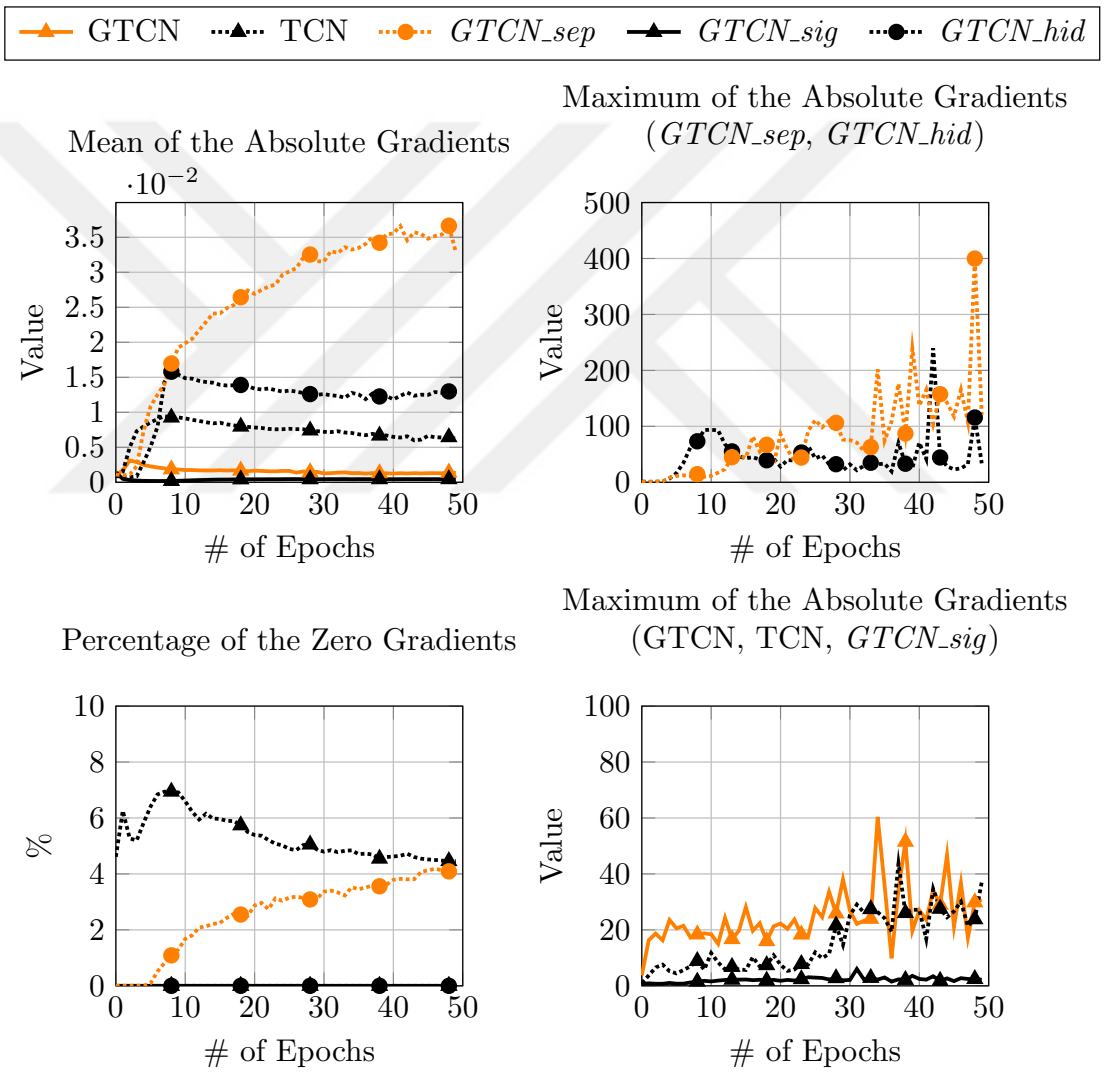


Figure 6.8. Gradient analysis for the TCN, GTCN, *GTCN_hid*, *GTCN_sig*, and *GTCN_sep* architectures on the speech commands dataset with 50% drop rate.

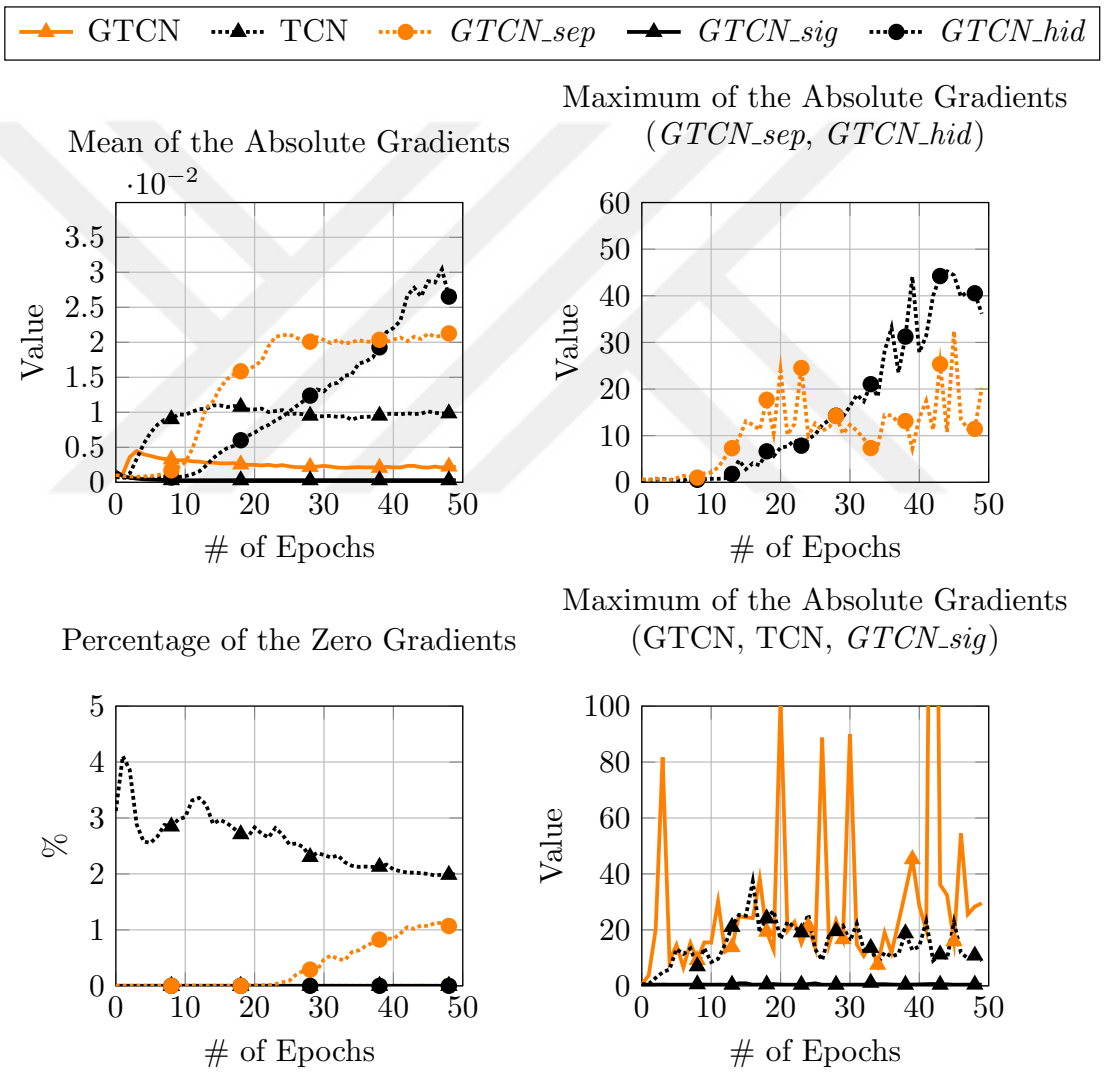


Figure 6.9. Gradient analysis for the TCN, GTCN, *GTCN_hid*, *GTCN_sig*, and *GTCN_sep* architectures on the speech commands dataset with 70% drop rate.

6.5 Visualisation of the Gate Information

We demonstrate the architecture's ability to model the irregularly sampled sequences by visualising the gate information. Despite the GTCN architecture being not specifically designed to extract features from images like 2D convolutional neural networks, the gate information can still be visualised using the sequential image datasets and the mask appending method. We use the sequential CIFAR10 dataset to visualise the gate information and provide insights on gated 1D convolutions and the modeling of sampling irregularities using the GTCN architecture. Gate informations are visualised after the sigmoid activation function in the GTCN.

6.5.1 Evolution of the Gate Information with respect to the Number of Epochs

The design of the GTCN architecture allows the gate information to evaluate the original inputs and the sampling information altogether. Therefore, the gate information focuses on both the sampling information and the semantic information present in the sequence itself. First layers focus on the sampling information more, while the deeper layers focus on both the semantic and the sampling information. Thus, visualisation of the first layer provides insights on the GTCN architecture's capability of modeling sampling information. Visualisation of the first layer is shown in Fig. 6.10.

The GTCN algorithm with the mask appending method is used to visualise the gate information. In order to visualise the gate information, handcrafted binary mask is used where the 256 pixels in the shape of a vertical rectangle are filled with gray, as shown in Fig. 6.10. The masked image shown in Fig. 6.10 is fed to the pretrained GTCN architecture in section 6.2 where 256 random pixels are filled with gray. The architectures are saved every epoch during training. The original image in Fig. 6.10 belongs to the test dataset of the CIFAR10. 5 channels from the first layer of the gate information are provided to demonstrate

the GTCN's ability to model the sampling information.

Gate information in Fig. 6.10 shows that the GTCN architecture is able to model the sampling information. Provided that nor the original image neither the mask is fed to the architecture during the training phase, sampling information is modeled successfully. However, despite being able to model the sampling information correctly, the architecture predicts this test image wrong.

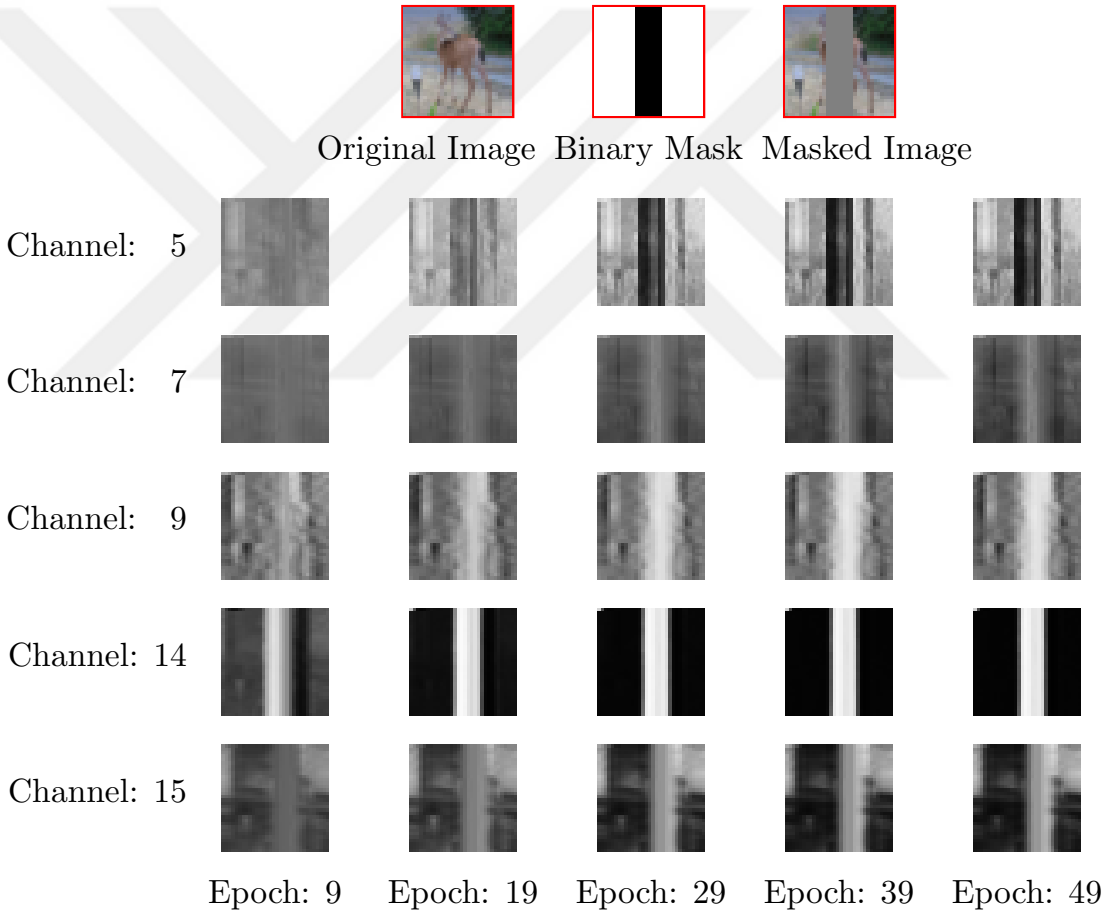
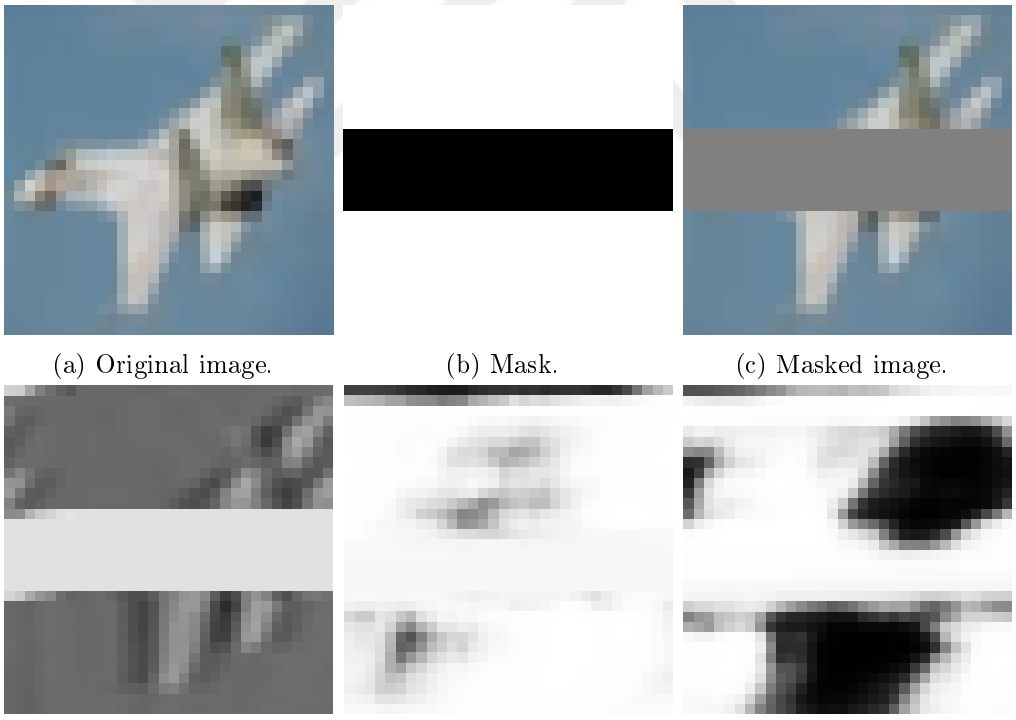


Figure 6.10. Evolution of the gate information in the first layer of the GTCN architecture with respect to the number of epochs.

6.5.2 Evolution of the Gate Information through Layers

It is demonstrated that gate information of the GTCN architecture learns to model the sampling information during the training. The gate information from

the different layers at the last epoch is provided in Fig. 6.11 to demonstrate that the gating mechanism also models the semantics present in the sequences. A different image and a different handcrafted binary mask is chosen and the masked image is fed to the same pretrained architecture from section 6.5.1. Fig. 6.11 shows that the gate information from the different layers of the GTCN architecture is able to model both the semantics and the sampling information. Despite extracting some semantic features in the deeper layers, the 1D temporal convolutional architectures such as Wavenet, GLU, TCN, and GTCN fall short to model the spatial relations in the image dataset with respect to 2D convolutions used in computer vision tasks. In Fig. 6.11f, the wings of the airplane is captured without details. The architecture predicts this test image correctly as plane.



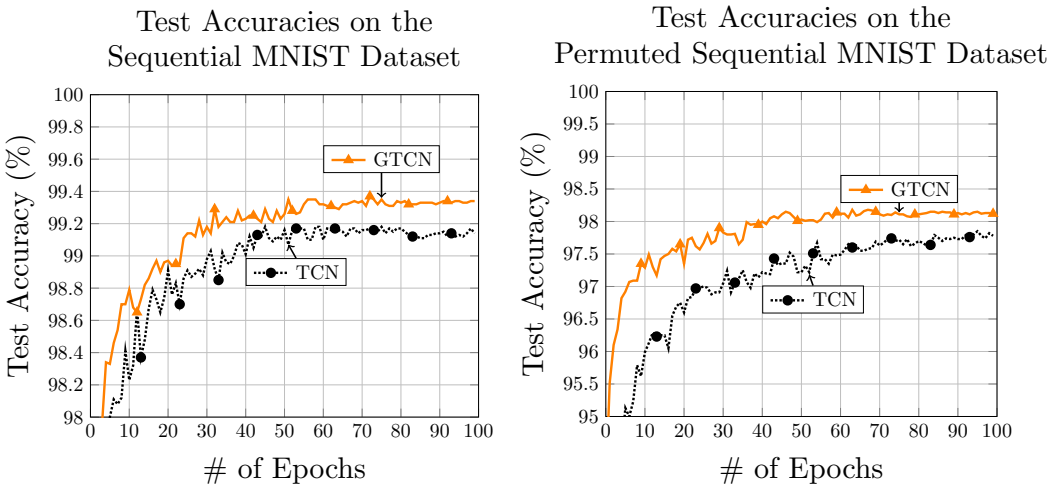
(a) Original image. (b) Mask. (c) Masked image.

(d) Layer 1, channel 19. (e) Layer 6, channel 24. (f) Layer 8, channel 19.

Figure 6.11. Gate information of the GTCN through deeper layers at the last epoch.

6.6 Comparison of TCN and GTCN with both Having Fewer Parameters

To clarify that the GTCN architecture is also able to model sequences by having fewer parameters, we provide a comparison between the TCN and GTCN architectures, with both having fewer parameters. We employ both algorithms to the sequential MNIST and the permuted sequential MNIST datasets. We set all hyperparameters except the number of filters in each layer according to Table 4.1. We set the number of filters in each layer as 10 for the GTCN architecture and 20 for the TCN architecture. The TCN architecture has 49,050 parameters, whereas the GTCN architecture has 48,930 parameters. The evolution of the test accuracies of the algorithms with almost 50k parameters is shown in Figure 6.12. The experiment indicates that the GTCN architecture is superior to the TCN, regardless of the number of parameters.



(a) Sequential MNIST dataset. (b) Permuted sequential MNIST dataset.

Figure 6.12. Test accuracies of the TCN and the GTCN architectures on the MNIST datasets with respect to the number of epochs. Both algorithms have around 50k parameters.

6.7 Hyperparameter Selection Procedures for the Architectures

We set the number of hidden layers and the kernel size of the architectures sufficient to capture the whole sequence. We experimented on the number of filters in each layer and tested the GTCN architecture using 10, 25, and 40 channels in each layer using datasets in Chapter 4. Experiments showed that the GTCNs with 25 channels in each layer efficiently capture the long-term dependencies. Thus, we set the number of filters to 25 for GTCNs in each experiment. We set the number of filters to 50 for TCNs to ensure that the number of parameters is comparable with the GTCNs. We performed a grid search for the learning rate, the gradient clip, and the scheduler parameters. In all experiments, we used *Adam* optimizer and *ReduceLROnPlateau* scheduler.

The hyperparameter selections for the experiments in Chapter 4 are shown in Table 4.1. The hyperparameter selections for the experiments with the Character Trajectories and the Speech Commands datasets are provided in Tables 5.2 and 5.3, respectively. In ablation studies, we set the hyperparameters of all architectures the same and provide the hyperparameter selections and the number of parameters in each algorithm in Table 6.2. In the experiments with the mask appending method in section 6.2 and sporadically sampled sequences in section 6.3, we set the hyperparameters of the GTCN architecture the same as the sequential CIFAR10 dataset with no missing values, which is shown in Table 4.1. For the gradient analysis in section 6.4, we follow the same methodology as in ablation studies and set the parameters of all architectures the same, except the TCN architecture. Hyperparameters for the gradient analysis experiments is provided in Table 6.5. Further, we provide the number of parameters for all architectures in the relevant tables. *Number of Layers* in Tables 4.1, 5.2, 5.3, 6.2, and 6.5 refers to the number of the residual blocks and gated residual blocks. Provided that one residual or gated residual block consists of two layers, actual number of layers is twice the shown value.

6.8 Limitations of the GTCN Architecture

Convolution based sequential architectures such as Wavenet, GLU, TCN and GTCN requires more disk space to store the parameters of the convolution operations with respect to the recurrent architectures. As a result, convolution based architectures can not be used in applications with memory constraints. Another limitation of the GTCN architecture is having more parameters than the generic TCN architecture. However, the number of parameters does not restrict the efficiency of the GTCN in most cases, as shown in section 6.6. Experiments on the Character Trajectories dataset in Chapter 6 show that the GTCN performs poorly when the sequences are not very long.

Chapter 7

Conclusion

We have studied sequential modeling and introduced a novel gating mechanism to be used on the basic TCN [1] architectures. The GTCN architecture is able to model the long-term dependencies and the irregularities in the sampling intervals in the sequential data. By this gated architecture, we avoid dying activation functions and alleviated the vanishing or exploding gradient problem by altering the way of gradient flow. The GTCN algorithm has a wide range of application areas where the sequences are sampled regularly or irregularly. Through extensive set of experiments, we demonstrate that the basic GTCN architecture is capable of modeling long-term dependencies even when the sequences have $16,000$ observations and able to handle the sequences with missing values even when the 75% of the sequences are dropped. We achieve significant performance gain in various well-known benchmarks and real-life applications compared to the state-of-the-art results. We also provide insights on gradients of the architectures and visualisation of the gate information. Comparison with the state-of-the-art architectures and the ablation study demonstrates that the design choice of the GTCN is efficient in modeling sequential data which sampled regularly, irregularly or sporadically.

Bibliography

[1] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[2] X. Yan, W. Zhang, L. Ma, W. Liu, and Q. Wu, "Parsimonious quantile regression of financial asset tail dynamics via sequential learning," in *NeurIPS* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 1582–1592, 2018.

[3] E. Choi, C. Xiao, W. F. Stewart, and J. Sun, "Mime: Multilevel medical embedding of electronic health records for predictive healthcare," in *NeurIPS* (S. Bengio, H. M. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, eds.), pp. 4552–4562, 2018.

[4] S. Wang, Y. Zeng, X. Liu, E. Zhu, J. Yin, C. Xu, and M. Kloft, "Effective end-to-end unsupervised outlier detection via inlier priority of discriminative network," in *NeurIPS* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 5960–5973, 2019.

[5] A. C. Singer, G. W. Wornell, and A. V. Oppenheim, "Nonlinear autoregressive modeling and estimation in the presence of noise," *Digital Signal Processing*, vol. 4, no. 4, pp. 207–221, 1994.

[6] C. Szegedy, A. Toshev, and D. Erhan, "Deep neural networks for object detection," in *NeurIPS* (C. J. C. Burges, L. Bottou, Z. Ghahramani, and K. Q. Weinberger, eds.), pp. 2553–2561, 2013.

[7] S. Khodadadeh, L. Bölöni, and M. Shah, "Unsupervised meta-learning for few-shot image classification," in *NeurIPS* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 10132–10142, 2019.

[8] T. Ergen and S. S. Kozat, "Unsupervised anomaly detection with lstm neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 31, no. 8, pp. 3127–3141, 2019.

[9] J. L. Elman, "Finding structure in time," *Cognitive science*, vol. 14, no. 2, pp. 179–211, 1990.

[10] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[11] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *Proceedings of SSST@EMNLP 2014, Eighth Workshop on Syntax, Semantics and Structure in Statistical Translation, Doha, Qatar, 25 October 2014* (D. Wu, M. Carpuat, X. Carreras, and E. M. Vecchi, eds.), pp. 103–111, Association for Computational Linguistics, 2014.

[12] I. Goodfellow, Y. Bengio, A. Courville, and Y. Bengio, *Deep learning*, vol. 1. MIT press Cambridge, 2016.

[13] D. W. Romero, A. Kuzina, E. J. Bekkers, J. M. Tomczak, and M. Hoogendoorn, "Ckconv: Continuous kernel convolution for sequential data," *arXiv preprint arXiv:2102.02611*, 2021.

[14] Z. Che, S. Purushotham, K. Cho, D. Sontag, and Y. Liu, "Recurrent neural networks for multivariate time series with missing values," *Scientific Reports*, vol. 8, no. 1, pp. 1–12, 2018.

[15] Z. C. Lipton, D. C. Kale, C. Elkan, and R. C. Wetzel, "Learning to diagnose with LSTM recurrent neural networks," in *ICLR* (Y. Bengio and Y. LeCun, eds.), 2016.

[16] X. Shi, Z. Chen, H. Wang, D. Yeung, W. Wong, and W. Woo, "Convolutional LSTM network: A machine learning approach for precipitation nowcasting," in *NeurIPS* (C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, eds.), pp. 802–810, 2015.

[17] J. Bradbury, S. Merity, C. Xiong, and R. Socher, "Quasi-recurrent neural networks," in *ICLR*, OpenReview.net, 2017.

[18] A. van den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. W. Senior, and K. Kavukcuoglu, "Wavenet: A generative model for raw audio," in *The 9th ISCA Speech Synthesis Workshop, Sunnyvale, CA, USA, 13-15 September 2016*, p. 125, ISCA, 2016.

[19] Y. N. Dauphin, A. Fan, M. Auli, and D. Grangier, "Language modeling with gated convolutional networks," in *International Conference on Machine Learning*, pp. 933–941, PMLR, 2017.

[20] D. Neil, M. Pfeiffer, and S. Liu, "Phased LSTM: accelerating recurrent network training for long or event-based sequences," in *NeurIPS* (D. D. Lee, M. Sugiyama, U. von Luxburg, I. Guyon, and R. Garnett, eds.), pp. 3882–3890, 2016.

[21] S. O. Sahin and S. S. Kozat, "Nonuniformly sampled data processing using lstm networks," *IEEE Transactions on Neural Networks and Learning Systems*, vol. 30, no. 5, pp. 1452–1461, 2018.

[22] E. D. Brouwer, J. Simm, A. Arany, and Y. Moreau, "Gru-ode-bayes: Continuous modeling of sporadically-observed time series," in *NeurIPS* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 7377–7388, 2019.

[23] P. Kidger, J. Morrill, J. Foster, and T. J. Lyons, "Neural controlled differential equations for irregular time series," in *NeurIPS* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), 2020.

[24] M. Lechner and R. M. Hasani, "Learning long-term dependencies in irregularly-sampled time series," in *NeurIPS* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), 2020.

[25] S. Chang, Y. Zhang, W. Han, M. Yu, X. Guo, W. Tan, X. Cui, M. J. Witbrock, M. A. Hasegawa-Johnson, and T. S. Huang, "Dilated recurrent neural networks," in *NeurIPS* (I. Guyon, U. von Luxburg, S. Bengio, H. M. Wallach, R. Fergus, S. V. N. Vishwanathan, and R. Garnett, eds.), pp. 77–87, 2017.

[26] R. H. Hahnloser, R. Sarpeshkar, M. A. Mahowald, R. J. Douglas, and H. S. Seung, "Digital selection and analogue amplification coexist in a cortex-inspired silicon circuit," *Nature*, vol. 405, no. 6789, pp. 947–951, 2000.

[27] L. Lu, Y. Shin, Y. Su, and G. E. Karniadakis, "Dying relu and initialization: Theory and numerical examples," *arXiv preprint arXiv:1903.06733*, 2019.

[28] T. H. Trinh, A. M. Dai, T. Luong, and Q. V. Le, "Learning longer-term dependencies in rnns with auxiliary losses," in *ICML* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 4972–4981, PMLR, 2018.

[29] 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.

[30] P. Warden, "Speech commands: A dataset for limited-vocabulary speech recognition," *arXiv preprint arXiv:1804.03209*, 2018.

[31] S. Li, W. Li, C. Cook, Y. Gao, and C. Zhu, "Deep independently recurrent neural network (indrnn)," *arXiv preprint arXiv:1910.06251*, 2019.

[32] S. Bai, J. Z. Kolter, and V. Koltun, "Trellis networks for sequence modeling," in *ICLR*, OpenReview.net, 2019.

[33] A. Gu, T. Dao, S. Ermon, A. Rudra, and C. Ré, "Hippo: Recurrent memory with optimal polynomial projections," in *NeurIPS* (H. Larochelle, M. Ranzato, R. Hadsell, M. Balcan, and H. Lin, eds.), 2020.

[34] T. K. Rusch and S. Mishra, "Unicornn: A recurrent model for learning very long time dependencies," *arXiv preprint arXiv:2103.05487*, 2021.

[35] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[36] A. Krizhevsky, G. Hinton, *et al.*, "Learning multiple layers of features from tiny images," 2009.

[37] A. Bagnall, H. A. Dau, J. Lines, M. Flynn, J. Large, A. Bostrom, P. Southam, and E. Keogh, "The uea multivariate time series classification archive, 2018," *arXiv preprint arXiv:1811.00075*, 2018.

[38] Y. Rubanova, T. Q. Chen, and D. Duvenaud, "Latent ordinary differential equations for irregularly-sampled time series," in *NeurIPS* (H. M. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. B. Fox, and R. Garnett, eds.), pp. 5321–5331, 2019.