Neural Sequential Modeling and Applications

Guokun Lai

CMU-LTI-21-010

Language Technology Institute
School of Computer Science
Carnegie Mellon University
5000 Forbes Ave., Pittsburgh, PA 15213
www.lti.cs.cmu.edu

Thesis Committee:
Yiming Yang (Chair), Carnegie Mellon University
Alan W. Black, Carnegie Mellon University
Emma Strubell, Carnegie Mellon University
Hanxiao Liu, Google Research

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy
In Language and Information Technologies

Contents

List of Figures

List of Tables

Abstract

How to model sequential data in various settings is an important machine learning problem across many domains, including predictions over time series data, natural language text, and event streams. Sequential data in different fields usually have different characteristics. For example, natural-language text can be viewed as a a sequence of a discrete variable while sensor-network signals can be treated as a multi-variate sequence in a continuous vector space. In order to develop successful neural network models in such various real-world domains, we need to customize the architectures and algorithms based on the nature of the data and the problems. This thesis designs novel and efficient neural network solutions for the sequential modeling and applications. Specifically, the contributions can be grouped into four parts.

- The first part focuses on the correlation among variables in the multivariate sequential data, such as the time series of multiple sensors, and proposes novel algorithms namely *Depthwise Separable Graph Convolution Network* (DSGC) (Chapter 2) [60] and *Factorized Recurrent Neural Newtork* (FRNN) (Chapter 3) [63] for leveraging correlation patterns and improving prediction accuracy.
- The second part focus on incorporating human prior knowledge in temporal modeling of dependency pattens in sequential data. Specifically, we propose a novel approach named *the Long- and Short-term Time-series Network* (LSTNet) (Chapter 4) [59] which is proven to be particularly effective for capturing various periodic patterns in different applications.
- The third part focuses on efficient algorithms for Transformers in sequence classification tasks. Specifically, by identifying the computation redundancy in the commonly used Transformer architectures and by proposing a novel replacement namely the *Funnel Transformer* (Chapter 5) [27], we achieve a better trade-off between computation and accuracy.
- The fourth part focuses on the modeling/prediction of the temporal relationship among events, where the major challenge is effective learning from sparsely labeled data. We address this challenge via the combination of advanced data augmentation, semi-supervised learning and introduction of human prior knowledge (Chapter 6). As a result, we improve the state-of-the-art performance of this task by a large margin.

Chapter 1

Introduction

1.1 Background and motivation

In the past decade, machine learning, especially deep learning, has achieved great success in multiple real-world applications, such as image classification [58], speech recognition [107] and natural language processing [30]. To achieve better performance in these domains, researchers have developed different neural network architectures and algorithms to accommodate different characteristics of data. For example, in the computer vision domain, the convolution neural network is the dominant architecture due to its ability to capture shift-invariant features of images. In the language-related domain, the language modeling style pretraining is the most popular algorithm, because it can take the advantage of the large-scale unlabeled data in this domain and leads to the best performance.

The sequential structure is a very important data structure in real-world applications, including time-series data from sensors, natural language, event stream from news and social networks, and so on. The machine learning problems with sequential data have been extensively studied in past decades. Many classic algorithms were proposed, such as the auto-regression method, Hawkes Process, and N-gram Language model. In the last decade, the neural networks were playing the most important role in solving problems with sequential data. Among these problems, classical ones in the natural language domain, such as language modeling and machine translation, have been extensively studied and well addressed by applying the deep neural network. However, how to extend the success to other types of sequential data is still an open challenge. For example, for time-series data, which is in the form of real numbers, the scale of data would have a significant influence on the prediction stability and accuracy of the model, but it would not be a concern for the natural language which is presented as discretized labels. Another example is that how to model dependency among the variables of multi-variate sequential data is a key factor of the accuracy of the model. But it is not considered in the network design for natural language which only has one discrete variable.

In summary, different types of sequential data have their own characteristics. To maximize the performance of the machine learning models, we need to inject the prior knowledge of problems and data structures to models. In this thesis, we customize the deep learning architecture and algorithm in different applications with sequential data according to their own characteris-

tics, and achieve better performance compared with previous methods. Our contribution to this topic can be summarized into four parts:

- Capturing the correlation of variables in the multivariate sequential data (DSGN and FRNN).
- Learning periodic temporal dependency from data efficiently with human prior knowledge (LSTNet).
- Efficient Transformer architecture for the sequence classification task (Funnel-Transformer).
- Event temporal modeling with sparse labels.

In the following, we detail all the contributions listed above.

1.2 Capture the intra-step dependency in the multivariate sequential data

1.2.1 Depthwise Separable Graph Convolution (DSGC) [60]

In Chapter 2, we focus on the spatiotemporal data, such as the temperature and solar energy output. Each of these sequential variables represents a sensor with geographic location information. With this kind of data, we can utilize the geometric information to capture the correlations among the variables when we are learning the temporal dependency. To model the spatial feature, the most popular approach is the 2D convolution neural network [58] in the image domain. But the 2D convolution cannot be applied to real-world data without a regular 2D-grid structure. A workaround is to use graph convolution method (GCN) [14, 29, 55], which can take any graph structure data as input. The problem of GCN is that they rely on the input graph structure to model the interaction between nodes. They don't learn spatial features from data.

To combine the advantages of 2D convolution and Graph convolution, we propose Depthwise Separable Graph Convolution (DSGC), a novel graph convolution approach derived from the 2D-convolution method. It inherits the strength of depthwise separable convolution, which has contributed to multiple state-of-the-art image classification frameworks including Inception Network [104], Xception Network [20] and MobileNet [47]. Compared with the previous graph and geometric methods, DSGC is more expressive and compatible with the depthwise separable convolution network and shares the desirable characteristic of small parameter size as in the depthwise separable convolution. In the experiment section, the DSGC produces the new state-of-the-art results in 3 different spatiotemporal datasets.

1.2.2 Factorized Recurrent Neural Network (FRNN) [63]

Except for the spatiotemporal data, there is multivariate time series data without usable side features, which does not allow us to construct a graph for the DSGC method. In Chapter 3, we study how to capture the correlation between variables in this scenario. The naive approach is viewing the problem as a multitask learning problem [59]. It relies on the hidden layers in the neural models to capture dependency implicitly. In the output layer, the model makes predictions on the different variables independently. Another popular method in previous publications [1, 6, 22, 34, 38, 62] is to introduce stochastic latent variables to model the dependency among

variables. In their experiments, they show a significant improvement over the multitask style approaches.

In our work, We first re-exam both style approaches from both theoretical and experimental perspectives. In summary, our re-examination reveals that the stochastic latent variables are serving as a way to explicitly model the correlation between output variables. Finally, we propose a more powerful approach according to our mathematical analysis, which is applying an auto-regressive loss inside each time step and over the dimension of the variables. During the inference, it predicts the ith variable based on all variables with the index smaller than i. The experiments show the proposed method is better than the stochastic model style algorithms and able to capture the correlation between variables without side information.

1.3 Long- and Short-term Time-series Network (LSTNet) [59]

Besides capturing the dependency of variables in the sequential data, we also explore how to better model the temporal dependency of the sequential data by leveraging the prior knowledge for specific tasks. Time series forecasting is a long-standing research problem. The autoregressive (AR) model is the most representative one among the linear models designed for this problem. Its variant, autoregressive integrated moving average - ARMIA model [9], is viewed as the best forecasting model for decays until the neural network model appears. A big class of temporal data in real life is the periodic time series, such as electricity usage and traffic occupation. To introduce the human prior about the period to the learning model, the researchers have proposed to integrate the Fourier transformation with ARIMA method [128]. But there is no work to study how to leverage periodic information with the neural network techniques.

In Chapter 4, we propose Long- and Short-term Time-series Network (LSTNet), which is the first deep learning framework designed for periodic time series forecasting. A novel recurrent structure, namely Recurrent-skip, is designed for capturing very long-term dependence patterns and making the optimization easier as it utilizes the periodic property of the input time-series signals. Additionally, the LSTNet incorporates a traditional autoregressive linear model in parallel to the non-linear neural network part, which makes the non-linear deep learning model more robust for the time series with violated scale changing. Experiments over real-world seasonal time series datasets consistently show the stronger performance of the proposed model over the traditional linear models and recurrent neural network. In addition, we also perform a statistical analysis to reveal that the neural network model performs better on the data with strong temporal correlation, while on the more chaotic data, such as stock prices and currency exchange rate, the neural network model performance would be similar to the linear ones.

1.4 Funnel Transformer [27]

In Chapter 5, we shift our attention to the sequence classification problem. We propose an efficient Transformer architecture for this problem. The Transformer model [110] has been proved as a strong neural architecture to deal with sequential data, especially with the help of the pretraining technique [30]. However, the classification problems, such as time series classification and

sentence classification, only require a hidden representation for the whole sequence. The Transformers always maintain a full-length token-level representation. To improve the efficiency, we examine this much-overlooked redundancy. With this intuition, we propose Funnel-Transformer which gradually compresses the sequence of hidden states to a shorter one and hence reduces the computation cost. More importantly, by re-investing the saved FLOPs from length reduction in constructing a deeper or wider model, we further improve the model capacity. In addition, to perform token-level predictions as required by common pretraining objectives, Funnel-Transformer is able to recover a deep representation for each token from the reduced hidden sequence via a decoder. Empirically, with comparable or fewer FLOPs, Funnel-Transformer outperforms the standard Transformer on a wide variety of sequence-level classification tasks and achieves better efficiency and accuracy trade-off.

1.5 Event Temporal Modeling with Sparse Labels

In Chapter 6, we focus on utilizing the advanced deep learning techniques to solve a real-world application with sequential data, event temporal relationship classification. This task takes natural language text and the extracted events as input and predicts the temporal relationship labels of each pair of events. By solving this task, it can provide us a tool to summarize and structure news streams. However, the events and temporal relationship both require human annotation, which is expensive. The data sparsity issue is the main challenge of this task. A decade ago, the main approach for this task was based on Hawkes Process, which predefined a mixture of Bayesian processes for the event distribution based on human prior knowledge. Until the advent of pre-trained language models [30], we can use limited size data to achieve better performance. In this work, We first study and compare several baseline methods on this task, including linear model, Hawkes Process [45], Neural Hawkes Process [80], and pretrained language model [77]. Although the pretrained language model already achieved strong results, there are no works to combine the task-specific prior knowledge and advanced regularization technique to better address the data sparsity issue when using the pretrained language model. Based on this intuition, we propose 3 remedies:

- Leverage the human prior knowledge about event temporal sequence includes symmetry and transitivity.
- Data augmentation, which maximizes the labeled data utilization.
- Utilizing the unlabeled data to make our model more robust.

With these improvements, we gain a significant performance boost compared to the baseline methods and achieve the new state-of-the-art results on the benchmark dataset.

Chapter 2

Depthwise Separable Graph Convolution

2.1 Background and Motivation

Convolution Neural Network (CNN) [67], also referred to as 2D-convolution in the following chapter, has been proven to be an efficient model family in extracting hierarchical local patterns from grid-structured data, which has significantly advanced the state-of-the-art performance of a wide range of machine learning tasks, including image classification, object detection and audio recognition [68]. Recently, growing attention has been paid to dealing with data with a non-grid structure, such as prediction tasks in sensor networks [119], transportation systems [72], and 3D shape correspondence application in the computation graphics [13]. How to replicate the success of CNNs for manifold-structured data remains an open challenge.

In this chapter, we provide a unified view of the 2D-convolution methods and the graph convolution (including the geometric convolution) with the label propagation process [129]. To best of our knowledge, it is the first time that the 2D-convolution and graph convolution proposed in [55] are unified mathematically. It helps us better understand and compare the difference between them, and shows that the fundamental difference can be summarized as two points, (1) 2D-convolution learns spatial filters from the data. (2) Spatial filters in 2D-convolution are channel-specific.

Many graph convolution and geometric convolution methods have been proposed recently. The spectral convolution methods are the mainstream algorithms developed as the graph convolution methods. Their theory is based on the graph Fourier analysis [96]. Another group of approaches are geometric convolution methods, which focus on various ways to leverage spatial information about nodes [7, 37, 79, 82]. Existing models mentioned above are either fully trusting the given graph or applying one graph filter across all channels, which are corresponding to the two differences between the traditional 2D-convolution and the graph convolution. Firstly, as a result of trusting the given graph, namely only using a given graph filter across the whole model, the model ability to discover the special graph filters from the supervision data is limited. And applying one graph filter across all channels would also introduce several drawbacks to the model as follows. (1) It makes the mathematical formulation of the graph convolution methods incompatible with the traditional 2D-convolution. (2) The model cannot propagate information with different diffusion patterns in one layer. (3) The image recognition experiment in Section

2.5.2 shows that multiple filters are critical to the model performance in the task which requires extracting complex local features. Some models, such as MoNet [82] and Graph Attention Network [111], try to model multiple filters by simultaneously learning K sub-layers, each with one global filter, and summarizes them as one layer. However, this approach would lead to a larger number of parameters and more expensive computation cost, and it is still incompatible with the traditional 2D-convolution method.

In this chapter, we derive a novel graph convolution approach directly from the 2D-convolution method. We propose the Depthwise Separable Graph Convolution (DSGC), which inherits the strength of depthwise separable convolution that has been extensively used in different state-of-the-art image classification frameworks including Inception Network [104], Xception Network [20] and MobileNet [47]. Compared with previous graph and geometric methods, the DSGC is more expressive and compatible with the depthwise separable convolution network, and shares the desirable characteristic of small parameter size as in the depthwise separable convolution. In experiments section, we evaluate the DSGC and baselines in three different machine learning tasks. The experiment results show that the performance of the proposed method is close to the standard convolution network in the image classification task on CIFAR dataset. And it outperforms previous graph convolution and geometric convolution methods in all tasks. Furthermore, we demonstrate that the proposed method can easily leverage the state-of-the-art architectures developed for image classification to enhance the model performance, such as the Inception module [104], the dense block [49] and the Squeeze-and-Excitation block [48].

The main contribution of this chapter is threefold:

- A unified view of traditional 2D-convolution and graph convolution methods by introducing depthwise separable convolution.
- A novel Depthwise Separable Graph Convolution (DSGC) for data residing on arbitrary manifolds.
- We demonstrate the efficiency of the DSGC module with extensive experiments and show that it can be plugged into existing state-of-the-art CNN architectures to improve the performance for graph tasks.

2.2 A Graph Perspective of Convolution

In this section, we provide a unified view of several convolution operations by showing that they are different message aggregation protocols over the graphs or manifolds. Unless otherwise specified, we denote a matrix by $\boldsymbol{X}$, the i-th row in the matrix by $\boldsymbol{x}_i$, and the (i, j)-th element in the matrix by x_{ij}. Superscripts are used to distinguish different matrices when necessary. All the operations below can be viewed as a function that transforms input feature maps $\boldsymbol{X} \in \mathbb{R}^{N \times P}$ to output feature maps $\boldsymbol{Y} \in \mathbb{R}^{N \times Q}$, where N is the number of nodes and P, Q are the number of its associated input and output features (channels) respectively. We use $\boldsymbol{G}$ to denote the adjacency matrix of a graph, and $G(i)$ to denote the set of neighbors for node i. For tasks over sensor networks [119], transportation graphs [72] or computational graphics [13], graph G often corresponds to the latent structure of the underlying manifold, and is induced from the spatial coordinates of input data.

2.2.1 Convolution over Graphs

For the operations discussed below, the filter weights are fully determined by the given graph G.

Label Propagation

Label propagation (LP) [129] can be viewed as a simplistic convolution operation to aggregate local information over a graph:

$$\boldsymbol{y}_i = \sum_{j \in G(i)} G_{ij} \boldsymbol{x}_j \tag{2.1}$$

In other words, the feature map for each node is updated as the weighted combination of its neighbors' feature maps. In this case, the numbers of input and output channels are identical.

Graph Convolution

Graph convolution [55] (GC) can be viewed as an extension of LP, formulated as:

$$\boldsymbol{y}_i = \sum_{j \in G(i)} G_{ij} \boldsymbol{z}_j \quad where \quad \boldsymbol{z}_j = \boldsymbol{x}_j \boldsymbol{U} \tag{2.2}$$

While both LP and GC utilize the graph structure in $\boldsymbol{G}$, GC has an learnable linear transformation $\boldsymbol{U} \in \mathbb{R}^{P \times Q}$ that maps $\boldsymbol{x}_j$ into the intermediate representation $\boldsymbol{z}_j$. This additional step enables GC to capture the dependencies among channels.

2.2.2 Convolution over Grid-Structures

Here, we write the convolution methods over 2D-grid in the Label Propagation framework. Let Δ_{ij} be the coordinate offset from i-th node to j-th node, we say $j \in G(i)$ if j is one of i's k-nearest neighbors based on the relative distance $|\Delta_{ij}|$.

Full Convolution

The full convolution [67] can be formulated as

$$y_{iq} = \sum_{j \in G(i)} \sum_{p=1}^{P} w_{\Delta_{ij}}^{(pq)} x_{jp} \tag{2.3}$$

For Euclidean grid-structured data such as images, Δ_{ij} denotes the offset between pixel i and pixel j, and $G(i)$ contains pixel i's surrounding pixels. For example, the size of $G(i)$, or k, is 9 for 3×3 convolution and 25 for 5×5 convolution, corresponding to the size of the receptive field. The full convolution operation captures the channel correlation and spatial correlation simultaneously by $\boldsymbol{W}^{(pq)}$.

Depthwise Separable Convolution

The Depthwise Separable Convolution (DSC) [20] is a factorized version of full convolution under the intuition that the channel correlation and spatial correlation can be decoupled. In practice, DSC is able to achieve comparable performance as full convolution with a substantially smaller number of parameters. We focus on DSC here due to its simplicity and intimate connections to Graph Convolution. DSC can be formulated in a graph-based fashion as follows

$$y_{iq} = \sum_{j \in G(i)} w^{(q)}_{\Delta_{ij}} z_{jq} \quad where \quad \boldsymbol{z}_j = \boldsymbol{x}_j \boldsymbol{U} \tag{2.4}$$

The formulation of DSC is analogous to GC by substituting $\boldsymbol{G}$ in eq. (2.2) with $\boldsymbol{W}$. However, unlike LP and GC which directly utilize the graph $\boldsymbol{G}$ to define their filter weights, weights $\boldsymbol{W}$ in eq. (2.4) is a learnable lookup table of size $Q \times R$, where R is the number of possible choices for Δ_{ij}. For example, $R = 9$ for 3×3 convolution, since Δ_{ij} can take any value in $\{-1, 0, 1\} \times \{-1, 0, 1\}$.

2.3 Depthwise Separable Graph Convolution

2.3.1 Motivation

We notice that DSC is more powerful than GC in the following aspects:

1. The spatial filter in GC is fully determined once the graph is given [1], but the spatial filters in DSC are learned automatically from data. This means GC would fully trust the given graph even if it is suboptimal for the task and data on hand.
2. Compared with full convolution and DSC, which are capable of modeling channel-specific convolution filters, GC uses a global spatial filter for all channels (features), which can be viewed as a restricted version of DSC. Thus a GC module is unable to simultaneously capture or fuse diverse information based on different channels/features over the graph.

On the other hand, while GC is generally applicable to arbitrary graphs, the DSC method so far is only designed for regular grid-based structures and hence only applicable to the domains like image processing, where the pixels naturally form a grid structure. For nodes scattered with arbitrarily spatial coordinates, the number of possible choices for Δ_{ij} can be infinite. That is, using a lookup table $\boldsymbol{W}$ to memorize the filter weights for each Δ_{ij} is no longer feasible. This makes traditional DSC not directly applicable to arbitrary graphs.

2.3.2 Proposed Method

To address the aforementioned limitations of Graph Convolution, we propose Depthwise Separable Graph Convolution (DSGC), which naturally generalizes DSC and GC as:

$$y_{iq} = \sum_{j \in G(i)} w^{(q)}(\Delta_{ij}) z_{jq} \quad where \quad \boldsymbol{z}_j = \boldsymbol{x}_j \boldsymbol{U} \tag{2.5}$$

[1]The linear transformation $\boldsymbol{U}$ in GC is not a graph/spatial filter, as it only fuses the information across the channels.

where we slightly abuse the notation by overloading $w^{(q)}(\cdot)$ as a function (neural network) that maps Δ_{ij} to a real scalar, namely the predicted filter weight for the q-th channel.

The key distinctions in our formulation are as follows.

1. Different from DSC (eq. (2.4)), the filter weight is calculated using a "soft" function approximator. That is, DSGC predicts the convolution filter weights from Δ_{ij} via the function instead of memorizing them in a look-up table. In our experiments, function $w^{(q)}(\cdot)$ is implemented as a two-layer MLP.
2. Different from GC (eq. (2.2)), DSGC enables the learning of channel-specific spatial convolution filters (channels are indexed by q in eq. (2.5)). This amounts to simultaneously constructing channel-specific graphs under the different node-node similarity metrics, where the metrics are implicitly defined by neural networks and hence, are jointly optimized during the training.

The idea of predicting the filter weights has also been explored in Message Passing Neural Network (MPNN) [37]. However, MPNN learns only a global function across all channels, hence, MPNN is incapable of capturing channel-specific spatial filters as in DSC.

2.3.3 Parameter Grouping Strategy

Overfitting is a common issue in graph-based applications due to limited data available. To alleviate this issue, a simple strategy is to group the original Q channels into C groups, where $D = Q/C$ channels in the same group would share the same filter:

$$w^{(q)}(\cdot) = w^{(q')}(\cdot) \quad if \quad \lfloor \frac{q}{D} \rfloor = \lfloor \frac{q'}{D} \rfloor \tag{2.6}$$

where $\lfloor \cdot \rfloor$ denotes the floor function.

2.3.4 Filter Normalization

A common practice in label propagation and graph convolution is to normalize the adjacency matrix for G. In DSGC, a natural way to carry out normalization is to apply a softmax function as the final layer of the filter weights predictor, to ensure that $\sum_{j \in G(i)} w^{(q)}(\Delta_{ij}) = 1$ for each i. We empirically found that normalization leads to improved performance and faster convergence.

2.4 Closely Related Models

Several representative works in graph convolution are worth discussing w.r.t. their connections to ours.

2.4.1 Spectral Convolution Methods

The Spectral Network [14] is derived from the graph signal processing work [96], which generalizes Fourier analysis for its use in the graph domain as:

$$\begin{aligned} y_{iq} &= \sum_{j \in G(i)} \sum_{p=1}^{P} w_{ij}^{(pq)} x_{jp} \\ where \quad \boldsymbol{W}^{(pq)} &= \Phi \Lambda^{(pq)} \Phi^T \end{aligned} \tag{2.7}$$

where $\Phi \in \mathbb{R}^{n \times n}$ are the eigenvectors of G's graph Laplacian matrix, and Λ are learnable non-parametric filters from the training data. The Spectral Network can be matched with the full convolution (eq.(2.3)), but with the different filter subspace, in other words, with different basic filters. Limitations of Spectral Networks include its high computation cost due to eigen-decomposition of the graph Laplacian, the lack of spatial locality and the large number of parameters which grows linearly over the graph size.

These limitations are partially addressed in the Chebyshev Networks [29] (ChebyNet), which approximates the non-parametric filters as:

$$y_{iq} = \sum_{j \in G(i)} \sum_{k=1}^{K} T_k(L)_{ij} z_{iq}^{(k)}, \quad \boldsymbol{z}_i^{(k)} = \boldsymbol{x}_j \boldsymbol{U}^{(k)} \tag{2.8}$$

where $T_k(L)$ is the k-th order Chebyshev polynomial term. While being faster than Spectral Networks, ChebyNet suffers from insufficient expressiveness, similar to the limitations of GC. The expressiveness of ChebyNet can be improved by enlarging K, which requires a much larger number of parameters and eventually converges to Spectral Networks.

2.4.2 Geometric Convolution Methods

Several geometric convolution methods [7, 79, 82] are proposed for manifold structured data, among which MoNet [82] is the state-of-the-art.

The updating formula for MoNet can be written as

$$\begin{aligned} y_{iq} &= \sum_{j \in G(i)} \sum_{k=1}^{K} w_k(v(i,j)) z_{jq}^{(k)}, \quad \boldsymbol{z}_j^{(k)} = \boldsymbol{x}_j \boldsymbol{U}^{(k)} \\ w_k(v) &= \exp\left(-\frac{1}{2}(v - \mu_k)^T \Sigma_k^{-1} (v - \mu_k)\right) \end{aligned} \tag{2.9}$$

where $v(i,j)$ is the embedding of a node pair similar to Δ_{ij} in our model, and μ_k, Σ_k are both learnable parameters. MoNet can be viewed as an extension of ChebyNet where the graph filters are learned from data instead of being fully determined by a given graph. However, a graph filter in MoNet is still applied across all channels. In order to have k filters in MoNet, it needs to learn k different channel filters $\boldsymbol{U}$. Then the total number of model parameters will grow linearly with k. While DSGC only learns a channel filter $\boldsymbol{U}$ for one layer. By taking advantage

of that, the number of parameters in DSGC would not be significantly growing with k as MoNet. Similar to it, the recently proposed Graph Attention Network (GAT) [111] is also required to learn multiple channel filters in order to model multiple filters in one layer, which would lead to a larger number of the model parameters. Furthermore, in the setting that nodes in graph have geometric information, GAT can be viewed as a MoNet extension with the filter normalization trick. We empirically found that the extension exhibits obvious improvement over the original MoNet. In the following parts, we denote it as MoNet with GAT.

Message Passing Neural Networks (MPNN) [37, 95] are developed for modeling information propagation over graphs, specialized for the prediction tasks in quantum chemistry. Similar to DSGC, MPNN utilizes a neural network to predict the filter weights for the convolution operations. The key difference is that while DSGC allows channel-specific graph convolution filters (hence allowing a variety of diffusion patterns over the graph), MPNN learns only a single graph filter function for all channels in a layer. So MPNN can be viewed as a special case of DSGC with $C = 1$ in eq. (2.6). In our experiments, our method consistently outperforms MPNN across tasks in a variety of domains.

2.5 Experiments

2.5.1 Experimental Design

Our experiments for evaluating the proposed DSGC approach consists of three parts. Firstly, we evaluate DSGC on a popular image classification dataset (Sec. 2.5.2). The purpose is to confirm the strong performance of DSGC in handling grid-based convolution although it is designed for more general graph structures. Secondly, we compare DSGC and strong methods in the tasks of time series forecasting (Sec. 2.5.3) and text categorization (Sec. 2.5.4) where grid-based convolutions are invalid but graph convolution would have advantages instead as they are designed for more flexible graph structures. Thirdly, we examine the flexibility and effectiveness of using DSGC as a building block (module) in multiple well-known neural network architectures, including Inception [104], DenseNet framework [49] and etc.

For controlled experiments, all the graph convolution methods share the same empirical settings unless otherwise specified, including network structures, the dimension of latent factors, and hyper-parameter tunning process. The neural network used to model the spatial convolution filter ($w^{(q)}(\cdot)$) in eq. (2.5) is a two-layer MLP with 256 hidden dimensions and tanh activation function. We have conducted ablation tests with the two-layer MLP by changing the number of layers and activation function of each hidden layer, and by trying several weight sharing strategies. The results are very similar; the two-layer MLP provides a reasonable performance with the shortest running time. The algorithms are implemented in PyTorch; all the data and the code including baselines are made publicly accessible [2].

[2]Code: https://github.com/laiguokun/DSGC
Data: https://drive.google.com/drive/folders/0BweQMXBkrHAcSkpkejFsOXNId2s?usp=sharing

2.5.2 Evaluation on Image Classification

We conduct experiments on CIFAR10 and CIFAR100 [57], which are popular benchmark datasets in image classification. Both sets contain 60000 images with 32×32 pixels but CIFAR10 has 10 category labels and CIFAR100 has 100 category labels. Each image is typically treated as a 32×32 grid structure for standard image-based convolution. To enable the comparison on generic graphs, we create the modified versions of CIFAR10 and CIFAR100 respectively, by subsampling only 25% of the pixels from each graph. As illustrated in Figure 2.1, the subsampling result is irregularly scattered nodes for each image.

(a) (b) (c)

Figure 2.1: How to construct subsampled CIFAR datasets: (a) is an example image from CIFAR dataset. (b) is the subsampled pixels map. The blue points indicate which points are sampled. (c) is the image after sampling, where the black points are those not being sampled.

For all methods, we use the VGG-13 architecture [97] as the basic framework, and replace its convolution layers with different convolution modules. The experiment results are summarized in Table 2.1. The best performances among the graph-based neural networks are in bold. Firstly, we observe that Xception and CNN have the best results; this is not surprising because both methods use grid-based convolution which is naturally suitable for image recognition. Secondly, DSGC outperforms all the other graph-based convolution methods, and its performance is very close to that of the grid-based convolution methods. We also see that the models that learn multiple filters (DSGC and MoNet) have better performance than the models that only learn one global graph filter, such as MPNN and GCN. It demonstrates that it is necessary to enable multiple filters in this task. Furthermore, contributed by the depthwise separable convolution and graph sharing technique, our model can achieve a competitive performance without increasing the number of parameters as GCN, the one with the smallest number of parameters among graph convolution approaches. On the contrary, MoNet and ChebyNet have a relatively larger number of parameters in order to model multiple filters in one layer.

2.5.3 Evaluation on Time Series Forecasting

In time-series forecasting, we are usually interested in how to effectively utilize the geometric information about sensor networks. For example, how to incorporate the longitudes/latitudes of sensors w.r.t. temporal cloud movement is a challenge in spatiotemporal modeling for predicting the energy output of solar energy farms in the United States.

	Subsampled Images			Original Images		
Method	CIFAR10	CIFAR100	#params	CIFAR10	CIFAR100	#params
DCNN [3]	43.68%	76.65%	12M	55.56%	84.16%	50M
ChebyNet [29]	25.04%	49.44%	10M	12.99%	36.96%	19M
GCN [55]	26.78%	51.30%	5.6M	19.09%	41.64 %	9.8M
MoNet (with GAT) [82]	21.20%	47.87%	11M	8.34%	29.56%	20M
MPNN [37]	22.71%	49.03%	5.6M	11.01%	32.95%	9.9M
DSGC (ours)	**18.72**%	**44.33**%	5.7M	**7.31**%	**27.29**%	9.9M
CNN (VGG-13) [97]	18.03%	43.42%	18M	6.86%	26.86%	18M
CNN (Xception) [20]	17.07%	41.54%	3.1M	7.08%	26.84%	3.1M

Table 2.1: Test-set error rates on CIFAR10 and CIFAR100. DSGC has the best performance among the graph-based convolution method group (first six methods), and is comparable to the state-of-the-art grid-based convolution methods (VGG-13 and Xception) which are tailed for image classification.

We choose three publicly available benchmark datasets for this task:

- The U.S Historical Climatology Network (USHCN)[3] dataset, used in [4], contains daily climatological data from 1,218 meteorology sensors over the years from 1915 to 2000. The sequence length is 32,507. It includes five subsets, and each has a climate variable: (1) maximum temperature, (2) minimum temperature, (3) precipitation, (4) snowfall and (5) snow depth. We use daily maximum temperature data and precipitation data, and refer them as the **TMAX** and **PRCP** sets, respectively.
- The solar power production records in the year of 2006 has the data with the production rate of every 10 minutes from 1,082 solar power stations in the west of the U.S. The sequence length is 52,560. We refer this set of data as **Solar**.

All the datasets have been split into the training set (60%), the validation set (20%) and the test set (20%) in chronological order.

Table 2.2 summarizes the evaluation results of all the methods, where the performance is measured using the Root Square Mean Error (RMSE). The best result on each dataset is highlighted in boldface. Overall, our proposed method (DSGC) has the best performance on all the datasets, demonstrating its strength in capturing informative local propagation patterns both temporally and spatially.

2.5.4 Evaluation on Document Categorization

Following the experiment in [29], we test DSGC and other baselines in the text categorization application, and use the 20NEWS dataset ([51]) for our experiments. 20NEWS consists of 18,845 text documents with 20 topic labels. Individual words in the document vocabulary are the nodes in the graph for convolution. Each node has its word embedding vector generated by Word2Vec algorithm ([81]) on the same corpus. Following the experiment settings in [29] we select the top 1000 most frequent words as the nodes. Table 2.3 summarizes the results of the graph convo-

[3] `http://cdiac.ornl.gov/epubs/ndp/ushcn/daily_doc.html`

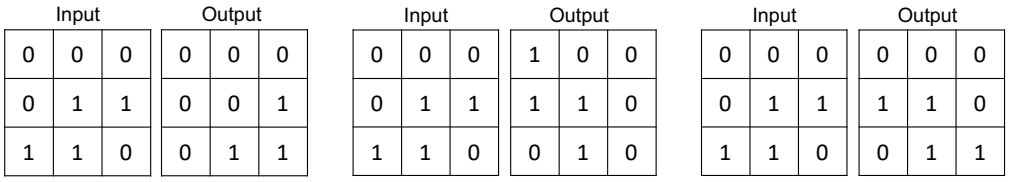
Method	TMAX	PRCP	Solar
DCNN	6.5188	29.0424	0.02652
ChebyNet	5.5823	27.1298	0.02531
GCN	5.4671	27.1172	0.02512
MoNet (with GAT)	5.8263	26.8076	0.02564
MPNN	5.3331	26.4766	0.02496
DSGC (ours)	**5.1438**($\pm$0.0498)	**25.8228**($\pm$0.249)	**0.02453**($\pm$0.00022)

Table 2.2: Test-set performance for graph convolution methods on time series prediction tasks measuring in RMSE. For our method, we report the standard deviation of the performance by running the model with 10 random seeds.

Method	Accuracy
Linear SVM†	65.90%
Multinomial NB†	68.51%
Softmax†	66.28%
FC2500†	64.64%
FC2500-FC500†	65.76%
DCNN	70.35%
ChebyNet	70.92%
GCN	71.01%
MoNet (with GAT)	70.60%
MPNN	71.58%
DSGC (ours)	**72.11**%($\pm$0.285)

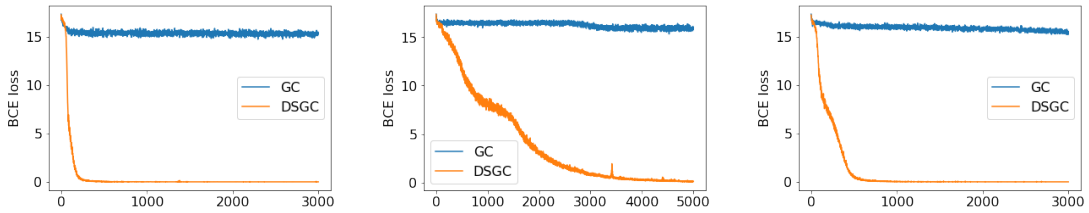
Table 2.3: Accuracy on the validation set of 20NEWS. Results marked with † come from [29]. The number in the parenthesis is the standard deviation.

lution methods plus three popular traditional classifiers (Linear SVM, Multivariate Naive Bayes and Softmax). DSGC has the best result on this dataset. Notice that the traditional classifiers are trained and tested with the feature set of the top 1000 words, which is the same setting as in the graph convolution models. If all words are used, traditional classifiers would have higher performance.



(a) Shift (b) Rotation (c) Left-Right Flip

Figure 2.2: Toy examples visualizing 3 graph operations: Shift, Rotation and Left-Right Flipping.



(a) Shift (b) Rotation (c) Left-Right Flip

Figure 2.3: Binary Cross Entropy (BCE) loss vs. training epoch for DSGC and GC on simulated data

2.5.5 Simulation-based Analysis

To compare the representation power of DSGC and GC (given comparable number of parameters), we generated simulated data for the following tasks:

- **Shift**: Shift the input graph 1 step to the right.
- **Rotation**: Rotate the input graph by 90 degree.
- **Flip**: Left-Right flip the input graph.

Toy examples for those tasks are visualized in Figure 2.2.

We use identical hyper-parameter settings for DSGC- and GC-based architectures in all those tasks, including the number of layers (fixed to be 3) and the activation function. The total numbers of model parameters of DSGC and GC are roughly equal.

The training loss curves for GC and DSGC are shown in Figure 2.3, where DSGC clearly outperforms GC. This is because GC treats the surrounding nodes for any given node as exchangeable [12], while DSGC learns to distinguish the neighbors conditioned on the tasks. The ability of learning asymmetric relationships is important in real applications such as wind power forecasting because of the directional influences (trends and anisotropy) from one geographic location to its adjacent locations.

2.5.6 DSGC with Multiple Neural Architectures

As the proposed DSGC is mathematically compatible with the traditional convolution method by performing channel special filter learning within one-layer, naturally, we can directly replace the convolution layers of general deep convolution frameworks with the DSGC modules while keeping a similar performance without modifying the framework structure. We examine DSGC with the following frameworks which are popular in recent years for standard convolution over images: (1) Inception [104], (2) DenseNet framework [49] and (3) Squeeze-and-Excitation block [48]. The results are presented in Table 2.4. Clearly, combined with the advantageous architectures, the performance of DSGC in image classification can be further improved (DSGC-DenseNet over DSGC-VGG-13). It demonstrates that the DSGC can easily enjoy the benefits of framework design for free from the traditional 2d-convolution network community.

	Subsampled Images			Original Images		
Method	CIFAR10	CIFAR100	#params	CIFAR10	CIFAR100	#params
DSGC-VGG-13	18.72%	44.33%	5.7M	7.31%	27.29%	9.9M
DSGC-INCEPTION	18.27%	43.41%	9.9M	**6.44**%	28.55%	12M
DSGC-DenseNet	17.17%	43.34%	2.7M	7.14%	**26.50**%	2.9M
DSGC-SE	18.71%	44.15%	6.1M	7.00%	27.26%	10M
VGG-13	18.03%	43.42%	18M	6.86%	26.86%	18M
Xception	**17.07**%	**41.54**%	3.1M	7.08%	26.84%	3.1M

Table 2.4: Test-set error rates of DSGC-based architectures (first group) and CNNs (second group)

2.5.7 Training Time Comparison

In Table 2.5, we report the mean training time per epoch for GCN, DSGC and MoNet. The proposed DSGC computes the convolution weight for each edge in the graph, which requires more computation resources compared to GCN. However, we always perform the graph convolution on a sparse k-nearest neighbor graph, where the number of edges grows only linearly with the node size. Therefore the training is fairly efficient. Notably, DSGC consistently performs better than all graph convolution methods with around 1.5x-4x running time compared to the fastest graph convolution framework (GCN). And MoNet, which also learns multiple filters in a layer but without the channel separation technique applied for DSGC, would be 1x slower than the proposed DSGC method.

Method	CIFAR10	TMAX	20news
GCN	1.75	0.465	0.207
MoNet	6.87	2.81	0.550
DSGC (ours)	3.81	1.73	0.280

Table 2.5: Training time per epoch for GCN, MoNet and DSGC in three benchmark datasets, meausred in minutes.

2.6 Implementation Details

2.6.1 Implementation Details of CIFAR Experiment

In section 2.5.2 and 2.5.6, we conduct the experiment on the CIFAR10 and CIFAR100 datasets. We will introduce the architecture settings for the DSGC and baseline models. Table 2.6 illustrates the basic architecture used in the experiment. In the DSGC-VGG13 and DSGC-DenseNet models, the k-conv refers to the spatial convolution (Eq.2.5) with k-nearest neighbors as the neighbor setting. So the 1-conv is the same as the 1×1 conv, which is doing linear transformation on channels. The hidden dimensions of VGG13 and DSGC-VGG13 are set as $\{256, 512, 512, 512\}$ and $\{256, 512, 512, 1024\}$. The growth rate of DSGC-DenseNet is 32. And the baseline graph

Layers	VGG13	DSGC-VGG13	DSGC-DenseNet
Convolution	$\begin{bmatrix} 3 \times 3 \text{ conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 6$
Transition			1-conv
Pooling	2×2 max-pooling	4 max-pooling	
Convolution	$\begin{bmatrix} 3 \times 3 \text{ conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 12$
Transition			1-conv
Pooling	2×2 max-pooling	4 max-pooling	
Convolution	$\begin{bmatrix} 3 \times 3 \text{ conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 24$
Transition			1-conv
Pooling	2×2 max-pooling	4 max-pooling	
Convolution	$\begin{bmatrix} 3 \times 3 \text{ conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 16$
Transition			1-conv
Pooling	2×2 max-pooling	4 max-pooling	
Convolution	$\begin{bmatrix} 3 \times 3 \text{ conv} \end{bmatrix} \times 2$	$\begin{bmatrix} \text{9-conv} \end{bmatrix} \times 2$	
Pooling	2×2 max-pooling	4 max-pooling	
Classifier	512D fully-connected, softmax		

Table 2.6: Neural Network architecture for CIFAR datasets. Please see the text for more details.

and geometric convolution methods use the identical architecture as DSGC-VGG13. For the subsampled CIFAR experiment, We eliminate the first convolution, transition and pooling layer, and change the spatial convolution from 9-conv to {16-conv, 12-conv, 8-conv, 4-conv}. For the DSGC-SE, we follow the method described in [48] to add the SE block to DSGC-VGG13 architecture. We use the dropout scheme described in [49] for the DSGC-DenseNet model, and add the dropout layer after the pooling layer for VGG13 and DSGC-VGG13 models. For the DSGC-Inception model, we imitate the design of the Inception Network ([104]). The key idea is letting a convolution layer have different size of convolution filters. We use a simple example as our Inception module, which is illustrated in Figure 2.4.

For the CNN model, we still format the input signal in the matrix shape. The signals in invalid points are set as 0. Furthermore, to perform the fair comparison with standard CNN in the subsampled situation, we append a mask matrix as an additional channel for input signals to indicate whether the pixel is valid or not. For the ChebyNet, we set the polynomial order as $K = 3$.

The pooling layer is implemented by K-means clustering. The centroid of each clusters is regarded as the new node after pooling, and its hidden vector is the mean or max over the nodes in that cluster. Notice that, we only normalize the input signals to [0,1] and do not adopt any other data preprocessing or augmentation tricks.

For the $\triangle_{ij}$ used in DSGC and MoNet, we use a 5 dimension feature vector. We denote the coordinate of i-th node as (x_i, y_i), and $\triangle x_{ij} = x_i - x_j, \triangle y_{ij} = y_i - y_j, \triangle d_{ij} = \triangle x_{ij}^2 + \triangle y_{ij}^2$. Then $\triangle_{ij} = (sign(\triangle x_{ij}), |\triangle x_{ij}|, sign(\triangle y_{ij}), |\triangle y_{ij}|, \triangle d_{ij})$.

The same learning schedule is applied to all models. We use SGD to train the model for 400 epochs. The initial learning rate is 0.1, and is divided by 10 at 50% and 75% of the total number of training epochs.

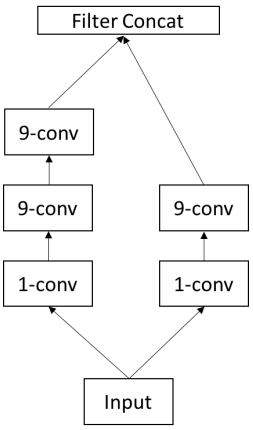


Figure 2.4: Inception Module

2.6.2 Implementation Details of Time Series Prediction

Firstly, we will give the formal definition of the time series forecasting, that is, spatiotemporal regression problem. We formulate the the spatiotemporal regression problem as a multivariate time series forecasting task with the sensors' location as the input. More formally, given a series of time series signals observed from sensors $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T\}$ where $\boldsymbol{y}_t \in \mathbb{R}^n$ and n are the number of sensors, and the locations of sensors $\boldsymbol{L} = \{\boldsymbol{l}_1, \boldsymbol{l}_2, \cdots, \boldsymbol{l}_n\}$ where $\boldsymbol{l}_i \in \mathbb{R}^2$ and indicates the coordinate of the sensor, the task is to predict a series of future signals in a rolling forecasting fashion. That being said, to predict $\boldsymbol{y}_{T+h}$ where h is the desirable horizon ahead of the current time stamp T, we assume $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T\}$ are available. Likewise, to predict the signal of the next time stamp $\boldsymbol{y}_{T+h+1}$, we assume $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \cdots, \boldsymbol{y}_T, \boldsymbol{y}_{T+1}\}$ are available. In this paper, we follow the setting of the autoregressive model. Define a window size p which is a hyper-parameter firstly. The model input at time stamp T is $\boldsymbol{X}_T = \{\boldsymbol{y}_{T-p+1}, \cdots, \boldsymbol{y}_T\} \in \mathbb{R}^{n \times p}$. In the experiments of this paper, the horizon is always set as 1.

Intuitively, different sensors may have node-level hidden features influencing its propagation patterns and final outputs. For each node, we let the model learn a node embedding vector and concatenate it with the input signals. The embedding size is tuned according to the validation set. By using this trick, each node has limited freedom to interface with its propagation patterns.

One thing readers may notice is that there are 10% data in USHCN dataset missing. To deal with that, we add an additional feature channel to indicate which point is missing. For the time series models, we tune the historical window p according to the validation set. For the rest of models, we set the window size $p = 18$ for Solar dataset and $p = 6$ for USHCN datasets. The network architecture used in this task is 7 convolution layers followed by a regression layer. The $\triangle_{ij}$ setting is the same as the previous one. We use the Adam optimizer [52] for this task, and train each model 200 epochs with learning rate 0.001.

Except for the graph convolution methods, we also add in traditional methods of time series forecasting for comparison, such as (1) Autoregressive model (AR) which predicts future signal using a window of historical data based on a linear assumption about temporal dependencies, (2) Vector autoregressive model (VAR) which extends AR to the multivariate version, namely, the input is the signals from all sensors in the history window, and (3) the LSTNet deep neural

network model [59] which combines the strengths of CNN, RNN and AR. None of those methods is capable of leveraging locational dependencies via graph convolution.

Method	TMAX	PRCP	Solar
AR	8.2354	30.3825	0.03195
VAR	17.9743	29.2597	0.03296
LSTNet	10.1973	29.0624	0.02865
DCNN	6.5188	29.0424	0.02652
ChebyNet	5.5823	27.1298	0.02531
GCN	5.4671	27.1172	0.02512
MoNet (with GAT)	5.8263	26.8076	0.02564
MPNN	5.3331	26.4766	0.02496
DSGC (ours)	**5.1438**(± 0.0498)	**25.8228**(± 0.249)	**0.02453**(± 0.00022)

Table 2.7: Test-set performance for graph convolution methods on time series prediction tasks measuring in RMSE. For our method, we report the standard deviation of the performance by running the model with 10 random seeds.

Table 3.5 summarizes the evaluation results of all the methods, where the performance is measured using the Root Square Mean Error (RMSE). The best result on each dataset is highlighted in boldface. The group of the first three methods does not leverage the spatial or locational information in data. The second group (graph-based convolution methods) consists of the neural network models which leverage the spatial information about sensor networks. The methods in the second group clearly outperform the methods in the first one, which does not explicitly model the spacial correlation within sensor networks.

2.6.3 Implementation Details of Document Categorization

The data preprocessing follows the experiment details in [29]. And the network architecture for all models is 5 convolution layers followed by two MLP layers as the classifier. After each convolution layer, a dropout layer is performed with dropout rate of 0.5. The nodes' coordinate is the word embedding, and the method to calculate $\triangle_{ij}$ is similar to the previous ones. The optimizer used in this task is the same as the CIFAR experiment.

2.7 Summary

This chapter presents a unified view of graph convolution and grid-based convolution methods, and proposes the novel DSGC approach that is applicable to non-grid spatial data. DSGC subsumes several existing graph convolution methods as special cases and is compatible to depthwise separable convolution for image classification by performing channel-special filters learning in data manifold. The proposed DSGC yields state-of-the-art performance on multi-domain benchmark datasets with a relatively small number of model parameters, reasonable computation cost, and is easy to be plugged in different neural network architectures. For future research

we plan to extend DSGC to a broader range of problems, including social network and citation graph analysis, where the spatial coordinates of the nodes (node embeddings) can be jointly learned along with the convolution filters, or defined by node embedding algorithm.

Chapter 3

Factorized Recurrent Neural Network

3.1 Motivation

As a fundamental problem in machine learning, probabilistic sequence modeling aims at capturing the sequential correlations in both short and long ranges. Among many possible model choices, deep auto-regressive models [40, 105] have become one of the most widely adopted solutions. Typically, a deep auto-regressive model factorizes the likelihood function of sequences in an auto-regressive manner, i.e., $p(\mathbf{x}) = \prod_{t=1}^{|\mathbf{x}|} p(x_t \mid \mathbf{x}_{<t})$. Then, a neural network (e.g. RNN) is employed to encode the conditional context $\mathbf{x}_{<t}$ into a compact hidden representation $h_t = f(\mathbf{x}_{<t})$, which is then used to define the output distribution $p(x_t \mid \mathbf{x}_{<t}) \triangleq p(x_t \mid h_t)$.

Despite the state-of-the-art (SOTA) performance in many domains [19, 26, 33, 88], the hidden representations of standard auto-regressive models are produced in a completely deterministic way. Hence, the stochastic aspects of the observed sequences can only be modeled by the output distribution, which however, usually has a simple parametric form such as a unimodal distribution or a finite mixture of unimodal distributions. A potential weakness of such simple forms is that they may not be sufficiently expressive for modeling real-world sequential data with complex stochastic dynamics.

Recently, many efforts have been made to enrich the expressive power of auto-regressive models by injecting stochastic latent variables into the computation of hidden states. Notably, relying on the variational auto-encoding (VAE) framework [53, 92], stochastic recurrent models (SRNN) have outperformed standard RNN-based auto-regressive models by a large margin in modeling raw sound-wave sequences [1, 6, 22, 34, 38, 62].

However, the success of stochastic latent variables does not necessarily generalize to other domains such as text and images. For instance, the authors [38] report that an SRNN trained by Z-Forcing lags behind a baseline RNN in language modeling. Similarly, for the density estimation of natural images, PixelCNN [87, 94, 108] consistently outperforms generative models with latent variables [31, 41, 42, 54, 91].

To better understand the discrepancy, we perform a re-examination on the role of stochastic variables in SRNN models. By carefully inspecting of the previous experiment settings for sound-wave density estimation, and systematically analyzing the properties of SRNN, we identify two potential causes of the performance gap between SRNN and RNN. Controlled experi-

ments are designed to test each hypothesis, where we find that previous evaluations impose an unnecessary restriction of fully factorized output distributions, which has led to an unfair comparison between SRNN and RNN. Specifically, under the factorized parameterization, SRNN can still implicitly leverage the intra-step correlation, i.e., the simultaneity [8], while the RNN baselines are prohibited to do so. Meanwhile, we also observe that the posterior learned by SRNN can get outperformed by a simple hand-crafted posterior, raising serious doubt about the general effectiveness of injecting latent variables.

To provide a fair comparison, we propose an evaluation setting where both the SRNN and RNN can utilize an auto-regressive output distribution to model the intra-step correlation explicitly. Under the new setting, we re-evaluate SRNN and RNN on a diverse collection of sequential data, including human speech, MIDI music, handwriting trajectory and frame-permuted speech. Empirically, we find that sequential models with continuous latent variables fail to offer any practical benefits, despite their widely believed theoretical superiority. On the contrary, explicitly capturing the intra-step correlation with an auto-regressive output distribution consistently performs better, substantially improving the SOTA performances in modeling speech signals. Overall, these observations show that the previously reported performance "advantage" of SRNN is merely the result of a long-existing experiment bias of using factorized output distributions.

3.2 Background

In this section, we briefly review SRNN and RNN for probabilistic sequence modeling. Throughout the chapter, we will use bold font $\mathbf{x}$ to denote a sequence, $\mathbf{x}_{<t}$ and $\mathbf{x}_{\leq t}$ to indicate the subsequence of first $t-1$ and t elements respectively, and x_t to represent the t-th element. Note that x_t can either be a scalar or a multivariate vector. In the latter case, $x_{t,i}$ denotes the i-th element of the vector x_t.

Given a set of sequences $\mathcal{D} = \left\{\mathbf{x}^1, \mathbf{x}^2, \cdots, \mathbf{x}^{|\mathcal{D}|}\right\}$, we are interested in building a density estimation model for sequences. A widely adapted solution is to employ an auto-regressive model powered by a neural network, and utilize MLE to perform the training:

$$\max_{\theta} \mathcal{L}_{\mathcal{D}} = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}} \sum_{t=1}^{T_x} \log p_\theta(x_t \mid \mathbf{x}_{<t}), \tag{3.1}$$

where T_x is the length of the sequence $\mathbf{x}$. More concretely, the conditional distribution $p_\theta(x_t \mid \mathbf{x}_{<t})$ is usually jointly modeled by two sub-modules:

- The pre-defined distribution family of the output distribution $p_\theta(x_t \mid \mathbf{x}_{<t})$, such as a Gaussian, Categorical or Gaussian Mixture;
- The sequence model f_θ, which encodes the contextual sequence $\mathbf{x}_{<t}$ into a compact hidden vector h_t;

Under this general framework, RNN and SRNN can be seen as two different instantiations of the sequence model. As we have discussed in Section 3.1, the computation inside RNN is fully deterministic.

To improve the model expressiveness, SRNN takes an alternative route and incorporates *continuous* latent variables into the sequence model. Typically, SRNN associates the observed data

sequence $\mathbf{x}$ with a sequence of latent variables $\mathbf{z} = [z_1, \ldots, z_{T_x}]$, one for each step. With latent variables, the internal dynamics of the sequence model is not deterministic any more, offering a theoretical possibility to capture more complex stochastic patterns. However, the improved capacity comes with a computational burden — the log-likelihood is generally intractable due to the integral:

$$\mathcal{L}_{\mathcal{D}}^{\text{SRNN}} = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}} \log \int p_\theta(\mathbf{x}, \mathbf{z}) d\mathbf{z}.$$

Hence, standard MLE training cannot be performed.

To handle the intractability, SRNN utilizes the VAE framework and maximizes the evidence lower bound (ELBO) of the log-likelihood (3.1) for training:

$$\max_{\theta,\phi} \mathcal{F}_{\mathcal{D}} = \mathop{\mathbb{E}}_{\mathbf{x}\sim\mathcal{D}} \mathop{\mathbb{E}}_{q_\phi(\mathbf{z}|\mathbf{x})} \sum_{t=1}^{T_x} \log \frac{p_\theta(x_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}) p_\theta(z_t | \mathbf{z}_{<t}, \mathbf{x}_{<t})}{q_\phi(z_t \mid \mathbf{z}_{<t}, \mathbf{x})} \leq \mathcal{L}_{\mathcal{D}}^{\text{SRNN}}, \tag{3.2}$$

where $q_\phi(\mathbf{z} \mid \mathbf{x})$ is the approximate posterior distribution modeled by an encoder network. Computationally, several SRNN variants have been proposed [6, 22, 34, 38], mostly differing in how the generative distribution $p_\theta(\mathbf{x}, \mathbf{z})$ and the variational posterior $q_\phi(\mathbf{z} \mid \mathbf{x})$ are parameterized. In this work, we follow the parameterization and optimization in Z-forcing SRNN method [38], which is the one with the best performance.

3.3 Revisiting SRNN for Speech Modeling

3.3.1 Previous Setting for Speech Density Estimation

To compare SRNN and RNN, previous studies largely rely on the density estimation of sound-wave sequences. Usually, a sound-wave dataset consists of a collection of audio sequences with a sample rate of 16Hz, where each frame (element) of the sequence is a scalar in $[-1, 1]$, representing the normalized amplitude of the sound. Instead of treating each frame as a single step, the authors [22] propose a multi-frame setting, where every 200 consecutive frames are taken as a single step. Effectively, the data can be viewed as a sequence of 200-dimensional real-valued vectors, i.e., $x_t \in \mathbb{R}^L$ with $L = 200$. During training, every $T = 40$ steps (8,000 frames) are taken as an i.i.d. sequence to form the training set.

Under this data format, notice that the output distributions $p_\theta(x_t \mid \mathbf{x}_{<t})$ and $p_\theta(x_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t})$ now correspond to an L-dimensional random vector x_t. Therefore, how to parameterize this multivariate distribution can largely influence empirical performance. That said, recent approaches [34, 38] have all followed [22] to employ a fully factorized parametric form which ignores the inner dependency:

$$p_\theta(x_t \mid \mathbf{x}) \approx \prod_{i=1}^{L} p_\theta(x_{t,i} \mid \mathbf{x}_{<t}), \tag{3.3}$$

$$p_\theta(x_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}) \approx \prod_{i=1}^{L} p_\theta(x_{t,i} \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}). \tag{3.4}$$

Here, we have used the $\approx$ to emphasize this choice effectively poses an independent assumption. Despite this convenience, note that the restriction of a fully factorized form is not necessary at all. Nevertheless, we will refer to the models in Eqn. (3.3) and Eqn. (3.4), respectively, as factorized RNN (F-RNN) and factorized SRNN (F-SRNN) in the following.

To provide a baseline for further discussion, we replicate the experiments under the setting introduced above and evaluate them on three speech datasets, namely TIMIT, VCTK, and Blizzard. Following the previous work [22], we choose a Gaussian mixture to model the per-frame distribution $p_\theta(x_{t,i} \mid \mathbf{x}_{<t})$ of F-RNN, which enables a basic multi-modality.

We report the averaged test log-likelihood in Table 3.1. For consistency with previous results in the literature, the results of TIMIT and Blizzard are based on *sequence-level* average, while the result of VCTK is *frame-level* average. As we can see, similar to previous observations, F-SRNN outperforms F-RNN on all three datasets by a dramatic margin.

Models	**TIMIT**	**VCTK**	**Blizzard**
F-RNN	32,745	0.786	7,610
F-SRNN	**69,296**	**2.383**	**15,258**

Table 3.1: Performance comparison on three benchmark datasets.

3.3.2 Decomposing the Advantages of Factorized SRNN

To understand why the F-SRNN outperforms F-RNN by such a large margin, it is helpful to examine the effective output distribution $p_\theta(x_t \mid \mathbf{x}_{<t})$ of F-SRNN after marginalizing out the latent variables:

$$p_\theta(x_t \,|\, \mathbf{x}_{<t}) = \int p_\theta(\mathbf{z}_{\leq t} \,|\, \mathbf{x}_{<t}) \prod_{i=1}^{L} p_\theta(x_{t,i} \,|\, \mathbf{z}_{\leq t}, \mathbf{x}_{<t}) d\mathbf{z}_{\leq t}. \tag{3.5}$$

From this particular form, we can see two potential causes of the performance gap between F-SRNN and F-RNN in the multi-frame setting:

- **Advantage under High Volatility**: By incorporating the continuous latent variable, the distribution $p_\theta(x_t \mid \mathbf{x}_{<t})$ of F-SRNN essentially forms an infinite mixture of simpler distributions (see first line of Eqn. (3.5)). As a result, the distribution is significantly more expressive and flexible, and it is believed to be particularly suitable for modeling high-entropy sequential dynamics [22].

  The multi-frame setting introduced above well matches this description. Concretely, since the model is required to predict the next L frames all together in this setting, the long prediction horizon will naturally involve a higher uncertainty. Therefore, the high volatility of the multi-frame setting may provide a perfect scenario for SRNN to exhibit its theoretical advantage in expressiveness.
- **Utilizing the Intra-Step Correlation**: From Eqn. (3.5), notice that the distribution $p_\theta(x_t \mid \mathbf{x}_{<t})$ after marginalization is generally not factorized any more, due to the coupling with $\mathbf{z}$. In contrast, recall the same distribution of the F-RNN (Eqn. (3.3)) is fully factorized $p_\theta(x_t \mid \mathbf{x}_{<t}) = \prod_{i=1}^{L} p_\theta(x_{t,i} \mid \mathbf{x}_{<t})$. Therefore, in theory, a factorized SRNN could still model the

correlation among the L frames within each step, if properly trained, while the factorized RNN has no means to do so at all. Thus, SRNN may also benefit from this difference.

While both advantages could have jointly led to the performance gap in Table 3.1, the implications are totally different. The first advantage under high volatility is a unique property of latent-variable models that other generative models without latent variables can hardly to obtain. Therefore, if this property significantly contributes to the superior performance of F-SRNN over F-RNN, it suggests more general effectiveness of incorporating stochastic latent variables.

Quite the contrary, being able to utilize the intra-step correlation is more like an *unfair* benefit to SRNN, since it is the unnecessary restriction of fully factorized output distributions in previous experimental design that prevents RNNs from modeling the correlation. In practice, one can easily enable RNNs to do so by employing a non-factorized output distribution. In this case, it remains unclear whether this particular advantage will sustain. Motivated by the distinct implications, in the sequel, we will try to figure out how much each of the two hypotheses above actually contributes to the performance gap.

3.3.3 Advantage under High Volatility

In order to test the advantage of F-SRNN in modeling high-volatile data in isolation, the idea is to construct a sequential dataset where each step consists of a single frame (i.e., a uni-variate variable), while there exists high volatility between every two consecutive steps.

Concretely, for each sequence $\mathbf{x} \in \mathcal{D}$, we create a sub-sequence by selecting one frame from every M consecutive frames, i.e., $\hat{\mathbf{x}} = [x_1, x_{M+1}, x_{2M+1}, \ldots]$ with $x_t \in \mathbb{R}$. Intuitively, a larger *stride* M will lead to a longer horizon between two selected frames and hence a higher uncertainty. Moreover, since each step corresponds to a single scalar, the second advantage (i.e., the potential confounding factor) automatically disappears.

Following this idea, from the original datasets, we derive the stride-TIMIT, stride-VCTK and stride-Blizzard with different stride values M, and evaluate the RNN and SRNN on each of them. Again, we report the sequence- or frame-average test likelihood in Table 3.2.

	Stride = 50			**Stride = 200**		
Model	TIMIT	VCTK	Blizzard	TIMIT	VCTK	Blizzard
RNN	**20,655**	**0.668**	**4,607**	**4,124**	**0.177**	**-320**
SRNN	14,469	0.605	3,603	-1,137	0.0187	-1,231

Table 3.2: Performance comparison on high-volatility datasets.

Surprisingly, RNN consistently achieves a better performance than SRNN in this setting. It suggests the theoretically better expressiveness of SRNN does not help that much in high-volatility scenarios. Hence, this potential advantage does not really contribute to the performance gap observed in Table 3.1.

3.3.4 Utilizing the Intra-Step Correlation

After ruling out the first hypothesis, it becomes more likely that being able to utilize the intra-step correlation actually leads to the superior performance of F-SRNN. However, despite the

non-factorized form in Eqn. (3.5), it is still not clear how F-SRNN computationally captures the correlation in practice. Here, we provide a particular possibility.

Recall that in ELBO function of SRNN method (Eqn. (3.2)), the vector x_t, we hope to reconstruct at step t, is included in the conditional input to the posterior $q_\phi(z_t \mid \mathbf{z}_{<t}, \mathbf{x})$. With this computational structure, the encoder could theoretically leak a *subset* of the vector x_t into the latent variable z_t, and leverage the leaked subset to predict (reconstruct) the rest elements in x_t. Intuitively, the procedure of using the leaked subset to predict the remained subset is essentially exploiting the dependency between the two subsets, or in other words, the correlation within x_t.

Proposition 1. *Given a vector x_t, we split its elements into two arbitrary disjoint subsets, the leaked subset x_t^a and its complement $x_t^b = x_t \backslash x_t^a$. Assume that the latent variables and leaked subset have the same dimensionality, $|z_t| = |x_t^a|$. Define the posterior distribution as a delta function:*

$$q_\phi(z_t \mid \mathbf{z}_{<t}, \mathbf{x}) = \delta_{z_t = x_t^a} = \begin{cases} \infty, & \text{if } z_t = x_t^a \\ 0, & \text{otherwise} \end{cases}, \tag{3.6}$$

We further assume $p_\theta(x_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}) \approx p_\theta(x_t \mid z_t, \mathbf{x}_{<t})$. The ELBO function (Eqn. (3.2)) would reduce to a special case of auto-regressive factorization:

$$\max_\theta \mathcal{L}_\mathcal{D} = \mathop{\mathbb{E}}_{\mathbf{x} \sim \mathcal{D}} \sum_{t=1}^{T_x} \log p_\theta(x_t^a \mid \mathbf{x}_{<t}) + \log p_\theta(x_t^b \mid x_t^a, \mathbf{x}_{<t}). \tag{3.7}$$

Now, the second term in Eqn. (3.7) is conditioned on the leaked subset of x_t^a to predict x_t^b, which is exactly utilizing the correlation between the two subsets. In other words, with a proper posterior, F-SRNN can recover a certain auto-regressive parameterization, making it possible to utilize the intra-step correlation, even with a fully factorized output distribution.

Although the analysis and construction above provide a theoretical possibility, we still lack concrete evidence to support the hypothesis that F-SRNN has significantly benefited from modeling the intra-step correlation. While it is difficult to verify this hypothesis in general, we can parameterize an RNN according to Eqn. (3.7), which is equivalent to an F-SRNN with a delta posterior. Therefore, by measuring the performance of this special RNN, we can get a *conservative* estimate of how much modeling the intra-step correlation can contribute to the performance of F-SRNN.

To finish the special RNN idea, we still need to specify how x_t is split into x_t^a and x_t^b. Here, we consider two methods with different intuitions:

- **Interleaving**: The first method takes one out of every U elements to construct $x_t^a = \{x_{t,1}, x_{t,U+1}, x_{t,2U+1}, \ldots\}$. Essentially, this method interleaves the two subsets x_t^a and x_t^b. In the extreme case of $U = 2$, x_t^a includes the odd elements of x_t and x_t^b the even ones. Hence, when predicting an even element $x_{t,2k} \in x_t^b$, the output distribution is conditioned on both the elements to the left $x_{t,2k-1}$ and to the right $x_{t,2k+1}$, making the problem much easier.
- **Random**: The second method simply uniformly selects V random elements from x_t to form x_t^a, and leaves the rest for x_t^b. Intuitively, this can be viewed as an informal "lower bound" of performance gain through modeling the intra-step correlation.

Models	TIMIT	VCTK	Blizzard
F-RNN	32,745	0.786	7,610
F-SRNN	69,296	**2.383**	15,258
δ-RNN ($U = 2$)	70,900	2.027	**15,306**
δ-RNN ($U = 3$)	**72,067**	2.262	15,284
δ-RNN ($V = 50$)	66,122	2.199	14,389
δ-RNN ($V = 75$)	66,453	2.120	14,585

Table 3.3: Performance comparison between δ-RNN and F-SRNN. Note that a smaller U corresponds to leaking more elements.

Since the parametric form Eqn. (3.7) is derived from a delta posterior, we will refer to the special RNN model as δ-RNN. Based on the two split methods, we train δ-RNN on TIMIT, VCTK and Blizzard with different values of U and V. The results are summarized in Table 3.3. As we can see, when the interleaving split scheme is used, δ-RNN significantly improves upon F-RNN and becomes very competitive with F-SRNN. Specifically, on TIMIT and Blizzard, δ-RNN can even outperform F-SRNN in certain cases. More surprisingly, the δ-RNN with the random-copy scheme can also achieve a performance that is very close to that of F-SRNN, especially compared to F-RNN.

Recall that δ-RNN is equivalent to employing a manually designed delta posterior that can only copy but never compresses (auto-encodes) the information in x_t. As a result, compared to a posterior that can learn to compress information, the delta posterior will involve a higher KL cost when leaking information through the posterior. Furthermore, the correlation between historical latent variables and outputs is ignored. It would decrease the model capacity of δ-RNN. Despite these disadvantages, δ-RNN is still able to match or even surpasses the performance of F-SRNN, suggesting the learned posterior in F-SRNN is far from satisfying. Quite contrary to that, the limited performance gap between F-SRNN and the random copy baseline raises a serious concern about the effectiveness of current variational inference techniques.

Nevertheless, putting the analysis and empirical evidence together, we can conclude that the performance advantage of F-SRNN in the multi-frame setting can be entirely attributed to the second cause. That is, under the factorized constraint in previous experiments, F-SRNN can still implicitly leverage the intra-step correlation, while F-RNN is prohibited to do so. However, as we have discussed earlier in Section 3.3.2, this is essentially an unfair comparison. More importantly, the claimed superiority of SRNN over RNN may be misleading, as it is unclear whether performance advantage of SRNN will sustain or not when a non-factorized output distribution is employed to capture the intra-step correlation explicitly.

As far as we know, no previous work has carefully compared the performance of SRNN and RNN when non-factorized output distribution is allowed. On the other hand, as shown in Table 3.3, by modeling the multivariate simultaneity in the simplest way, δ-RNN can achieve dramatic performance improvement. Motivated by the huge potential as well as the lack of a systematic study, we will next include non-factorized output distributions in our consideration, and properly re-evaluate SRNN and RNN for multivariate sequence modeling.

3.4 Proper Multivariate Sequence Modeling with or without Latent Variables

3.4.1 Avoiding the Implicit Data Bias

In this section, we aim to eliminate any experimental bias and provide a proper evaluation of SRNN and RNN for multivariate sequence modeling. Apart from the "model bias" of employing fully factorized output distributions we have discussed, another possible source of bias is actually the experimental data. For example, as we discussed in Section 3.3.1, the multi-frame speech sequences are constructed by reshaping L consecutive real-valued frames into L-dimensional vectors. Consequently, elements within each step x_t are simply temporally correlated with a natural order, which would favor a model that recurrently process each element from $x_{t,1}$ to $x_{t,L}$ with parameter sharing.

Thus, to avoid such "data bias", besides speech sequences, we additionally consider three more types of multivariate sequences with different patterns of intra-step correlation, they are MIDI sound sequence data (including Muse and Nottingham datasets), handwriting trajectory data (IAM-OnDB) and the Perm-TIMIT dataset. The Perm-TIMIT is a variant of multivariate TIMIT dataset. It permutes the elements within each time step, which is designed to remove the temporal bias.

3.4.2 Modeling Simultaneity with Auto-Regressive Decomposition

With proper datasets, we now consider how to construct a family of non-factorized distributions that (1) can be easily integrated into RNN and SRNN as the output distribution, and (2) are reasonably expressive for modeling multivariate correlations. Among many possible choices, the most straightforward choice would be the auto-regressive parameterization. Compared to other options such as the normalizing flow or Markov Random Field (e.g. RBM), the auto-regressive structure is conceptually simpler and can be applied to both discrete and continuous data with full tractability. In light of these benefits, we choose to follow this simple idea, and decompose the output distribution of the RNN and SRNN, respectively, as

$$p_\theta(x_t \mid \mathbf{x}_{<t}) = \prod_{i=1}^{L} p_\theta(x_{t,i} \mid \mathbf{x}_{<t}, x_{t,<i}), \tag{3.8}$$

$$p_\theta(x_t \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}) = \prod_{i=1}^{L} p_\theta(x_{t,i} \mid \mathbf{z}_{\leq t}, \mathbf{x}_{<t}, x_{t,<i}). \tag{3.9}$$

Notice that although we use the natural decomposition order from smallest index to largest one, this particular order is generally *not* optimal for modeling multivariate distributions. A better choice could be adapting the orderless training previously explored in literature [105]. But for simplicity, we will stick to this simple approach.

Given the auto-regressive decomposition, a natural neural instantiation would be a recurrent *hierarchical model* that utilizes a two-level architecture to process the sequence:

Models	TIMIT	VCTK	Blizzard	Muse	Nottingham	IAM-OnDB	Perm-TIMIT
VRNN†	28,982	-	9,392	-	-	1384	-
SRNN†	60,550	-	11,991	-6.28	-2.94	-	-
Z-Forcing†	70,469	-	15,430	-	-	-	-
SWaveNet†‡	72,463	-	15,708	-	-	1301	-
STCN†‡	77,438	-	17,670	-	-	**1796**	-
F-RNN	32,745	0.786	7,610	-6.991	-3.400	1397	25,679
F-SRNN	69,296	2.383	15,258	-6.438	-2.811	1402	67,613
δ-RNN-random	66,453	2.199	14,585	-6.252	-2.834	N/A	61,103
RNN-flat	**117,721***	**3.2173***	**22,714***	-5.251	-2.180	N/A	15,763
SRNN-flat	109,284	3.2062	22,290	-5.616	-2.324	N/A	14,278
RNN-hier	109,641	3.1822	21,950	**-5.161**	**-2.028**	1440	**95,161**
SRNN-hier	107,912	3.1423	21,845	-5.483	-2.065	1395	94,402

Table 3.4: Performance comparison on a diverse set of datasets. The models with † indicate that the performances are directly copied from previous publications. Numbers with * indicate the state-of-the-art performances. N/A suggests the model is not application on the dataset. The models with ‡ have other architectures than recurrent neural network as the backbone.

- Firstly, a high-level RNN or SRNN is employed to encode the multivariate steps $\mathbf{x} = [x_1, \ldots, x_T]$ into a sequence of high-level hidden vectors $\mathbf{h} = [h_1, \ldots, h_T]$, which follows exactly the same as the computational procedure used in F-RNN and F-SRNN . Recall that, in the case of SRNN, the computation of high-level vectors involves sampling the latent variables.
- Based on the high-level representations, for each multivariate step x_t, another neural model f_{low} will take both the elements $[x_{t,1}, \cdots, x_{t,L}]$ and the high-level vector h_t as input, and auto-regressively produce a sequence of low-level hidden vectors $[g_{t,1}, \cdots, g_{t,L}]$ where $g_{t,i} = f_{\text{low}}(x_{t,<i}, h_t)$. They can be then used to form the per-element output distributions in Eqn. (3.8) and (3.9).

In practice, the low-level model could simply be an RNN or a causally masked MLP [36], depending on our prior about the data. For convenience, we will refer to the hierarchical models as RNN-hier and SRNN-hier.

In some cases where all the elements within a step share the same statistical type, such as on the speech or MIDI dataset, one may alternatively consider a *flat model*. As the name suggests, the flat model will break the boundary between steps and flatten the data into a new uni-variate sequence, where each step is simply a single element. Then, the new uni-variate sequence can be directly fed into a standard RNN or SRNN model, producing each conditional factor in Eqn. (3.8) and (3.9) in an auto-regressive manner. Similarly, this class of RNN and SRNN will be referred to as RNN-flat and SRNN-flat, respectively. Compared to the hierarchical model, the flat variant implicitly assumes a sequential continuity between $x_{t,L}$ and $x_{t+1,1}$. Since this inductive bias matches the characteristics of multi-frame speech sequences, we expect the flat model to perform better in this case.

3.4.3 Experiment Results

Based on the seven datasets, we compare the performance of the models introduced above. To provide a random baseline, we include the δ-RNN with the random split scheme in the comparison. Moreover, previous results, if exist, are also presented to provide additional information. For a fair comparison, we make sure all models share the same parameter size. For more implementation details, please refer to the source code[1]. We also include the running time comparison in section 3.4.4. Finally, the results are summarized in Table 3.4, where we make several important observations.

Firstly, on the speech and MIDI datasets, models with auto-regressive (lower-half) output distributions obtain a dramatic advantage over models with fully factorized output distributions (upper-half), achieving new SOTA results on three speech datasets. This observation reminds us that, besides capturing the long-term temporal structure across steps, how to properly model the intra-step dependency is equally, if not more, crucial to the practical performance.

Secondly, when the auto-regressive output distribution is employed (lower-half), the non-stochastic recurrent models consistently outperform their stochastic counterparts across all datasets. In other words, the advantage of SRNN completely disappears once a powerful output distribution is used. Combined with the previous observation, it verifies our earlier concern that the so-called superiority of F-SRNN over F-RNN is merely a result of the biased experiment design in previous work.

In addition, as we expected, when the inductive bias of the flat model matches the characteristics of speech data, it will achieve a better performance than the hierarchical model. Inversely, when the prior does not match data property on the other datasets, the hierarchical model is always better. In the extreme case of permuted TIMIT, the flat model even falls behind factorized models, while the hierarchical model achieves a very decent performance that is even much better than what F-SRNN can achieve on the original TIMIT. This shows that the hierarchical model is usually more robust, especially when we don't have a good prior.

Overall, we don't find any advantage of employing stochastic latent variables for multivariate sequence modeling. Instead, relying on a full auto-regressive solution yields better or even state-of-the-art performances. Combined with the observation that δ-RNN-random can often achieve a competitive performance to F-SRNN, we believe that the theoretical advantage of latent-variable models in sequence modeling is still far from fulfilled, if ultimately possible. In addition, we suggest future development along this line compare with the simple but extremely robust baselines with an auto-regressive output distribution.

3.4.4 Training Time Comparison

Here, we report the training time of different methods in TIMIT dataset. The running times of training models for 40k updating steps on TIMIT are summarized in Table 3.5. The input length indicates the sample length of input during the training phrase. Admittedly, modeling the intra-step correlation (*-hier and *-flat model) would require extra computation time. Hence, this leads to a trade-off between quality and speed. Ideally, latent-variable models would provide

[1] https://github.com/zihangdai/reexamine-srnn

Input Length	8000							1000
Model Name	F-RNN	F-SRNN	δ-RNN	RNN-hier	SRNN-hier	RNN-flat	SRNN-flat	RNN-hier
Training Time	0.54h	0.94h	0.90h	9.92h	12.52h	37.48h	42.26h	1.7h
Log-Likelihood	32,745	69,296	66,453	109,641	107,912	117,721	109,284	101,713

Table 3.5: Training time comparison between various models.

a solution close to the sweet point of this trade-off. However, in our experiment, we find a simple hierarchical auto-regressive model trained with a shorter input length could already achieve significantly better performance with a comparable computation time (RNN-hier vs. F-SRNN in Table 3.5).

3.5 Summary

In summary, our re-examination reveals a misleading impression on the benefits of latent variables in sequence modeling. From our empirical observation, the main effect of latent variables is only to provide a mechanism to leverage the intra-step correlation, which is however, not as powerful as employing the straightforward auto-regressive decomposition. It remains unclear what leads to the significant gap between the theoretical potential of latent variables and their practical effectiveness, which we believe deserves more research attention. Meanwhile, given the large gain of modeling simultaneity, using sequential structures to better capture local patterns is another good future direction in sequence modeling.

Chapter 4

Long- and Short-term Time-series Network

4.1 Background and Motivation

Multivariate time series data are ubiquitous in our everyday life ranging from the prices in stock markets, the traffic flows on highways, the outputs of solar power plants, the temperatures across different cities, just to name a few. In such applications, users are often interested in the forecasting of the new trends or potential hazardous events based on historical observations on time series signals. For instance, a better route plan could be devised based on the predicted traffic jam patterns a few hours ahead, and a larger profit could be made with the forecasting of the near-future stock market.

Multivariate time series forecasting often faces a major research challenge, that is, how to capture and leverage the dynamics dependencies among multiple variables. Specifically, real-world applications often entail a mixture of short-term and long-term repeating patterns, as shown in Figure 4.1 which plots the hourly occupancy rate of a freeway. Apparently, there are two repeating patterns, daily and weekly. The former portraits the morning peaks vs. evening peaks, while the latter reflects the workday and weekend patterns. A successful time series forecasting model should be capture both kinds of recurring patterns for accurate predictions. As another example, consider the task of predicting the output of a solar energy farm based on the measured solar radiation by massive sensors over different locations. The long-term patterns reflect the difference between days vs. nights, summer vs. winter, etc., and the short-term patterns reflect the effects of cloud movements, wind direction changes, etc. Again, without taking both kinds of recurrent patterns into account, accurate time series forecasting is not possible. However, traditional approaches such as the large body of work in autoregressive methods [10, 43, 71, 123, 125] fall short in this aspect, as most of them do not distinguish the two kinds of patterns nor model their interactions explicitly and dynamically. Addressing such limitations of existing methods in time series forecasting is the main focus of this , for which we propose a novel framework that takes advantages of recent developments in deep learning research.

Deep neural networks have received an increasing amount of attention in time series analysis. A substantial portion of the previous work has been focusing on *time series classification*, i.e., the task of automated assignment of class labels to time series input. For instance, RNN architectures have been studied for extracting informative patterns from health-care sequential data [17, 75]

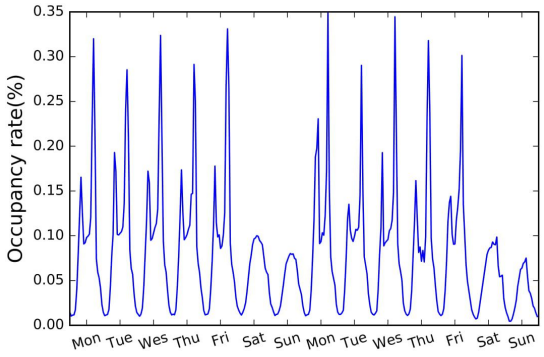


Figure 4.1: The hourly occupancy rate of a road in the bay area for 2 weeks

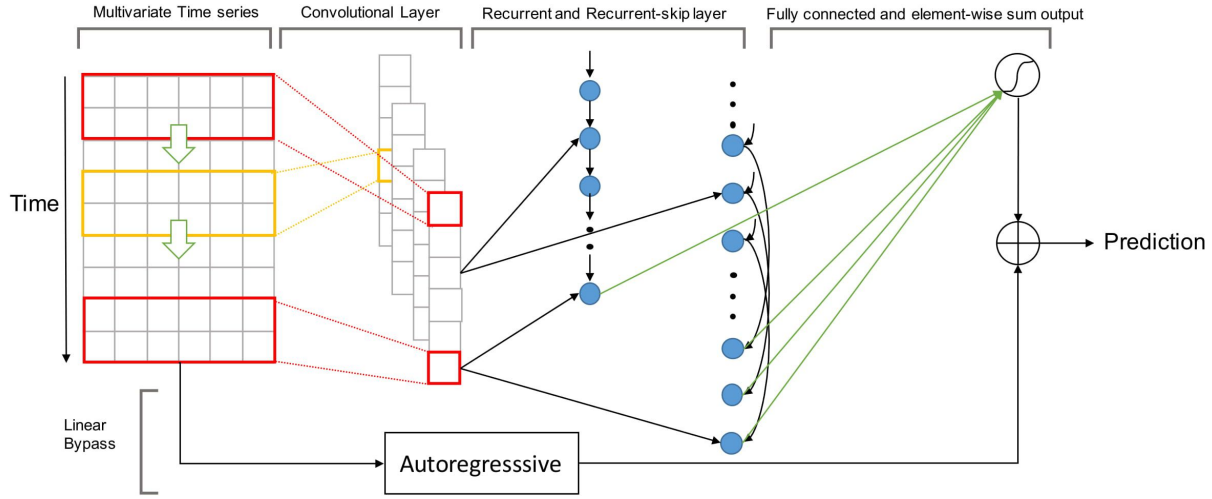


Figure 4.2: An overview of the Long- and Short-term Time-series network (LSTNet)

and classifying the data with respect diagnostic categories. RNN has been applied to mobile data, for classifying the input sequences with respect to actions or activities [44]. CNN models have been used in action/activity recognition [44, 66, 120], for the extraction of shift-invariant local patterns from input sequences as the features of classification models.

Deep neural networks have been studied for *time series forecasting* [28, 83, 124, 130], i.e., the task of using observed time series in the past to predict the unknown time series in a look-ahead horizon – the larger the horizon, the harder the problem. Efforts in this direction range from the early work using naive RNN models [24] and the hybrid models [50, 125, 126] combining the use of ARIMA [9] and Multilayer Perceptron (MLP), to the recent combination of vanilla RNN and Dynamic Boltzmann Machines in time series forecasting [28].

In this chapter, we propose a deep learning framework designed for the multivariate time series forecasting, namely Long- and Short-term Time-series Network (LSTNet), as illustrated in Figure 4.2. It leverages the strengths of both the convolutional layer to discover the local dependency patterns among multi-dimensional input variables and the recurrent layer to capture complex long-term dependencies. A novel recurrent structure, namely Recurrent-skip, is designed for capturing very long-term dependence patterns and making the optimization easier

as it utilizes the periodic property of the input time series signals. Finally, the LSTNet incorporates a traditional autoregressive linear model in parallel to the non-linear neural network part, which makes the non-linear deep learning model more robust for the time series with violate scale changing. In the experiment on the real world seasonal time series datasets, our model consistently outperforms the traditional linear models and GRU recurrent neural network.

4.2 Framework

In this section, we first formulate the time series forecasting problem, and then discuss the details of the proposed LSTNet architecture (Figure 4.2) in the following part. Finally, we introduce the objective function and the optimization strategy.

4.2.1 Problem Formulation

In this chapter, we are interested in the task of multivariate time series forecasting. More formally, given a series of fully observed time series signals $\boldsymbol{Y} = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T\}$ where $\boldsymbol{y}_t \in \mathbb{R}^n$, and n is the variable dimension, we aim at predicting a series of future signals in a rolling forecasting fashion. That being said, to predict $\boldsymbol{y}_{T+h}$ where h is the desirable horizon ahead of the current time stamp, we assume $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T\}$ are available. Likewise, to predict the value of the next time stamp $\boldsymbol{y}_{T+h+1}$, we assume $\{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T, \boldsymbol{y}_{T+1}\}$ are available. We hence formulate the input matrix at time stamp T as $X_T = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_T\} \in \mathbb{R}^{n \times T}$.

In the most of cases, the horizon of the forecasting task is chosen according to the demands of the environmental settings, e.g. for the traffic usage, the horizon of interest ranges from hours to a day; for the stock market data, even seconds/minutes-ahead forecast can be meaningful for generating returns.

Figure 4.2 presents an overview of the proposed LSTnet architecture. The LSTNet is a deep learning framework specifically designed for multivariate time series forecasting tasks with a mixture of long- and short-term patterns. In following sections, we introduce the building blocks for the LSTNet in detail.

4.2.2 Convolutional Component

The first layer of LSTNet is a convolutional network without pooling, which aims to extract short-term patterns in the time dimension as well as local dependencies between variables. The convolutional layer consists of multiple filters of width ω and height n (the height is set to be the same as the number of variables). The k-th filter sweeps through the input matrix X and produces

$$h_k = RELU(W_k * X + b_k) \tag{4.1}$$

where $*$ denotes the convolution operation and the output h_k would be a vector, and the $RELU$ function is $RELU(x) = \max(0, x)$. We make each vector h_k of length T by zero-padding on the left of input matrix X. The output matrix of the convolutional layer is of size $d_c \times T$ where d_c denotes the number of filters.

4.2.3 Recurrent Component

The output of the convolutional layer is simultaneously fed into the Recurrent component and Recurrent-skip component (to be described in subsection 4.2.4). The Recurrent component is a recurrent layer with the Gated Recurrent Unit (GRU) [21] and uses the $RELU$ function as the hidden update activation function. The hidden state of recurrent units at time t is computed as,

$$\begin{aligned} r_t &= \sigma(x_t W_{xr} + h_{t-1} W_{hr} + b_r) \\ u_t &= \sigma(x_t W_{xu} + h_{t-1} W_{hu} + b_u) \\ c_t &= RELU(x_t W_{xc} + r_t \odot (h_{t-1} W_{hc}) + b_c) \\ h_t &= (1 - u_t) \odot h_{t-1} + u_t \odot c_t \end{aligned} \tag{4.2}$$

where $\odot$ is the element-wise product, σ is the sigmoid function and x_t is the input of this layer at time t. The output of this layer is the hidden state at each time stamp. While researchers are accustomed to using $\tanh$ function as hidden update activation function, we empirically found $RELU$ leads to more reliable performance, through which the gradient is easier to back propagate.

4.2.4 Recurrent-skip Component

The Recurrent layers with GRU [21] and LSTM [46] unit are carefully designed to memorize the historical information and hence to be aware of relatively long-term dependencies. Due to gradient vanishing, however, GRU and LSTM usually fail to capture very long-term correlation in practice. We propose to alleviate this issue via a novel recurrent-skip component which leverages the periodic pattern in real-world sets. For instance, both the electricity consumption and traffic usage exhibit clear pattern on a daily basis. If we want to predict the electricity consumption at t o'clock for today, a classical trick in the seasonal forecasting model is to leverage the records at t o'clock in historical days, besides the most recent records. This type of dependencies can hardly be captured by off-the-shelf recurrent units due to the extremely long length of one period (24 hours) and the subsequent optimization issues. Inspired by the effectiveness of this trick, we develop a recurrent structure with temporal skip-connections to extend the temporal span of the information flow and hence to ease the optimization process. Specifically, skip-links are added between the current hidden cell and the hidden cells in the same phase in adjacent periods. The updating process can be formulated as,

$$\begin{aligned} r_t &= \sigma(x_t W_{xr} + h_{t-p} W_{hr} + b_r) \\ u_t &= \sigma(x_t W_{xu} + h_{t-p} W_{hu} + b_u) \\ c_t &= RELU(x_t W_{xc} + r_t \odot (h_{t-p} W_{hc}) + b_c) \\ h_t &= (1 - u_t) \odot h_{t-p} + u_t \odot c_t \end{aligned} \tag{4.3}$$

where the input of this layer is the output of the convolutional layer, and p is the number of hidden cells skipped through. The value of p can be easily determined for datasets with clear periodic patterns (e.g. $p = 24$ for the hourly electricity consumption and traffic usage datasets), and has to be tuned otherwise. In our experiments, we empirically found that a well-tuned p can

considerably boost the model performance even for the latter case. Furthermore, the LSTNet could be easily extended to contain variants of the skip length p.

We use a dense layer to combine the outputs of the Recurrent and Recurrent-skip components. The inputs to the dense layer include the hidden state of Recurrent component at time stamp t, denoted by h_t^R, and p hidden states of Recurrent-skip component from time stamp $t-p+1$ to t denoted by $h_{t-p+1}^S, h_{t-p+2}^S \ldots, h_t^S$. The output of the dense layer is computed as,

$$h_t^D = W^R h_t^R + \sum_{i=0}^{p-1} W_i^S h_{t-i}^S + b \tag{4.4}$$

where h_t^D is the prediction result of the neural network (upper) part in the Fig.4.2 at time stamp t.

4.2.5 Temporal Attention Layer

However, the Recurrent-skip layer requires a predefined hyper-parameter p, which is unfavorable in the nonseasonal time series prediction, or whose period length is dynamic over time. To alleviate such issue, we consider an alternative approach, attention mechanism [5], which learns the weighted combination of hidden representations at each window position of the input matrix. Specifically, the attention weights $\boldsymbol{\alpha}_t \in \mathbb{R}^q$ at current time stamp t are calculated as

$$\boldsymbol{\alpha}_t = AttnScore(H_t^R, h_{t-1}^R)$$

where $H_t^R = [h_{t-q}^R, \ldots, h_{t-1}^R]$ is a matrix stacking the hidden representation of RNN column-wisely and $AttnScore$ is some similarity functions such as dot product, cosine, or parameterized by a simple multi-layer perceptron.

The final output of temporal attention layer is the concatenation of the weighted context vector $\boldsymbol{c}_t = H_t \boldsymbol{\alpha}_t$ and last window hidden representation h_{t-1}^R, along with a linear projection operation

$$h_t^D = W[\boldsymbol{c}_t; h_{t-1}^R] + b.$$

4.2.6 Autoregressive Component

Due to the non-linear nature of the Convolutional and Recurrent components, one major drawback of the neural network model is that the scale of outputs is not sensitive to the scale of inputs. Unfortunately, in specific real datasets, the scale of input signals constantly changes in a non-periodic manner, which significantly lowers the forecasting accuracy of the neural network model. A concrete example of this failure is given in Section 4.3.6. To address this deficiency, similar in spirit to the highway network [101], we decompose the final prediction of LSTNet into a linear part, which primarily focuses on the local scaling issue, plus a non-linear part containing recurring patterns. In the LSTNet architecture, we adopt the classical Autoregressive (AR) model as the linear component. Denote the forecasting result of the AR component as $h_t^L \in \mathbb{R}^n$, and the coefficients of the AR model as $W^{ar} \in \mathbb{R}^{q^{ar}}$ and $b^{ar} \in \mathbb{R}$, where q^{ar} is the size of input

window over the input matrix. Note that in our model, all dimensions share the same set of linear parameters. The AR model is formulated as follows,

$$h_{t,i}^{L} = \sum_{k=0}^{q^{ar}-1} W_k^{ar} \boldsymbol{y}_{t-k,i} + b^{ar} \tag{4.5}$$

The final prediction of LSTNet is then obtained by integrating the outputs of the neural network part and the AR component:

$$\hat{\boldsymbol{Y}}_t = h_t^D + h_t^L \tag{4.6}$$

where $\hat{\boldsymbol{Y}}_t$ denotes the model's final prediction at time stamp t.

4.2.7 Objective function

The squared error is the default loss function for many forecasting tasks, the corresponding optimization objective is formulated as,

$$\underset{\Theta}{\text{minimize}} \quad \sum_{t \in \Omega_{Train}} ||\boldsymbol{Y}_t - \hat{\boldsymbol{Y}}_{t-h}||_F^2 \tag{4.7}$$

where Θ denotes the parameter set of our model, Ω_{Train} is the set of time stamps used for training, $||\cdot||_F$ is the Frobenius norm, and h is the horizon as mentioned in Section 4.2.1. The traditional linear regression model with the square loss function is named as Linear Ridge, which is equivalent to the vector autoregressive model with ridge regularization. However, experiments show that the Linear Support Vector Regression (Linear SVR) [109] dominates the Linear Ridge model in certain datasets. The only difference between Linear SVR and Linear Ridge is the objective function. The objective function for Linear SVR is,

$$\begin{aligned} \underset{\Theta}{\text{minimize}} \quad & \frac{1}{2}||\Theta||_F^2 + C \sum_{t \in \Omega_{Train}} \sum_{i=0}^{n-1} \xi_{t,i} \\ \text{subject to} \quad & |\hat{\boldsymbol{Y}}_{t-h,i} - \boldsymbol{Y}_{t,i}| \leq \xi_{t,i} + \epsilon, t \in \Omega_{Train} \\ & \xi_{t,i} \geq 0 \end{aligned} \tag{4.8}$$

where C and ϵ are hyper-parameters. Motivated by the remarkable performance of the Linear SVR model, we incorporate its objective function in the LSTNet model as an alternative of the squared loss. For simplicity, we assume $\epsilon = 0$[1], and the objective function above reduces to absolute loss (L1-loss) function as follows:

$$\underset{\Theta}{\text{minimize}} \quad \sum_{t \in \Omega_{Train}} \sum_{i=0}^{n-1} |\boldsymbol{Y}_{t,i} - \hat{\boldsymbol{Y}}_{t-h,i}| \tag{4.9}$$

The advantage of the absolute loss function is that it is more robust to the anomaly in the real time series data. In the experiment section, we use the validation set to decide to use which objective function, square loss Eq.4.7 or absolute one Eq.4.9.

[1]One could keep ϵ to make the objective function more faithful to the Linear SVR model without modifying the optimization strategy. We leave this for future study.

4.2.8 Optimization Strategy

In this chapter, our optimization strategy is the same as that in the traditional time series forecasting model. Supposing the input time series is $\boldsymbol{Y}_t = \{\boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_t\}$, we define a tunable window size q, and reformulate the input at time stamp t as $\boldsymbol{X}_t = \{\boldsymbol{y}_{t-q+1}, \boldsymbol{y}_{t-q+2}, \ldots, \boldsymbol{y}_t\}$. The problem then becomes a regression task with a set of feature-value pairs $\{\boldsymbol{X}_t, \boldsymbol{Y}_{t+h}\}$, and can be solved by Stochastic Gradient Decent (SGD) or its variants such as Adam [52].

4.3 Evaluation

We conducted extensive experiments with 9 methods (including our new methods) on 4 benchmark datasets for time series forecasting tasks. All the data and experiment codes are available online [2].

4.3.1 Methods for Comparison

The methods in our comparative evaluation are the follows.

- AR stands for the autoregressive model, which is equivalent to the one dimensional VAR model.
- LRidge is the vector autoregression (VAR) model with L2-regularization, which has been most popular for multivariate time series forecasting.
- LSVR is the vector autoregression (VAR) model with Support Vector Regression objective function [109] .
- TRMF is the autoregressive model using temporal regularized matrix factorization by [123].
- GP is the Gaussian Process for time series modeling. [35, 93]
- VAR-MLP is the model proposed in [125] that combines Multilayer Perception (MLP) and autoregressive model.
- RNN-GRU is the Recurrent Neural Network model using GRU cell.
- LSTNet-skip is our proposed LSTNet model with skip-RNN layer.
- LSTNet-Attn is our proposed LSTNet model with temporal attention layer.

For the single output methods above such as AR, LRidge, LSVR and GP, we just trained n models independently, i.e., one model for each of the n output variables.

4.3.2 Metrics

We used three conventional evaluation metrics defined as:

[2] https://github.com/laiguokun/LSTNet

- Root Relative Squared Error (RSE):

$$RSE = \frac{\sqrt{\sum_{(i,t)\in\Omega_{Test}}(Y_{it} - \hat{Y}_{it})^2}}{\sqrt{\sum_{(i,t)\in\Omega_{Test}}(Y_{it} - mean(\boldsymbol{Y}))^2}} \tag{4.10}$$

- Empirical Correlation Coefficient (CORR)

$$CORR = \frac{1}{n}\sum_{i=1}^{n} \frac{\sum_t \big(Y_{it} - mean(\boldsymbol{Y}_i)\big)\big(\hat{Y}_{it} - mean(\hat{\boldsymbol{Y}}_i)\big)}{\sqrt{\sum_t \big(Y_{it} - mean(\boldsymbol{Y}_i)\big)^2\big(\hat{Y}_{it} - mean(\hat{\boldsymbol{Y}}_i)\big)^2}} \tag{4.11}$$

where $\boldsymbol{Y}, \hat{\boldsymbol{Y}} \in \mathbb{R}^{n\times T}$ are ground true signals and system prediction signals, respectively. The RSE are the scaled version of the widely used Root Mean Square Error(RMSE), which is design to make more readable evaluation, regardless the data scale. For RSE lower value is better, while for CORR higher value is better.

4.3.3 Data

We used four benchmark datasets which are publicly available. Table 4.1 summarizes the corpus statistics.

- Traffic[3]: A collection of 48 months (2015-2016) hourly data from the California Department of Transportation. The data describes the road occupancy rates (between 0 and 1) measured by different sensors on San Francisco Bay area freeways.
- Solar-Energy[4] : the solar power production records in the year of 2006, which is sampled every 10 minutes from 137 PV plants in Alabama State.
- Electricity[5]: The electricity consumption in kWh was recorded every 15 minutes from 2012 to 2014, for n = 321 clients. We converted the data to reflect hourly consumption;
- Exchange-Rate: the collection of the daily exchange rates of eight foreign countries including Australia, British, Canada, Switzerland, China, Japan, New Zealand and Singapore ranging from 1990 to 2016.

All datasets have been split into training set (60%), validation set (20%) and test set (20%) in chronological order. To facilitate future research in multivariate time series forecasting, we publicize all raw datasets and the one after preprocessing in the website.

In order to examine the existence of long-term and/or short-term repetitive patterns in time series data, we plot autocorrelation graph for some randomly selected variables from the four datasets in Figure 4.3. Autocorrelation, also known as serial correlation, is the correlation of a signal with a delayed copy of itself as a function of delay defined below

$$R(\tau) = \frac{\mathbb{E}[(X_t - \mu)(X_{t+\tau} - \mu)]}{\sigma^2}$$

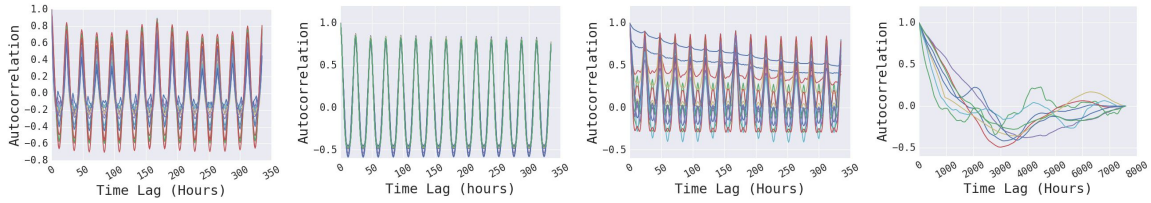
[3] http://pems.dot.ca.gov
[4] http://www.nrel.gov/grid/solar-power-data.html
[5] https://archive.ics.uci.edu/ml/datasets/ElectricityLoadDiagrams20112014

Datasets	T	D	L
Traffic	17,544	862	1 hour
Solar-Energy	52,560	137	10 minutes
Electricity	26,304	321	1 hour
Exchange-Rate	7,588	8	1 day

Table 4.1: Dataset Statistics, where T is length of time series, D is number of variables, L is the sample rate.

where X_t is the time series signals, μ is mean and σ^2 is variance. In practice, we consider the empirical unbiased estimator to calculate the autocorrelation.

We can see in the graphs (a), (b) and (c) of Figure 4.3, there are repetitive patterns with high autocorrelation in the Traffic, Solar-Energy and Electricity datasets, but not in the Exchange-Rate dataset. Furthermore, we can observe a short-term daily pattern (in every 24 hours) and long-term weekly pattern (in every 7 days) in the graph of the Traffic and Electricity dataset, which perfectly reflect the expected regularity in highway traffic situations and electricity consumptions. On the other hand, in graph (d) of the Exchange-Rate dataset, we hardly see any repetitive long-term patterns, expect some short-term local continuity. These observations are important for our later analysis on the empirical results of different methods. That is, for the methods which can properly model and successfully leverage both short-term and long-term repetitive patterns in data, they should outperform well when the data contain such repetitive patterns (like in Electricity, Traffic and Solar-Energy). On the other hand, if the dataset does not contain such patterns (like in Exchange-Rate), the advantageous power of those methods may not lead a better performance than that of other less powerful methods. We will revisit this point in Section 4.3.7 with empirical justifications.



(a) Traffic dataset (b) Solar dataset (c) Electricity dataset (d) Exchange dataset

Figure 4.3: Autocorrelation graphs of sampled variables form four datasets.

4.3.4 Experimental Details

We conduct grid search over all tunable hyper-parameters on the held-out validation set for each method and dataset. Specifically, all methods share the same grid search range of the window size q ranging from $\{2^0, 2^1, \ldots, 2^9\}$ if applied. For LRidge and LSVR, the regularization coefficient λ is chosen from $\{2^{-10}, 2^{-8}, \ldots, 2^8, 2^{10}\}$. For GP, the RBF kernel bandwidth σ and the noise level α are chosen from $\{2^{-10}, 2^{-8}, \ldots, 2^8, 2^{10}\}$. For TRMF, the hidden dimension is chosen from $\{2^2, \ldots, 2^6\}$ and the regularization coefficient λ is chosen from $\{0.1, 1, 10\}$. For LST-Skip

Dataset		Solar-Energy				Traffic				Electricity				Exchange-Rate			
		Horizon				Horizon				Horizon				Horizon			
Methods	Metrics	3	6	12	24	3	6	12	24	3	6	12	24	3	6	12	24
AR	RSE	0.2435	0.3790	0.5911	0.8699	0.5991	0.6218	0.6252	0.6293	0.0995	0.1035	0.1050	0.1054	0.0228	0.0279	**0.0353**	**0.0445**
(3)	CORR	0.9710	0.9263	0.8107	0.5314	0.7752	0.7568	0.7544	0.7519	0.8845	0.8632	0.8591	0.8595	0.9734	0.9656	0.9526	**0.9357**
LRidge	RSE	0.2019	0.2954	0.4832	0.7287	0.5833	0.5920	0.6148	0.6025	0.1467	0.1419	0.2129	0.1280	**0.0184**	0.0274	0.0419	0.0675
(3)	CORR	0.9807	0.9568	0.8765	0.6803	0.8038	0.8051	0.7879	0.7862	0.8890	0.8594	0.8003	0.8806	**0.9788**	**0.9722**	0.9543	0.9305
LSVR	RSE	0.2021	0.2999	0.4846	0.7300	0.5740	0.6580	0.7714	0.5909	0.1523	0.1372	0.1333	0.1180	0.0189	0.0284	0.0425	0.0662
(1)	CORR	0.9807	0.9562	0.8764	0.6789	0.7993	0.7267	0.6711	0.7850	0.8888	0.8861	0.8961	0.8891	0.9782	0.9697	**0.9546**	0.9370
TRMF	RSE	0.2473	0.3470	0.5597	0.9005	0.6708	0.6261	0.5956	0.6442	0.1802	0.2039	0.2186	0.3656	0.0351	0.0875	0.0494	0.0563
(0)	CORR	0.9703	0.9418	0.8475	0.5598	0.6964	0.7430	0.7748	0.7278	0.8538	0.8424	0.8304	0.7471	0.9142	0.8123	0.8993	0.8678
GP	RSE	0.2259	0.3286	0.5200	0.7973	0.6082	0.6772	0.6406	0.5995	0.1500	0.1907	0.1621	0.1273	0.0239	**0.0272**	0.0394	0.0580
(1)	CORR	0.9751	0.9448	0.8518	0.5971	0.7831	0.7406	0.7671	0.7909	0.8670	0.8334	0.8394	0.8818	0.8713	0.8193	0.8484	0.8278
VARMLP	RSE	0.1922	0.2679	0.4244	0.6841	0.5582	0.6579	0.6023	0.6146	0.1393	0.1620	0.1557	0.1274	0.0265	0.0304	0.0407	0.0578
(0)	CORR	0.9829	0.9655	0.9058	0.7149	0.8245	0.7695	0.7929	0.7891	0.8708	0.8389	0.8192	0.8679	0.8609	0.8725	0.8280	0.7675
RNN-GRU	RSE	0.1932	0.2628	0.4163	0.4852	0.5358	0.5522	0.5562	0.5633	0.1102	0.1144	0.1183	0.1295	0.0192	0.0264	0.0408	0.0626
(0)	CORR	0.9823	0.9675	0.9150	0.8823	0.8511	0.8405	0.8345	0.8300	0.8597	0.8623	0.8472	0.8651	0.9786	0.9712	0.9531	0.9223
LST-Skip	RSE	0.1843	0.2559	**0.3254**	0.4643	**0.4777**	**0.4893**	**0.4950**	**0.4973**	**0.0864**	**0.0931**	0.1007	**0.1007**	0.0226	0.0280	0.0356	0.0449
(17)	CORR	0.9843	0.9690	**0.9467**	0.8870	**0.8721**	**0.8690**	**0.8614**	**0.8588**	**0.9283**	**0.9135**	**0.9077**	**0.9119**	0.9735	0.9658	0.9511	0.9354
LST-Attn	RSE	**0.1816**	**0.2538**	0.3466	**0.4403**	0.4897	0.4973	0.5173	0.5300	0.0868	0.0953	**0.0984**	0.1059	0.0276	0.0321	0.0448	0.0590
(7)	CORR	**0.9848**	**0.9696**	0.9397	**0.8995**	0.8704	0.8669	0.8540	0.8429	0.9243	0.9095	0.9030	0.9025	0.9717	0.9656	0.9499	0.9339

Table 4.2: Results summary (in RSE and CORR) of all methods on four datasets: 1) each row has the results of a specific method in a particular metric; 2) each column compares the results of all methods on a particular dataset with a specific horizon value; 3) bold face indicates the best result of each column in a particular metric; and 4) the total number of bold-faced results of each method is listed under the method name within parentheses.

and LST-Attn, we adopted the training strategy described in Section 4.2.8. The hidden dimension of the Recurrent and Convolutional layer is chosen from $\{50, 100, 200\}$, and $\{20, 50, 100\}$ for Recurrent-skip layer. The skip-length p of Recurrent-skip layer is set as 24 for the Traffic and Electricity dataset, and tuned range from 2^1 to 2^6 for the Solar-Energy and Exchange-Rate datasets. The regularization coefficient of the AR component is chosen from $\{0.1, 1, 10\}$ to achieve the best performance. We perform dropout after each layer, except input and output ones, and the rate usually is set to 0.1 or 0.2. The Adam[52] algorithm is utilized to optimize the parameters of our model.

4.3.5 Main Results

Table 4.2 summarizes the evaluation results of all the methods (8) on all the test sets (4) in all the metrics (3). We set $horizon = \{3, 6, 12, 24\}$, respectively, which means the horizons was set from 3 to 24 hours for the forecasting over the Electricity and Traffic data, from 30 to 240 minutes over the Solar-Energy data, and from 3 to 24 days over the Exchange-Rate data. The larger the horizons, the harder the prediction tasks. The best result for each (data, metric) pair is highlighted in bold face in this table. The total count of the bold-faced results is 17 for LSTNet-Skip (one version of the proposed LSTNet), 7 for LSTNet-Attn (the other version of our LSTNet), and between 0 to 3 for the rest of the methods.

Clearly, the two proposed models, LSTNet-skip and LSTNet-Attn, consistently enhance over state-of-the-art on the datasets with periodic pattern, especially in the settings of large horizons. Besides, LSTNet outperforms the strong neural baseline RNN-GRU by **9.2%, 11.7%, 22.2%**

in RSE metric on Solar-Energy, Traffic and Electricity dataset respectively when the horizon is 24, demonstrating the effectiveness of the framework design for complex repetitive patterns. What's more, when the periodic pattern q is not clear from applications, users may consider LSTNet-attn as alternative over LSTNet-skip, given the former still yield considerable improvement over the baselines. But the proposed LSTNet is slightly worse than AR and LRidge on the Exchange-Rate dataset. Why? Recall that in Section 4.3.3 and Figure 4.3 we used the autocorrelation curves of these datasets to show the existence of repetitive patterns in the Solar-Energy, Traffic and Electricity datasets but not in Exchange-Rate. The current results provide empirical evidence for the success of LSTNet models in modeling long-term and short-term dependency patterns when they do occur in data. Otherwise, LSTNet performed comparably with the better ones (AR and LRidge) among the representative baselines.

Compared the results of univariate AR with that of the multivariate baseline methods (LRidge, LSVR and RNN), we see that in some datasets, i.e. Solar-Energy and Traffic, the multivariate approaches is stronger, but weaker otherwise, which means that the richer input information would causes overfitting in the traditional multivariate approaches. In contrast, the LSTNet has robust performance in different situations, partly due to its autoregressive component, which we will discuss further in Section 4.3.6.

4.3.6 Ablation Study

To demonstrate the efficiency of our framework design, a careful ablation study is conducted. Specifically, we remove each component one at a time in our LSTNet framework. First, we name the LSTNet without different components as follows.

- **LSTw/oskip**: The LSTNet models without the Recurrent-skip component and attention component.
- **LSTw/oCNN**: The LSTNet-skip models without the Convolutional component.
- **LSTw/oAR**: The LSTNet-skip models without the AR component.

For different baselines, we tune the hidden dimension of models such that they have similar numbers of model parameters to the completed LSTNet model, removing the performance gain induced by model complexity.

The test results measured using RSE and CORR are shown in Figure 4.5[6]. Several observations from these results are worth highlighting:

- The best result on each dataset is obtained with either LST-Skip or LST-Attn.
- Removing the AR component (in LSTw/oAR) from the full model caused the most significant performance drops on most of the datasets, showing the crucial role of the AR component in general.
- Removing the Skip and CNN components in (LSTw/oCNN or LSTw/oskip) caused big performance drops on some datasets but not all. All the components of LSTNet together leads to the robust performance of our approach on all the datasets.

[6]We omit the results in RAE as it shows similar comparison with respect to the relative performance among the methods.

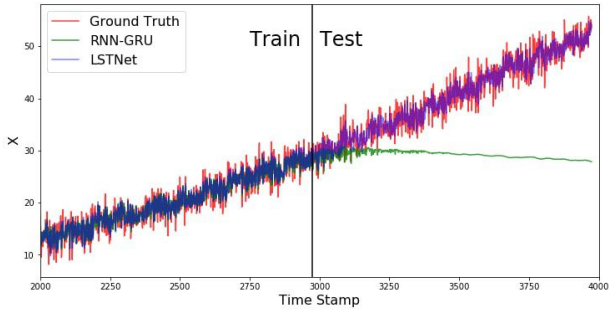


Figure 4.4: Simulation Test: Left side is the training set and right side is test set.

The conclusion is that our architecture design is most robust across all experiment settings, especially with the large horizons.

As for why the AR component would have such an important role, our interpretation is that AR is generally robust to the scale changing in data. To empirically validate this intuition we plot one dimension (one variable) of the time series signals in the electricity consumption dataset for the duration from 1 to 5000 hours in Figure 4.6, where the blue curve is the true data and the red curve is the system-forecasted signals. We can see that the true consumption suddenly increases around the 1000th hour, and that LSTNet-Skip successfully captures this sudden change but LSTw/oAR fails to react properly.

In order to better verify this assumption, we conduct a simulation experiment. First, we randomly generate an autoregressive process with the scale changing by the following steps. Firstly, we randomly sample a vector, $w \sim N(0, I), w \in \mathbb{R}^p$, where p is a given window size. Then the generated autoregressive process x_t can be described as

$$x_t = \sum_{i=1}^{p} w_i x_{t-i} + \epsilon \tag{4.12}$$

where $\epsilon \sim N(\mu, 1)$. To inject the scale changing, we increase the mean of Gaussian noise by μ_0 every T timestamp. Then the Gaussian noise of time series x_t can be written as

$$\epsilon \sim N(\lfloor t/T \rfloor \, \mu_0, 1) \tag{4.13}$$

where the $\lfloor \cdot \rfloor$ denotes the floor function. We split the time series as the training set and test in chronological order, and test the RNN-GRU and the LSTNet models. The result is illustrated in Figure 4.4. Both RNN and LSTNet can memorize the pattern in training set (left side). But, the RNN-GRU model cannot follow the scale changing pattern in the test set (right side). Oppositely, the LSTNet model fits the test set much better. In other words, the normal RNN module, or says the neural-network component in LSTNet, may not be sufficiently sensitive to violated scale fluctuations in data (which is typical in Electricity data possibly due to random events for public holidays or temperature turbulence, etc.), while the simple linear AR model can make a proper adjustment in the forecasting.

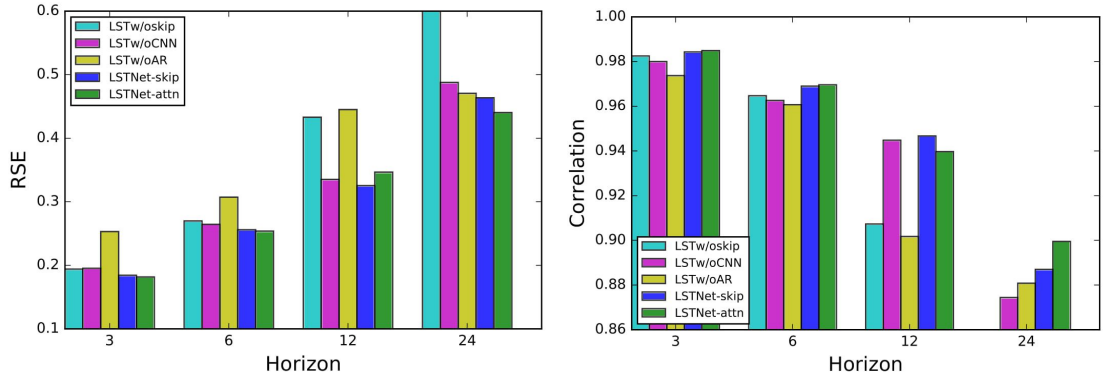
In summary, this ablation study clearly justifies the efficiency of our architecture design. All components have contributed to the excellent and robust performance of LSTNet.

4.3.7 Mixture of long- and short-term patterns

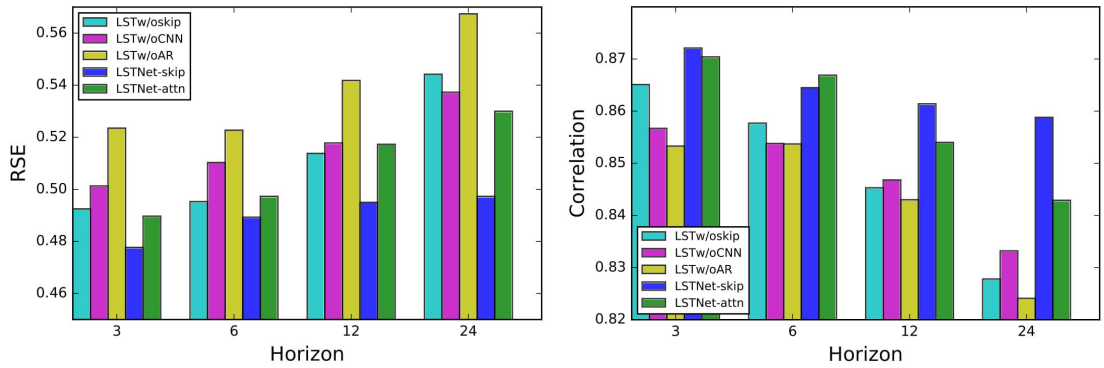
To illustrate the success of LSTNet in modeling the mixture of short-term and long-term recurring patterns in time series data, Figure 4.7 compares the performance of LSTNet and VAR on an specific time series (one of the output variables) in the Traffic dataset. As discussed in Section 4.3.3, the Traffic data exhibit two kinds of repeating patterns, i.e. the daily ones and the weekly ones. We can see in Figure 4.7 that the true patterns (in blue) of traffic occupancy are very different on Fridays and Saturdays, and another on Sunday and Monday. The Figure 4.7 is the prediction result of the VAR model (part (a)) and LSTNet (part (b)) of a traffic flow monitor sensor, where their hyper-parameters are chosen according to the RMSE result on the validation set. The figure shows that the VAR model is only capable to deal with the short-term patterns. The pattern of prediction results of the VAR model only depend on the day before the predictions. We can clearly see that the results of it in Saturday (2rd and 9th peaks) and Monday (4th and 11th peaks) is different from the ground truth, where the ground truth of Monday (weekday) has two peaks, one peak for Saturday (weekend). In the contrary, our proposed LSTNet model performs two patterns for weekdays and weekends respectfully. This example proves the ability of LSTNet model to memorize short-term and long-term recurring patterns simultaneously.

4.4 Summary

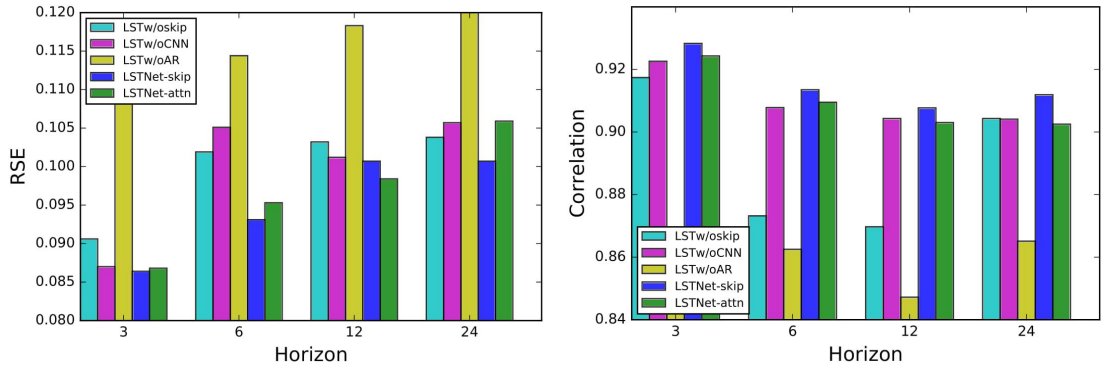
In this part of thesis, we presented a novel deep learning framework (LSTNet) for the task of multivariate time series forecasting. By combining the strengths of convolutional and recurrent neural networks and an autoregressive component, the proposed approach significantly improved the state-of-the-art results in time series forecasting on multiple benchmark datasets. With in-depth analysis and empirical evidence, we show the efficiency of the architecture of LSTNet model, and that it indeed successfully captures both short-term and long-term repeating patterns in data, and combines both linear and non-linear models for robust prediction.



(a) Solar-Energy dataset



(b) Traffic dataset



(c) Electricity dataset

Figure 4.5: Results of LSTNet in the ablation tests on the Solar-Energy, Traffic and Electricity dataset

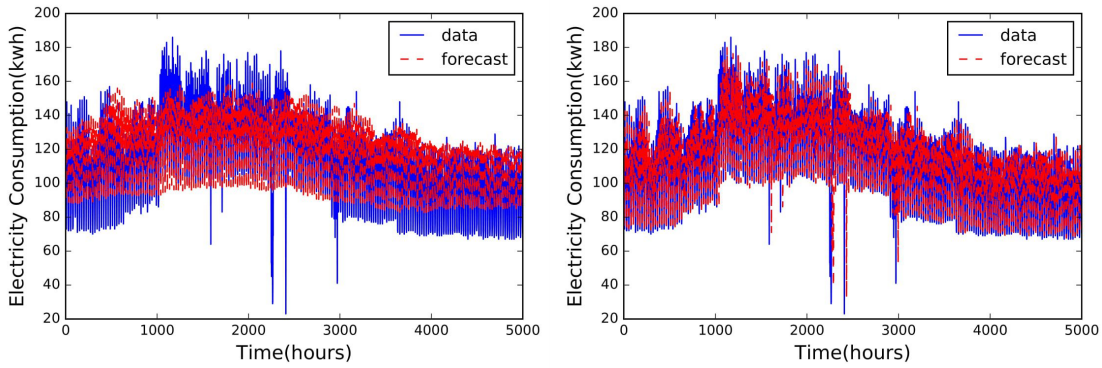


Figure 4.6: The predicted time series (red) by LSTw/oAR (a) and by LST-Skip (b) vs. the true data (blue) on Electricity dataset with $horizon = 24$

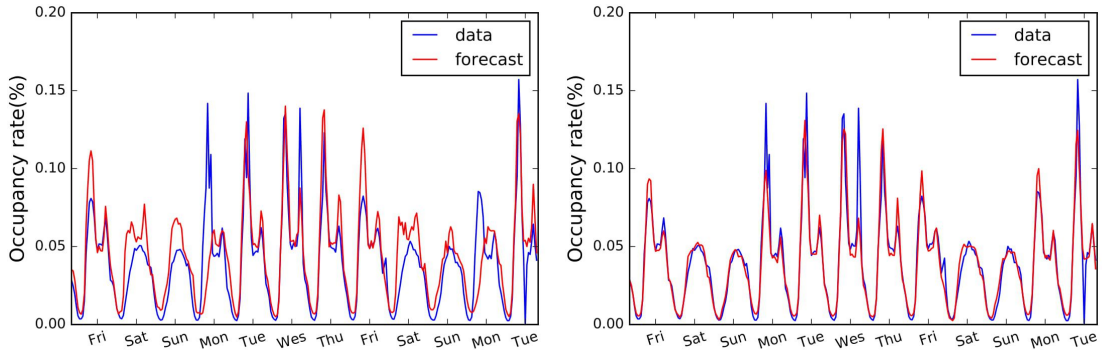


Figure 4.7: The true time series (blue) and the predicted ones (red) by VAR (a) and by LSTNet (b) for one variable in the Traffic occupation dataset. The X axis indicates the week days and the forecasting $horizon = 24$. VAR inadequately predicts similar patterns for Fridays and Saturdays, and ones for Sundays and Mondays, while LSTNet successfully captures both the daily and weekly repeating patterns.

Chapter 5

Funnel Transformer

5.1 Motivation

With the recent success of unsupervised language pretraining [23, 30, 56, 65, 70, 76, 77, 89, 90, 99, 100, 121], the power of neural self-attention models (a.k.a. Transformer) [110] has been pushed to a new level, leading to dramatic advancements in machine learning and natural language processing (NLP). More importantly, it has been observed that with more FLOPs invested in longer pretraining and/or larger models, the performance of pretrained Transformer models consistently improve. However, it is extremely expensive to pretrain or even just to finetune the state-of-the-art self-attention models, as they require much more FLOPs and memory resources compared to traditional models in NLP. This largely limits their applications and success in more fields.

Given this challenge, there has been an increasing amount of efforts to reduce the costs of pretraining and finetuning self-attention models. From the perspective of post-pretraining processing, typical approaches include distillation, pruning and quantization of various kinds, which try to derive a lighter model from an well-pretrained model by taking advantage of the richer signals in the larger model or learning to remove less important operations. Another line of research aims at designing an architecture that not only has a lower resource-to-performance ratio (more efficient) but also *scales as well as* the Transformer, at least in certain domains. Most of such methods build upon the Transformer backbone and focus on redesigning its building blocks. Representative solutions include searching for better micro operation or macro module designs [18, 98], replacing the full pairwise attention with local operations such as convolution [116] and dynamic convolution [115], and optimizing the hidden size combinations for existing blocks [103].

Across the wide variety of ideas mentioned above, a common strategy is to identify redundant operations or representations and replace them with more efficient ones. Inspired by this line of thinking, in this work, we will be focusing on the potential redundancy induced by always maintaining a *full-length sequence* of hidden representations across all layers in Transformer. Intuitively, for many sequence-level NLP tasks such as text classification and ranking, the most common use case is to extract a *single* vector from the entire sequence, which does not necessarily preserve all information down to the token-level granularity. Hence, for such tasks, the

full-length sequence of hidden states may contain significant redundancy. This is analogous to the case of image recognition, where the convolution neural network gradually reduces the spatial resolution/size of feature maps as the neural network goes deeper. In addition, linguistic prior also encourages gradually merging nearby tokens (words) into larger semantic units (phrases), which naturally leads to a shorter sequence of representations.

Concretely, we propose to gradually reduce the sequential resolution (i.e. length) of the hidden representation in self-attention models. Immediately, the reduction in sequence length can lead to significant savings in both FLOPs and memory. More importantly, the saved computational resource can be directly re-invested in constructing a deeper (or wider) model to boost the model capacity without additional computational burden. In addition, to address the challenge that common pretraining objectives such as masked language modeling (MLM) [30] require separate representations for each token, we design a simple strategy to decode a full-length sequence of deep representations from the hidden state of reduced length. As a result, the proposed model can be directly trained without modifying the pretraining objectives, as well as adopted for downstream tasks that require token-level representations.

Empirically, with comparable or even fewer FLOPs, by trading sequential resolution for depth, our proposed model achieves an improved performance over the standard Transformer on a wide variety of sequence-level prediction tasks, including text classification, language understanding, and reading comprehension.

5.2 Method

5.2.1 Background

Transformer Architecture The Transformer architecture [110] is a highly modularized neural network, where each Transformer layer consists of two sub-modules, namely the multi-head self-attention (S-Attn) and position-wise feed-forward network (P-FFN). Both sub-modules are wrapped by a residual connection and layer normalization. Schematically, given a length T sequence of hidden states $\mathbf{h} = [h_1, \ldots, h_T]$, the computation of a single Transformer layer can be expressed as

$$\mathbf{h} \leftarrow \text{LayerNorm}(\mathbf{h} + \text{S-Attn}(\text{Q} = \mathbf{h}, \text{KV} = \mathbf{h})), \tag{5.1}$$

$$h_i \leftarrow \text{LayerNorm}(h_i + \text{P-FFN}(h_i)), \quad \forall i = 1, \cdots, T. \tag{5.2}$$

Pretraining Objectives The most commonly used pretraining objective is the masked language modeling (MLM) proposed by BERT [30]. For a length-T natural language sequence $\mathbf{x}$ sample from a large unlabeled set $\mathcal{D}$, the MLM objective first constructs a corrupted sequence $\hat{\mathbf{x}}$ by randomly replacing 15% of the tokens of $\mathbf{x}$ with a special token `[mask]` and then trains a Transformer model [30] to reconstruct the original $\mathbf{x}$ based on $\hat{\mathbf{x}}$, i.e.,

$$\max_{\theta} \ \mathcal{J}_{\text{MLM}}(\theta) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\mathbb{E}_{\mathcal{I}} \sum_{i\in\mathcal{I}} \log P_\theta(x_i \mid \hat{\mathbf{x}}_{\mathcal{I}}) = \mathbb{E}_{\mathbf{x}\sim\mathcal{D}}\mathbb{E}_{\mathcal{I}} \sum_{i\in\mathcal{I}} \log \frac{\exp\left(e(x_i)^\top h_i(\hat{\mathbf{x}}_{\mathcal{I}})\right)}{\sum_{x'} \exp\left(e(x')^\top h_i(\hat{\mathbf{x}}_{\mathcal{I}})\right)},$$

where $\mathcal{I}$ is the positions of masked tokens, the subscript in $\hat{\mathbf{x}}_{\mathcal{I}}$ emphasizes its dependence on $\mathcal{I}$, $e(x)$ denotes the embedding of the token x, and $h_i(\hat{\mathbf{x}}_{\mathcal{I}})$ the last-layer hidden state at position i produced by the Transformer model. After pretraining, the entire model is finetuned in downstream tasks.

To show the generality of our proposed model, we also experiment with another pretraining objective ELECTRA [23]. Different from MLM, ELECTRA relies a pair of jointly trained generator and discriminator. Specifically, the generator usually has a smaller size (1/4 of that of the discriminator) and is directly trained via the MLM objective, i.e., $\max_{\theta_G} \mathcal{J}_{\text{MLM}}(\theta_G)$. Then, for each masked position, a token is sampled from the reconstruction distribution of the generator to replace the `[mask]` token and form a new sequence $\tilde{\mathbf{x}}$, i.e., if $i \in \mathcal{I}$, $\tilde{x}_i \sim P_{\theta_G}(x_i \mid \hat{\mathbf{x}}_{\mathcal{I}})$ else $\tilde{x}_i = x_i$. Given the new sequence $\tilde{\mathbf{x}}$, the discriminator is then trained to distinguish whether each token in $\tilde{\mathbf{x}}$ is real (same as $\mathbf{x}$) or fake (different from $\mathbf{x}$) via binary classification. After pretraining, only the discriminator will be used during finetuning and the generator is simply discarded.

Discussion Note that both pretraining objectives introduced above require the ability to produce a hidden state for each input token, i.e., $h_i(\hat{\mathbf{x}}_{\mathcal{I}})$ and $h_i(\tilde{\mathbf{x}})$. Due to this requirement, it seems natural to keep a full sequence of hidden states. However, in contrast, many sequence-level downstream tasks like classification or ranking only need a single-vector summary of the entire sequence. Fundamentally, this suggests that some kind of compression is usually required to remove the unnecessary redundancy during finetuning. This observation immediately leads to the following two questions:

- Can we design a general model that is equally expressive but more efficient by compressing the full sequence of hidden states into a more compact form?
- With the compressed representations, how can the model retain the ability to produce token-level representations for pretraining?

To answer these two questions, we next present our proposed architecture.

5.2.2 Proposed Architecture

To inherit the high capacity and optimization advantages of the Transformer architecture, the proposed model keeps the same overall skeleton of interleaved S-Attn and P-FFN sub-modules wrapped by residual connection and layer normalization. But differently, to achieve representation compression and computation reduction, our model employs an encoder that gradually reduces the sequence length of the hidden states as the layer gets deeper. In addition, for tasks involving per-token predictions like pretraining, a simple decoder is used to reconstruct a full sequence of token-level representations from the compressed encoder output.

Encoder As illustrated in the left part of Fig. 5.1, the encoder consists of several blocks of consecutive Transformer layers. Within each block, the sequence length of the hidden states always remains the same. But when going from a lower-level block to a higher-level block, the length of the hidden sequence is reduced by performing certain type of pooling along the

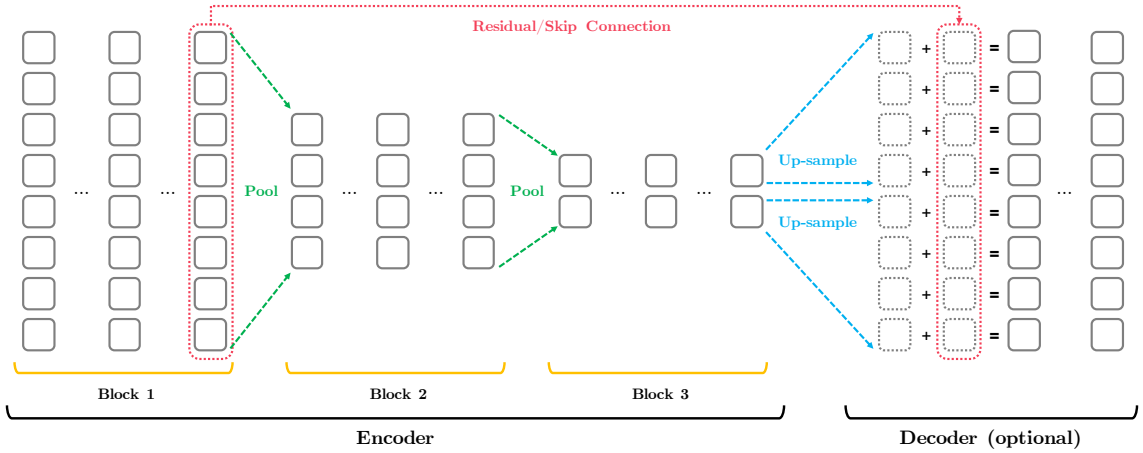


Figure 5.1: High-level visualization of the proposed Funnel-Transformer.

sequence dimension, i.e.,

$$\mathbf{h}' \leftarrow \text{Pooling}(\mathbf{h}), \tag{5.3}$$

where $\mathbf{h} \in \mathbb{R}^{T \times D}$ and $\mathbf{h}' \in \mathbb{R}^{T' \times D}$ for some $T' < T$. Importantly, instead of directly feeding the pooled sequence $\mathbf{h}'$ into the first S-Attn layer of the new block, we only use pooled sequence $\mathbf{h}'$ to construct the query vector (and the residual signal) of the self-attention, while the unpooled sequence $\mathbf{h}$ serves that role of key and value vectors, i.e.

$$\mathbf{h} \leftarrow \text{LayerNorm}(\mathbf{h}' + \text{S-Attn}(\text{Q} = \mathbf{h}', \text{KV} = \mathbf{h})). \tag{5.4}$$

Note that the output sequence of this special S-Attn module has the same length as the pooled sequence $\mathbf{h}'$. To understand the advantage of this particular design, it is helpful to compare the proposed "pool-query-only" variant with the naive alternative of using $\mathbf{h}'$ for both the query and key-value vectors, i.e., S-Attn$(\text{Q} = \mathbf{h}', \text{KV} = \mathbf{h}')$:

- Under the naive approach, the compression is solely controlled by the pooling operation, which is finished before the attention module. Hence, relatively simple pooling methods such as average/mean pooling won't be able to achieve good compression.
- Under the pool-query-only variant, the compression depends on not only how the pooling is performed, but also how the self-attention weighted sums the unpooled sequence to form each pooled vector. Effectively, the particular attention here can be seen as a type of linear compression that combines T bases into a smaller number of T' "compressed bases". Therefore, with minimum computational overhead, this variant makes compression operation more expressive.

With this particular pool-query-only design in place, we find the simplest strided mean pooling applied to each sliding window of the sequence work very well in practice. For simplicity, we only experiment with stride 2 and window size 2 in this work. Hence, the pooling operation will reduce the sequence by half and each pooled hidden state corresponds to a window of 2 unpooled hidden vectors. Intuitively, this type of pooling roughly follows the linguistic prior that nearby tokens could be gradually merged (or compressed) into a larger semantic component. Once the sequence length is halved after the pooling and pool-query-only attention, the rest of the encoder computation simply follows the standard updates in Eqn. (5.2) and (5.1).

Finally, as an extra implementation detail, recall that a particular design in language pretraining is to add a special token `[cls]` to the beginning of the original input sequence, and use the last-layer hidden state corresponding to `[cls]` (i.e., h_1) as the representation of the sequence. To prevent the pooling from destroying this special structure, we first separate the `[cls]` hidden state and the rest of hidden states and only apply the pooling to the rest of hidden states.

Decoder In order to recover a full sequence of hidden states from the encoder output of reduced length, a natural idea would be performing some kind of up-sampling. For instance, in image generation or super-resolution, deconvolution (transposed convolution) or parameter-free resizing with bilinear interpolation are often used to increase the spatial resolution of the feature map. Hence, we can simply adapt these ideas from 2D processing to our 1D case and apply proper up-sampling to the encoder output.

However, instead of performing multiple up-samplings with small expansion rate (e.g. increasing the sequence length by 2x each time) as in image domain, we here choose to employ a single up-sampling with a large expansion rate, as shown on the right part of Fig. 5.1. Specifically, given the output sequence $\mathbf{h}^{\mathrm{M}}$ of length $T_M = T/2^{M-1}$ from an M-block encoder, we directly up-sample it to a full-length sequence $\mathbf{h}^{\text{up}} = [h_1^{\text{up}}, \cdots, h_T^{\text{up}}]$ by repeating each hidden vector 2^{M-1} times:

$$\forall i = 1, \cdots, T, \quad h_i^{\text{up}} = h^M_{i//2^{M-1}}, \tag{5.5}$$

where $\cdot//\cdot$ denotes floor division. However, note that every 2^{M-1} consecutive vectors in $\mathbf{h}^{\text{up}}$ are exactly the same and hence do not contain detailed token-level information. Hence, we further extract the last-layer hidden states from the *first block* of the encoder $\mathbf{h}^1$, which still has the full length T and contains the uncompressed token-level information. Then, the lower-level representation $\mathbf{h}^1$ and up-sampled higher-level representation $\mathbf{h}^{\text{up}}$ are added together to form a deep token-level representation $\mathbf{g} = \mathbf{h}^1 + \mathbf{h}^{\text{up}}$. Effectively, this forms a residual/skip connection that enables detailed token information and potentially easier optimization. In addition, we stack a few more Transformer layers upon $\mathbf{g}$ to achieve a better deep fusion of the low-level and high-level features. In this work, we always use 2 Transformer layers in decoder.

It is important to emphasize that the decoder is *only used if the task requires token-level prediction*, such as in standard pretraining or sequence labeling. For tasks that only requires a single vectorial representation of the sequence like classification, the decoder is discarded after pretraining and only the encoder is finetuned. Finally, to emphasize the filtering/compression property of the encoder as well as its shape, we name the proposed model `Funnel-Transformer` (F-TFM).

5.2.3 Complexity & Capacity Analysis

With the architecture design specified, we now analyze how the sequence compression affects the complexity and capacity of the proposed model, especially compared to the standard Transformer.

Firstly, for a Transformer layer with an S-Attn and a P-FFN module of hidden size D, the

complexity of processing a length-T sequence is $O(T^2D + TD^2)$.[1] Hence, every time the sequence length is reduced by half in the encoder, we enjoy a *super-linear* (more than half) complexity drop. In practice, as the $O(TD^2)$ term has a large constant, a near-linear speedup is observed more often. The super-linear effect is more detectable when the sequence length is relatively long like in pretraining. Therefore, given the same FLOPs, we can at least trade a full-length layer in the 1st block for 2^{m-1} layers in the m-th block, which provides an economical way to increase the depth of network.

On the other hand, the capacity of a compressed-length layer is clearly upper-bounded by that of a normal full-length layer. In most cases where the compression is lossy, reducing the sequence length will inevitably lead to capacity drop. The good news is that the capacity drop of a single layer could be well compensated by re-investing the saved FLOPs in stacking more cheaper layers of reduced length or increasing the width of the model.

As a concrete example, for a Transformer of $\text{BERT}_{\text{Base}}$ size, i.e., 12 layers of hidden size 768 (L12H768), we may construct a Funnel-Transformer of 3 blocks where each block has 6 layers of hidden size 768 (B6-6-6H768). Despite having 18 layers in total, when finetuned for classification, the FLOPs of the B6-6-6H768 architecture only corresponds to at most $6 + 6/2 + 6/4 = 10.5$ full-length layers, clearly fewer than that of L12H768. More importantly, as we will show in the experiments, B6-6-6H768 significantly outperforms L12H768. While intuitive, how to construct an optimal block layout given this *depth-length trade-off* remains an open challenge. For this work, we only consider relatively regular layout and leave more systematic studies for future work.

Finally, notice that trading sequential resolution for depth or width has a side effect of increasing the total number of parameters. For instance, B6-6-6H768 has 1.5x Transformer parameters compared to L12H768. In practice, more parameters may increase communication cost in distributed training as well as the memory consumption and memory access time. A simple remedy is to perform certain parameter sharing, as used in ALBERT, to recover the same parameter count. Taking B6-6-6H768 as an example, one may tie the parameters for every two layers in the 2nd and 3rd blocks, denoted as B6-3x2-3x2H768, which gives back the same number of parameters to L12H768. However, parameter sharing could result in performance loss. Fundamentally, this brings us another trade-off between the gain (capacity) and cost (memory and communication cost) of using more parameters, which can be highly device dependent.

5.3 Implementation Optimization

5.3.1 Sequence Truncation for Separating `[cls]` trick

As discussed in Section 5.2.2, to avoid breaking the `[cls]` structure commonly used in pretraining, we do not apply the pooling operation to the `[cls]` and keep the hidden state corresponding to `[cls]` intact. While conceptually simple, a naive implementation could slow down the computation by 15% due to the "irregular" sequence length caused by such an operation. Specifically, assume that sequence length of an input sample is a power of two, i.e., 2^p, which usually is 512

[1] Since the corresponding memory complexity is simply $O(T^2 + TD)$, which is always offset by a multiplier $1/D$, we will focus on the computation complexity with the conclusion directly carried through.

in the pretraining phase. After one pooling operation with the `[cls]` intact, the length of the pooled sequence becomes $2^{p-1}+1$, which is not a power of 2 anymore. As a result, it can cause memory misalignment and the waste of paralleled computation power in accelerators, leading to substantial speed loss.

To resolve this issue, we employ a simple strategy to truncate the last token after the pooling. Formally, denoting the pooled hidden state as $\mathbf{h} = \{h_{\texttt{[cls]}}, h_1, \cdots, h_{2^{p-1}}\}$, the truncation can be expressed as

$$\hat{\mathbf{h}} = \text{truncate}(\mathbf{h}) = [h_{\texttt{[cls]}}, h_1, \cdots, h_{2^{p-1}-1}] \tag{5.6}$$

With this simple trick, we can always keep the sequence length a power of 2, hence avoiding the slowdown caused by maintaining an independent `[cls]` hidden state.

5.3.2 Relative Positional Attention Implementation

In this work, we use the relative positional attention parameterization proposed in the Transformer-XL [26]. To facilitate further discussion, we first review the details of this parameterization. Taking the case of single head attention as the example head. Let T, D be the sequence length and hidden dimension respectively. Then, the pre-softmax attention score A_{ij} between a pair of positions i and j consists of two terms:

$$A_{ij} = \underbrace{(W_Q h_i + v)^\top (W_K h_j)}_{\text{content term}} + \underbrace{(W_Q h_i + u)^\top (W_R r_{i-j})}_{\text{position term}}. \tag{5.7}$$

where $v, u \in \mathbb{R}^D$ are two trainable bias vectors, $W_Q, W_K, W_R \in \mathbb{R}^{D\times D}$ are three trainable projection matrices, and $r_{i-j} \in \mathbb{R}^D$ is the sinusoidal positional encoding that represents the relative distance $i-j$ between the two positions.

To compute the entire attention score matrix $\mathbf{A}$, the content term can easily be obtained via two head projections and an *outer product* of complexity $O(TD^2 + T^2D)$:

$$\mathbf{A}^{\text{content}} = (\mathbf{H}W_Q + v)(\mathbf{H}W_K)^\top,$$

where $\mathbf{H} = [h_1, \cdots, h_T] \in \mathbb{R}^{T\times D}$ collects all hidden states into a matrix. However, we cannot compute the position term in the same way as each A_{ij}^{position} corresponds to a different r_{i-j}. Hence, a naive solution will be stacking T^2 pairs of position encodings into a tensor $\hat{\mathbf{R}} \in \mathbb{R}^{T\times T\times D}$ where $\hat{\mathbf{R}}_{ij} = r_{i-j}$, and then perform the following tensor product:

$$\mathbf{A}^{\text{position}} = \texttt{einsum("id,ijd->ij"}, \mathbf{H}W_Q + u, \hat{\mathbf{R}}W_R).$$

Note that the head projection $\mathbf{R}W_K$ now has a complexity of $O(T^2D^2)$ and a memory footprint of $O(T^2D)$, dominating all other computations.

Standard Solution: Gather / Shift

To resolve the computation burden above, a common technique is to instead collect a matrix $\mathbf{R} \in \mathbb{R}^{2T-1\times D}$, where

$$\mathbf{R} = [r_{T-1}, \ldots, r_0, \cdots, r_{1-T}]$$

which includes all possible position encodings arranged from the maximum possible distance value $T-1$ to the minimum one $1-T$. Note that the full $\hat{\mathbf{R}}$ can be formed by gathering specific elements from $\mathbf{R}$ with an index matrix $\mathbf{I}$ of shape $[T \times T]$, i.e.,

$$\hat{\mathbf{R}} = \text{gather}(\mathbf{R}, \mathbf{I}), \quad I_{ij} = T + i - j.$$

Mathematically, this is equivalent to using a permutation tensor $\mathbf{P} \in \mathbb{R}^{T \times T \times 2T-1}$ to multiply $\mathbf{R}$, i.e., $\hat{\mathbf{R}} = \mathbf{PR}$, where $\mathbf{P}_{ij} \in \mathbb{R}^{2T-1}$ is a one-hot vector used to select/gather a single position of $\mathbf{R}$. As the attention score computation only involves linear operations, we can rearrange the computation of the position term as follows

$$\begin{aligned}
\mathbf{A}^{\text{position}} &= \texttt{einsum}(\texttt{"id,ijd->ij"}, \mathbf{H}W_Q + u, \textbf{(PR)}W_R) \\
&= \texttt{einsum}\Big(\texttt{"ijk,jk->ij"}, \mathbf{P}, \big[(\mathbf{H}W_Q + v)(\mathbf{R}W_R)^\top\big]\Big) \\
&= \texttt{gather}\Big((\mathbf{H}W_Q + v)(\mathbf{R}W_R)^\top, \mathbf{I}\Big)
\end{aligned}$$

Note that, assuming gathering T^2 elements only has a complexity of $O(T^2)$, which is true for CPU/GPU, this trick reduces the computation complexity back to $O(2TD^2 + 2T^2D)$. In practice, the gather operation can be implemented via a smart reshape operation, that is even cheaper.

Optimization for TPU: factorized relative positional attention

However, on TPUs, the assumption that gathering T^2 elements only has a complexity of $O(T^2)$ does not hold. Instead, we found that such a gather operation is dramatically slower on TPU. Hence, we here consider another implementation which is significantly faster on TPU.

Firstly, let's rewrite the position term as follows

$$\begin{aligned}
A_{ij}^{\text{position}} &= (W_Q h_i + u)^\top (W_R r_{i-j}) \\
&= \Big[\underbrace{W_R^\top (W_Q h_i + u)}_{q_i}\Big]^\top r_{i-j} \\
&= q_i^\top r_{i-j}. \qquad (5.8)
\end{aligned}$$

For easier derivation, we have introduced a notation of q_i. Then, recall the r_{i-j} is the sinusoidal encoding that consists of the sine and the cosine components $r_{i-j} = \texttt{cat}(\sin_{i-j}, \cos_{i-j})$, where

$$\begin{aligned}
\sin_t &= \big[\sin\big(t/10000^{2/D}\big), \sin\big(t/10000^{4/D}\big), \cdots, \sin\big(t/10000^{D/D}\big)\big] \in \mathbb{R}^{D/2}, \\
\cos_t &= \big[\cos\big(t/10000^{2/D}\big), \cos\big(t/10000^{4/D}\big), \cdots, \cos\big(t/10000^{D/D}\big)\big] \in \mathbb{R}^{D/2}.
\end{aligned}$$

Hence, we similarly divide q_i defined above into two parts, i.e.,

$$q_i = \texttt{cat}(q_i^{\sin}, q_i^{\cos}).$$

Given the definitions, we can further break Eqn. (5.8) into two terms:

$$A_{ij}^{\text{position}} = q_i^\top r_{i-j} = {q_i^{\sin}}^\top \sin_{i-j} + {q_i^{\cos}}^\top \cos_{i-j}\,.$$

Now, using the trigonometric identities $\sin(a-b) = \sin(a)\cos(b) - \cos(a)\sin(b)$ and $\cos(a-b) = \cos(a)\cos(b) + \sin(a)\sin(b)$, the two terms can be respectively reformulated into

$$\begin{aligned} q_i^{\sin\top} \sin_{i-j} &= q_i^{\sin\top}[\sin_i \odot \cos_j - \cos_i \odot \sin_j] \\ &= q_i^{\sin\top}(\sin_i \odot \cos_j) - q_i^{\sin\top}(\cos_i \odot \sin_j) \\ &= \left[q_i^{\sin} \odot \sin_i\right]^\top \cos_j + \left[q_i^{\sin} \odot (-\cos_i)\right]^\top \sin_j \end{aligned}$$

and

$$\begin{aligned} q_i^{\cos\top} \cos_{i-j} &= q_i^{\cos\top}[\cos_i \odot \cos_j + \sin_i \odot \sin_j] \\ &= q_i^{\cos\top}(\cos_i \odot \cos_j) + q_i^{\cos\top}(\sin_i \odot \sin_j) \\ &= [q_i^{\cos} \odot \cos_i]^\top \cos_j + [q_i^{\cos} \odot \sin_i]^\top \sin_j \end{aligned}$$

Hence, combining these two parts together, it follows that

$$\begin{aligned} q_i^\top r_{i-j} &= q_i^{\sin\top} \sin_{i-j} + q_i^{\cos\top} \cos_{i-j} \\ &= \left[q_i^{\sin} \odot \sin_i\right]^\top \cos_j + \left[q_i^{\sin} \odot (-\cos_i)\right]^\top \sin_j + [q_i^{\cos} \odot \cos_i]^\top \cos_j + [q_i^{\cos} \odot \sin_i]^\top \sin_j \\ &= \left\{\left[q_i^{\sin} \odot \sin_i\right]^\top \cos_j + [q_i^{\cos} \odot \cos_i]^\top \cos_j\right\} + \left\{\left[q_i^{\sin} \odot (-\cos_i)\right]^\top \sin_j + [q_i^{\cos} \odot \sin_i]^\top \sin_j\right\} \\ &= \Bigg[\underbrace{\texttt{cat}(q_i^{\sin}, q_i^{\cos})}_{=q_i} \odot \underbrace{\texttt{cat}(\sin_i, \cos_i)}_{:=\phi_i}\Bigg]^\top \underbrace{\texttt{cat}(\cos_j, \cos_j)}_{:=\psi_j} \\ &\quad + \Bigg[\underbrace{\texttt{cat}(q_i^{\sin}, q_i^{\cos})}_{=q_i} \odot \underbrace{\texttt{cat}(-\cos_i, \sin_i)}_{:=\pi_i}\Bigg]^\top \underbrace{\texttt{cat}(\sin_j, \sin_j)}_{:=\omega_j} \\ &= [q_i \odot \phi_i]^\top \psi_j + [q_i \odot \pi_i]^\top \omega_j, \end{aligned}$$

where $\phi_i, \psi_j, \pi_i, \omega_j$ above are simply 4 positional encodings formed by concatenating the cosine and sine vectors of the corresponding i and j in different ways. Note that, each term of the last line has a *factorized* form that can be computed via an *outer product*, just like the standard content term. Therefore, by stacking $\phi_i, \psi_j, \pi_i, \omega_j$ of all positions (i.e. $i = 1, \ldots, T$ and $j = 1, \ldots, T$) into the corresponding $\mathbf{\Phi}, \mathbf{\Psi}, \mathbf{\Pi}, \mathbf{\Omega} \in \mathbb{R}^{T \times D}$ respectively, the full position term can be expressed in a simple form

$$\mathbf{A}^{\text{position}} = \Big\{\big[(\mathbf{H}W_Q + u)W_R^\top\big] \odot \mathbf{\Phi}\Big\}\mathbf{\Psi}^\top + \Big\{\big[(\mathbf{H}W_Q + u)W_R^\top\big] \odot \mathbf{\Pi}\Big\}\mathbf{\Omega}^\top$$

which leads to the complexity of $O(2TD^2 + 4T^2D)$, which is comparable to the content term.

5.3.3 Potential Model Extensions

In this section, we discuss some potential model extensions of Funnel-Transformer. As described in section 5.2, Funnel-Transformer can be divided into an encoder with a compression functionality and a decoder that recovers the full-length token-level representations. To further extend the

proposed model, first note that the encoder-decoder framework can be formulated into a more general form:

$$\mathbf{h}_{\text{enc}} = \text{Encoder}(\mathbf{x}_{\text{enc}}),$$
$$\mathbf{h}_{\text{dec}} = \text{Decoder}(\mathbf{h}_{\text{enc}}, \mathbf{x}_{\text{dec}}),$$

where $\mathbf{x}_{\text{enc}}$ and $\mathbf{x}_{\text{dec}}$ are the encoder input sequence and the *optional* and *problem-specific* decoder input, respectively. The goal of encoder is to compressing the input sequence $\mathbf{x}_{\text{enc}}$ into the hidden representations $\mathbf{h}_{\text{enc}}$ with a reduced length. Then, conditioned on the decoder input $\mathbf{h}_{\text{enc}}$ if any, the decoder will extract relevant information/representations from $\mathbf{h}_{\text{enc}}$ to solve the specific NLP problem at hand. Next, we will how the general form of Funnel-Transformer can be instantiated into specific forms to solve corresponding NLP problems.

Sequence-level prediction This is essentially the case we consider in most of our experiments where we want to obtain a vectorial representation of the input sequence such as text classification. In this case, we don't really need the decoder $\mathbf{x}_{\text{dec}}$ (i.e. $\mathbf{x}_{\text{dec}} = \varnothing$) and the decoder simply extracts the hidden representation corresponding to the `[cls]` token from $\mathbf{h}_{\text{enc}}$ and feeds it into the task-specific structure (e.g. classifier).

Token-level prediction In the token-level prediction tasks such as the MLM pretraining, SQuAD and sequence labeling, we need a decoder to recover the token-level representations from the compressed sequence $\mathbf{h}_{\text{enc}}$. In many cases, $\mathbf{x}_{\text{dec}}$ could simply be the original sequence or a token-level hidden representation of it to provide fine grained low-level information of each token and hence ease the optimization. In this thesis, we utilize the last-layer hidden states of the 1st block (before the first pooling operation) as the additional decoder input.

But for problems that utilize additional input signals, such as the permutation order used for permuted language modeling in XLNet [121]. This additional information can be injected into Funnel-Transformer via the decoder input $\mathbf{x}_{\text{dec}}$ to (approximately) recover some more complex control of attention mechanism.

Sequence-to-sequence problems Another important category of NLP task is sequence-to-sequence problems, including machine translation, text summarization, and dialog generation, whose state-of-the-art solution is the conventional encoder-decoder framework. Hence, Funnel-Transformer naturally fits these tasks, where the decoder input x_{dec} corresponds to the target text sequence and the encoder input x_{enc} the source text sequence. This way, the key difference compared to conventional models is the source side compression Funnel-Transformer provides.

Overall, we summarize some potential directions to extend Funnel-Transformer presented in section 5.2.2 to NLP problems. Finally, although we focus on discussion on the NLP tasks in this thesis, Funnel-Transformer could be applied to any tasks dealing with sequential data, such as time series and video stream analysis.

5.4 Experiment

In this section, we empirically evaluate the proposed F-TFM by first pretraining it and then finetuning it in downstream tasks. Following previous work, for pretraining, we consider two common settings:

- **Base scale**: Pretraining models for 1M steps with batch size 256 on Wikipedia + Book Corpus. This is the setting used by original BERT [30]. We will rely on this setting to perform fair comparison between F-TFM and the standard Transformer as well as some ablation studies.
- **Large scale**: Pretraining models for 500K steps with batch size 8K on the five datasets used by XLNet [121] and ELECTRA [23] (Wikipedia + Book Corpus + ClueWeb + Gigaword + Common Crawl). We will compare F-TFM trained at this scale with previous state-of-the-art methods.

For finetuning, we mainly focus on sequence-level tasks that only requires a single vectorial representation of the input sequence, since F-TFM is designed with such a purpose in mind. Specifically, such tasks include the GLUE benchmark for language understanding [113], 7 widely used text (sentiment / topic) classification tasks (IMDB, AD, DBpedia, Yelp-2, Yelp-5, Amazon-2, Amazon-5) [127], and the RACE reading comprehension dataset [61]. In addition, to see how F-TFM performs when token-level prediction is needed, we consider the SQuAD question answering task which requires the model to select a token span from the context paragraph as the answer.

Finally, for all models implemented in this work including Transformer baselines in the base-scale comparison section 5.4.1, we always use the relative positional attention parameterization proposed by Transformer-XL [26] (see Appendix 5.3.2 for some implementation details of Transformer-XL).

5.4.1 Base-scale Results

Firstly, we evaluate how F-TFM performs compared to the standard Transformer under similar amount of computation (i.e., FLOPs). For this purpose, we consider three commonly used model sizes for the standard Transformer, namely large (L24H1024), base (L12H768) and small (L6H768). Then, for each Transformer baseline, we construct F-TFMs of different block layouts and parameters, while ensuring the F-TFMs always have fewer or similar FLOPs. Based on the MLM pretraining objective, the results on GLUE benchmark and text classification are presented in Table 5.1, where we also include the *relative* FLOPs and #Params. Here, we can make a few key observations:

- Given similar or fewer FLOPs, by trading sequential resolution for more layers, the F-TFM outperforms the standard Transformer in most tasks except STS-B, especially for smaller models.
- When we only compress the sequence length without increasing the depth (and #Params), F-TFM could suffer from some performance loss in certain settings on the GLUE datasets. However, as the model size increases, such performance gaps become smaller or even disappear.
- In addition, we find partial parameter-sharing often harms the performance. Therefore, the practical trade-off should be made according to the actual task and computation device.

Model size	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE-AVG
L24H1024	63.2	94.8	91.8/88.5	91.1	88.7/91.7	88.7	94.0	80.5	86.6
B10-10-10	64.8	95.0	92.5/89.5	90.7	88.6/91.5	88.9	94.0	81.5	**87.0**
B8-8-8	63.5	94.7	92.2/89.0	90.7	88.9/91.7	88.8	93.6	81.2	86.7
L12H768	60.5	93.0	92.2/89.0	89.4	88.1/91.2	86.0	92.2	73.6	84.4
B6-6-6	62.5	94.0	92.2/89.0	89.5	88.4/91.4	87.0	92.7	76.5	**85.3**
B6-3x2-3x2	60.5	93.6	92.4/89.2	89.4	88.2/91.3	86.4	92.5	75.0	84.7
B4-4-4	59.1	92.7	91.8/88.7	89.1	88.2/91.3	85.5	92.0	73.2	83.9
L6H768	55.2	91.5	91.1/87.8	88.1	87.2/90.6	82.7	90.0	64.6	81.3
B3-4-4	59.0	92.8	91.8/88.5	88.5	87.8/90.9	84.8	91.8	73.2	**83.7**

Model size	IMDB	AG	DBpedia	Yelp2	Yelp5	Amazon2	Amazon5	FLOPs	#Params
L24H1024	4.440	**4.987**	0.646	1.758	28.73	2.409	32.78	1.00x	1.00x
B10-10-10	**4.404**	5.026	**0.617**	**1.734**	**28.52**	**2.400**	**32.65**	0.73x	1.22x
B8-8-8	4.552	5.079	0.664	1.713	28.84	2.438	32.87	0.58x	1.00x
L12H768	5.328	5.184	0.663	2.013	29.35	2.571	33.14	1.00x	1.00x
B6-6-6	**4.908**	**5.079**	0.654	**1.939**	**29.03**	**2.518**	**32.91**	0.88x	1.39x
B6-3x2-3x2	5.144	5.342	**0.649**	1.892	**29.03**	2.570	33.01	0.88x	1.00x
B4-4-4	5.348	5.250	0.670	1.979	29.37	2.596	33.16	0.58x	1.00x
L6H768	6.252	5.421	0.697	2.203	30.33	2.801	33.69	1.00x	1.00x
B3-4-4	**5.520**	**5.342**	**0.670**	**2.042**	**29.51**	**2.603**	**33.16**	1.00x	1.53x

Table 5.1: MLM pretraining results at the base scale: GLUE dev *performances (the higher the better)* in the upper panel and text classification *error rates (the lower the better)* in the lower panel . The FLOPs and #Params both refer to the finetuning setting with only the encoder. The FLOPs is a rough estimation assuming linear complexity w.r.t. the sequence length. The #Params is exact including the embedding matrix.

To further test generality of F-TFM, we additionally consider ELECTRA for pretraining. The results are summarized in Table 5.2. Overall, we see a similar trend, though the gain is slightly smaller on the GLUE benchmark. This could be attributed to reusing two key hyper-parameters (discriminator loss coefficient and generator size multiplier) tuned for Transformer to train F-TFMs without any adjustment at all.

Running Time Comparison While FLOPs count offers a general idea of the model speed, it still differs from the actual running time, especially when other overhead exists.

5.4.2 Large-scale Results

Given the encouraging results of F-TFM at base-scale, we next consider training F-TFM under the large-scale setting and compare it with previous models pretrained in similar settings. Due to the slightly better performance of ELECTRA over MLM, we will use the ELECTRA objective for all large-scale experiments. Given the pretrained F-TFM of different sizes, we first compare the finetuning performance on the GLUE benchmark in Table 5.3. Similar to the base-scale

Model size	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	GLUE-AVG
L24H1024	66.5	94.3	92.8/90.0	91.5	89.6/92.2	89.4	94.1	84.5	87.8
B10-10-10	68.6	95.0	93.0/90.0	91.0	88.9/91.7	89.1	93.6	84.5	**87.9**
B8-8-8	66.6	94.8	92.6/89.7	90.7	88.8/91.7	89.0	93.6	82.1	87.3
L12H768	64.3	93.1	92.1/89.2	90.8	88.7/91.7	86.4	92.1	75.4	85.4
B6-6-6	64.3	94.2	92.8/89.7	90.1	88.7/91.6	87.4	92.5	78.3	**86.0**
B6-3x2-3x2	63.9	94.2	93.0/90.2	89.5	88.4/91.4	87.0	92.2	77.6	85.7
B4-4-4	62.8	93.6	92.5/89.2	89.2	88.4/91.3	86.0	91.6	74.3	84.8
L6H768	62.1	91.1	90.8/86.8	88.9	88.2/91.3	83.9	89.7	66.7	82.6
B3-4-4	59.0	93.1	90.8/87.5	88.7	88.1/91.0	85.8	91.1	72.5	**83.6**

Model size	IMDB	AG	DBpedia	Yelp2	Yelp5	Amazon2	Amazon5	FLOPs	#Params
L24H1024	4.724	**5.053**	0.653	1.874	28.84	2.425	32.85	1.00x	1.00x
B10-10-10	**4.324**	5.250	**0.639**	**1.789**	**28.68**	**2.419**	**32.72**	0.73x	1.22x
B8-8-8	4.364	5.408	0.651	1.729	28.76	2.447	32.85	0.58x	1.00x
L12H768	5.248	5.355	0.657	1.953	29.24	2.596	33.04	1.00x	1.00x
B6-6-6	**4.792**	**5.237**	**0.650**	**1.850**	**28.73**	**2.499**	**32.79**	0.88x	1.39x
B6-3x2-3x2	4.924	5.342	0.671	1.913	29.00	2.523	32.85	0.88x	1.00x
B4-4-4	5.152	5.382	0.659	2.032	29.33	2.566	33.03	0.58x	1.00x
L6H768	6.220	5.395	0.674	2.287	30.16	2.759	33.57	1.00x	1.00x
B3-4-4	**5.396**	**5.342**	**0.653**	**2.000**	**29.60**	**2.591**	**33.09**	1.00x	1.53x

Table 5.2: ELECTRA pretraining results at the base scale.

results, with fewer or comparable FLOPs, F-TFM outperforms the corresponding baselines in the majority of tasks, suggesting the good scalability of F-TFM. We also test the models on the 7 text classification tasks.

Next, we consider the RACE dataset, which is quite different from the GLUE benchmark. At the core, RACE is a multiple-choice reading comprehension task requiring complex reasoning, which though, can be formulated as classifying the correct choice. Also, paragraphs in RACE are much longer. To F-TFM, this presents both a challenge, as it requires detailed reasoning, and an opportunity to compress long paragraph. As we can see in Table 5.4, F-TFM achieves better performances compared to all previous models. In particular, within the base model group, the gain is very significant. It shows that F-TFM can also excel for sequence-level task that involves long text and reasoning.

Finally, although F-TFM is mainly designed for tasks that only require a sequence-level representation, it is possible to apply F-TFM to token-level tasks by additionally finetuning the decoder. To test this ability, we finetune F-TFM on the SQuAD datasets and compare it with previous models in Table 5.5. While F-TFM outperforms previous models in the base group by a large margin, in the large model group, the F-TFM with about 83% FLOPs (B10-10-10) still falls behind the standard Transformer that always maintains a full-length token-level representations. This suggests sequential compression could harm the performance when detailed token-level information is critical. On the other hand, compared to the results on SQuAD1.1, F-TFMs perform

Model	CoLA	SST-2	MRPC	STS-B	QQP	MNLI	QNLI	RTE	WNLI	AVG
Dev set results (single model)										
$\text{ROBERTA}_{\text{Large}}$ [77]	68.0	96.4	-/90.9	92.4	-/92.2	90.2	94.7	86.6	-	88.9
$\text{XLNet}_{\text{Large}}$ [121]	69.0	97.0	-/90.8	92.5	-/92.3	90.8	94.9	85.9	-	89.2
$\text{ELECTRA}_{\text{Large}}$ [23]	69.1	96.9	-/90.8	92.6	-/92.4	90.9	95.0	88.0	-	89.5
B10-10-10H1024	72.4	96.8	93.5/90.9	92.1	89.8/92.4	91.1/-	95.1	89.5	-	**90.0**
B8-8-8H1024	71.3	96.8	93.1/90.7	91.7	89.8/92.4	90.8/-	94.7	89.2	-	89.7
$\text{ROBERTA}_{\text{Base}}$ [77]	63.6	94.8	-/90.2	91.2	-/91.9	87.6/-	92.8	78.7	-	86.4
$\text{MPNet}_{\text{Base}}$ [100]	65.0	95.4	-/91.5	90.9	-/91.9	88.5/-	93.3	85.2	-	87.7
B6-6-6H768	70.1	96.3	93.2/90.4	91.1	89.2/92.0	89.7/-	93.7	83.4	-	**88.3**
B6-3x2-3x2H768	68.5	95.6	92.5/89.5	91.0	89.3/92.0	89.1/-	93.0	83.4	-	87.8
B4-4-4H768	68.2	95.0	92.8/90.2	90.3	89.0/91.8	88.6/-	92.6	79.1	-	87.0
Leaderboard test set results (single task & single model)										
$\text{ELECTRA}_{\text{Large}}$ [23]	68.1	96.7	89.2/92.0	92.1/91.7	74.8/90.4	90.7/90.2	95.5	86.1	65.1	85.2
B10-10-10H1024	68.9	97.2	89.4/92.1	91.6/91.3	74.3/90.2	90.9/90.9	95.5	86.5	65.1	**85.4**
B8-8-8H1024	68.3	96.9	89.2/92.0	91.5/91.1	73.8/90.1	90.7/90.7	95.1	85.3	65.1	85.0
$\text{ELECTRA}_{\text{Base}}$ [23]	64.6	96.0	88.1/91.2	91.0/90.2	73.2/89.5	88.5/88.0	93.1	75.2	65.1	82.7
B6-6-6H768	68.3	96.5	89.1/91.9	90.6/89.9	73.3/89.9	89.7/89.4	94.0	80.4	65.1	**84.0**
B6-3x2-3x2H768	65.9	96.0	87.8/91.0	90.0/89.6	73.3/89.8	88.9/88.7	93.8	79.9	65.1	83.4
Leaderboard test set results (multi-task & ensemble)										
$\text{ROBERTA}_{\text{Large}}$ [77]	67.8	96.7	89.8/92.3	92.2/91.9	74.3/90.2	90.8/90.2	95.4	88.2	89.0	88.1
$\text{ELECTRA}_{\text{Large}}$ [23]	71.7	97.1	90.7/93.1	92.9/92.5	75.6/90.8	91.3/90.8	95.8	89.8	91.8	89.4
B10-10-10H1024	70.5	97.5	91.2/93.4	92.6/92.3	75.4/90.7	91.4/91.1	95.8	90.0	94.5	**89.7**

Table 5.3: Comparison with previous methods on the GLUE benchmark under large-scale pre-training.

Model	RACE		
	Total	High	Middle
$\text{ROBERTA}_{\text{Large}}$ [77]	83.2	81.3	86.5
$\text{XLNet}_{\text{Large}}$ [121]	85.4	84.0	88.6
B10-10-10	**85.7**	**84.4**	**88.8**
B8-8-8	85.2	83.9	88.4
$\text{ALBERT}_{\text{Base}}$ [65]	66.0	-	-
$\text{MPNet}_{\text{Base}}$ [100]	72.0	76.3	70.3
B6-6-6	**79.7**	**78.2**	**83.4**
B6-3x2-3x2	78.8	77.5	82.0
B4-4-4	76.2	74.6	80.0

Table 5.4: RACE test performance comparison.

Model	SQuAD2.0		SQuAD1.1	
	EM	F1	EM	F1
$\text{ROBERTA}_{\text{Large}}$ [77]	86.5	89.4	88.9	94.6
$\text{ELECTRA}_{\text{Large}}$ [23]	**88.0**	**90.6**	**89.7**	**94.9**
B10-10-10	87.6	90.4	89.0	94.7
B8-8-8	87.1	89.8	88.7	94.4
$\text{ROBERTA}_{\text{Base}}$ [77]	80.5	83.7	84.6	91.5
$\text{MPNet}_{\text{Base}}$ [73]	80.5	83.3	86.8	92.5
B6-6-6	**85.1**	**87.7**	**87.4**	**93.3**
B6-3x2-3x2	84.2	87.0	87.0	93.0
B4-4-4	82.6	85.5	85.9	92.2

Table 5.5: SQuAD dev performance comparison.

relatively better on SQuAD2.0, which additionally requires the model to make a sequence-level

prediction on whether the question is answerable. This again shows the general effectiveness of the F-TFM in sequence-level tasks.

5.4.3 Ablation Study

ID	Layout	(FLOPs / Params)	Pool-Op	Pool-query-only	Sep `[cls]`	Rel-Attn	GLUE-AVG
(1)	B6-6-6	(1.00x / 1.00x)	Mean	✓	✓	✓	**83.5**
(2)			Mean	✓		✓	82.9
(3)			Mean		✓	✓	83.0
(4)			Mean	✓	✓		81.4
(5)			Max	✓	✓	✓	83.4
(6)			Top-Attn	✓	✓	✓	75.8
(7)	B8-8	(1.14x / 0.91x)	Mean	✓	✓	✓	83.4
(8)	B5-5-5-5	(0.89x / 1.08x)	Mean	✓	✓	✓	82.9

Table 5.6: Ablation study of F-TFMs with different designs.

Finally, based on the GLUE benchmark, we perform a series of ablation studies on the importance of various designs in F-TFM, including the block layout design, the type of pooling operation, the pool-query-only technique, maintaining a separate `[cls]` vector and the usage of Transformer-XL parameterization.

- Pooling operation: Including the mean pooling we finally employ in F-TFM, we actually test two types of pooling operations.
  (1) The first type is just the strided mean/max pooling as described in section 5.2.
  (2) The second type aims to select a subset of "hub" states, which refer to those hidden vectors that are attended most in the previous S-Attn layer and hence likely to carry most critical information about the sequence. Concretely, given the attention map from the previous S-Attn layer, we reduce sum the scores along the number of head and query length dimensions to a score for each position. Then, we simply choose the top 50% of states to achieve the same compression rate. Note that, this type of pooling operation is essentially the same as the important states selection procedure in Power-BERT [39].
- Pool-query-only design
- Separating `[cls]` in the pooling operation
- Block layout design: In our experiments, all models actually utilize a 3-block design. Here, we compare the 3-blocks design with the 2-blocks and the 4-blocks design.
- Relative attention parameterization proposed in Transformer-XL [26]. We compare this parameterization with the learned absolute position embedding as used in the BERT [30].

The ablation results are included in Table 5.6. To save the computation resources, the size of model hidden states in table 5.6 is set as 512. From the ablation results, we can make the following observations:

- Comparing pooling different operation ((1), (5), and (6)), we found that the performance of the mean and max pooling operation is similar. But they are significantly better than the idea of utilizing attention score (Top-Attn pooling) to select the "hub" states.
- Comparing (1) with (2) and (3) respectively, we see that the two special designs, i.e. "pool-query-only" and maintaining a separate non-pooled `[cls]` , can both bring a clear improvement to the proposed model.
- Comparing (1) and (4), we find that the relative positional parameterization is key to the performance of the proposed F-TFM. We suspect that the pooling operation could destroy the positional information carried by the absolute position encoding, which is only injected to the model in the input embedding layer. As a result, the higher blocks may not have enough positional information to learn a good enough attention pattern. In comparison, the positional information is injected to each layer under the relative positional attention scheme. Therefore, to achieve good result with F-TFM based on absolute positional embedding, one may inject the absolute positional embedding into each attention layer. Actually, a contemporary application of Transformer to the detection problem in computer vision shows injecting positional embedding into each layer is important [15].
- Finally, we study the influence of block layout design in our framework. With B6-6-6 as the 3-block benchmark, we consider two other layout design with similar FLOPs and number of parameters. Specifically, we consider B8-8 for the 2-block design and B5-5-5-5 for the 4-block design. Comparing the results in (1), (7), and (8), we find that the performance of the 3-block (B6-6-6) design achieves the best performance, which is significantly better than the 4-block design and slightly better than the 2-block design. However, if we further taking the FLOPs/#Params into consideration, it is more clear that the 3-block design is superior. Therefore, in the this thesis, we always use the 3-block design.

5.4.4 Training Cost Comparison

In this section, we test the pretraining and finetuning speed of the F-TFM in comparison to the standard Transformer on the TPU and GPU platform. For the pretraining speed evaluation, we test F-TFM on TPU v3-16 (16 cores x 16Gb) with TensorFlow. For the finetuning speed evaluation, we test F-TFM on TPU v2-8 (8 cores x 8Gb) with TensorFlow and on Nvidia-V100 (16Gb) GPU with the PyTorch. The TensorFlow version is 2.2.0, and the PyTorch version is 1.5.0. For the GPU experiments, we use an 8-GPU node on the Google Cloud Platform. All running speeds are reported with the FP16 optimizer. In the PyTorch implementation, we use "O2" options of AMP manager in the apex[2] package to handle the FP16 optimization. For finetuning, we consider three different sequence lengths, namely 128, 256 and 512. For pretraining, we only consider the sequence length 512. In each case, we choose the maximum possible batch size allowed by the memory size of the device(s). We measure the actual model *running time* by performing 1000 steps gradient descent with random input sequences with the fixed length.

Firstly, we compare the model speed in the finetuning stage. Note that the decoder is not used in this setting. Table 5.7 and 5.8 summarize the finetuning running time comparison on GPUs

[2] `https://github.com/NVIDIA/apex`

Sequence length	128			256			512		
Metrics	Run time 1 GPU	Run time 8 GPUs	Mem	Run time 1 GPU	Run time 8 GPUs	Mem	Run time 8 GPUs	Mem	GLUE
Batch size / GPU	64			32			16		
L12H768	1.00x	1.00x	9.2G	1.00x	1.00x	11.0G	1.00x	14.3G	84.40
B6-6-6	0.97x	0.99x	9.1G	0.95x	0.97x	10.3G	0.94x	12.5G	85.37
B6-3x2-3x2	0.93x	0.93x	8.4G	0.91x	0.92x	9.5G	0.90x	11.8G	84.78
B4-4-4	0.67x	0.67x	6.6G	0.65x	0.66x	7.5G	0.64x	9.0G	83.99
Batch size / GPU	32			12			4		
L24H1024	1.00x	1.00x	14.8G	1.00x	1.00x	14.4G	1.00x	13.9G	86.62
B10-10-10	0.87x	0.92x	14.0G	0.90x	0.93x	13.0G	0.96x	12.7G	87.03
B8-8-8	0.70x	0.73x	11.6G	0.73x	0.75x	10.8G	0.78x	10.5G	86.70

Table 5.7: Running time and memory consumption comparison between F-TFMs and the standard Transformer on the GPU. In each model group, the standard Transformer (first model) is used as the benchmark for the rest of F-TFM models. Note that, given the same batch size per GPU, the memory consumption is roughly the same for 1 GPU and 8 GPUs.

Sequence length	128	256	512	
Metrics	Run time on 8 TPU cores (TPUv2-8)			GLUE
Batch size / TPU core	64	32	16	
L12H768	1.00x	1.00x	1.00x	84.40
B6-6-6	0.99x	0.88x	0.81x	85.37
B6-3x2-3x2	0.97x	0.87x	0.77x	84.78
B4-4-4	0.69x	0.62x	0.55x	83.99
Batch size / TPU core	16	8	4	
L24H1024	1.00x	1.00x	1.00x	86.62
B10-10-10	0.89x	0.81x	0.73x	87.03
B8-8-8	0.66x	0.60x	0.56x	86.70

Table 5.8: Running time between F-TFMs and the standard Transformer on the TPU v2-8. In each model group, the standard Transformer (first model) is used as the benchmark for the rest of F-TFM models.

and TPUs, respectively.

- In the base model (L12H768) group, we observe that the speed of B6-6-6H768 is similar or faster than the base Transformer model, despite the fact that B6-6-6 is deeper, has more parameters. Moreover, B6-6-6H768 achieves better results compared with the base Transformer model. The similar conclusion applies to the B6-3x2-3x2 model, which has the same amount

of parameters as the base model. The B4-4-4 model, which has the same depth and model parameters as the base model, is able to provide 30%-50% speedup without losing too much performance.

- In the large model (L24H1024) group, the conclusion is similar. The speed of the larger model B10-10-10 is almost the same as the large model, and the speed of B8-8-8 is significantly faster than the large model. In addition, when sequence length equals 512, the acceleration of F-TFM on the TPU is more obvious than the GPU.
- In the both groups, all the tested F-TFM variants have smaller memory footprint compared with the standard TFM models, showing the memory efficiency of F-TFM.

Next, we compare the model speed during pretraining under the MLM objective in table 5.9, which has an additional cost due to the decoder. The results show that the proposed method can still substantially improve the pretraining speed compared to the standard Transformer, though the speed gain is slightly smaller than the finetuning stage. In summary, this study demonstrates that the proposed method is more efficient in both the finetuning and pretraining stages in modern parallel computing platforms.

Sequence Length	512	
	Running Time	FLOPs
#TPU cores / Total bsz	16 / 512	
L12H768	1.00x	1.00x
B6-6-6H768D2	0.99x	1.04x
B6-3x2-3x2H768D2	0.97x	1.04x
B4-4-4H768D2	0.79x	0.75x
#TPU cores / Total bsz	16 / 128	
L24H1024	1.00x	1.00x
B10-10-10H1024D2	0.83x	0.81x
B8-8-8H1024D2	0.71x	0.66x

Table 5.9: TPU pretraining speed comparison. The suffix "D2" means that the F-TFM model has 2 decoder layers.

5.5 Conclusion & Discussion

In this work, under the pretraining-finetuning paradigm, we investigate a largely overlooked dimension of complexity in language processing. With the proposed Funnel-Transformer, we show how sequential resolution can be compressed in a simple form to save computation and how the saved FLOPs can be re-invested in improving the model capacity and hence the performance. Open challenges for future research include the better ways to improve the compression scheme, to optimize the block layout design and to re-invest the saved FLOPs. In addition, combining

Funnel-Transformer with model compression techniques like knowledge distillation and quantization would be an important direction towards the enhancement of practical impact.

Chapter 6

Event Temporal Modeling with Sparse Labels

6.1 Motivation

An important task in temporal sequence modeling is to automatically extract and summarize the semantic information from news stories for the events of interest. Although topic detection and tracking (TDT) [2] has been studied for decades by the information extraction (IE) [74] and information retrieval (IR) [78], how to effectively predict the semantic relationships among events, especially how to use cutting-edge neural network technologies to improve temporal and causal reasoning, are still open challenges for research [106].

This chapter focuses on the event-level temporal relationship classification problem [84, 106]. The task is to take the context (natural language text) and an events pair as input and predict the temporal relationship label of the event pair. A lot of methods have been proposed for this task in the past decades. They roughly can be split into 3 classes.

1. Rule Based System: The CAEVO [16] is the representative one among the rule based systems. It classifies the temporal relationship of the event pairs according to a set of predefined rules, including comparing the timestamp of two events and checking the preposition that bridging two events.
2. Point Process: The point process method views an event as a point of a point stream and uses the Bayesian process to describe the distribution of points in this stream over time. The Hawkes Process [45] is the most famous one of them, which defines the point process as a mixture of self-exciting exponential processes. However, the point process doesn't consider the context of events, which limits its performance.
3. Neural Network Approaches: Since the neural network achieves state-of-the-art performance in various domains, the researchers have applied it to this problem [84] and achieve good performance. They also combine neural networks with the Hawkes process to propose the neural Hawkes process [80]. Then the pretrained language model [30] is proposed, which learns natural language prior knowledge from the large-scale unlabeled data. The researchers have shown that we can achieve better results by leveraging the pretraining models [86].

To better understand the bottleneck of current approaches, we first study and compare the performance of the above methods. In our experiments, we found that the biggest challenge of this task is the label sparsity issue, which leads neural network models to quick overfitting. To solve this problem, we propose a more effective learning strategy for finetuning the Roberta model. The proposed strategy can be split into three parts.

1. Regularization by the Human Prior Knowledge: The event temporal sequence has two human prior knowledge about the sequence structure, i.e. symmetry and transitivity. We introduce this human prior to the machine learning model by adding them as the regularization in the loss function.
2. Data augmentation: Another popular method to prevent model overfitting is data augmentation [25]. Here we design a data augmentation method for the temporal event sequence to produce pseudo labeled training instances.
3. Utilizing unlabeled data: Although the annotated event temporal relationships are sparse, there are almost unlimited news documents available, which contain plenty of events in various domains. We utilize the large-scale unlabeled data by the advanced self-training technique [69] and semi-supervised learning over data augmentation graph [117] to improve the model robustness.

Finally, our experiment results show that, by incorporating the proposed improvements, we are able to greatly improve the performance of the model and achieve the new state-of-the-art results for the event temporal classification task.

6.1.1 Problem Formulation and Notation

Next, we are going to introduce the problem formulation and mathematical notation used by the rest of this chapter. The input includes a context $\mathbf{x} = \{x_1, x_2, \cdots, x_T\}$ and a pair of event (e_1, e_2), where the context is the natural language paragraph with T tokens, and the event pair contains two sets of tokens in the context, which are represented by $e_1, e_2 \in \mathbf{x}$. Given the input, we want to classify the temporal relationship of the event pair. The label is denoted as c and the label set is denoted as C, which contains 4 labels (1) Before, (2) After, (3) Simultaneous, and (4) Vague.

6.2 Proposed Strategy

6.2.1 Regularization by the Human Prior Knowledge

To deal with the label sparsity issue mentioned in Motivation, an effective strategy is introducing human prior knowledge to machine learning models. As discussed in [11, 85, 112], we can summarize several rules about temporal relationships of an event sequence. They are

- Symmetry: Considering a event pair e_1 and e_2. if the relationship of e_1 and e_2 is "Before", the relationship of e_2 and e_1 should be "After". Similarly, "After" $\longrightarrow$ "Before", "Simultaneously" $\longrightarrow$ "Simultaneously" and "Vague" $\longrightarrow$ "Vague".

- Transitivity: Considering an event triplet e_1, e_2 and e_3. Given the (e_1, e_2) and (e_2, e_3) relationship, we can know the (e_1, e_3) relationship should belong to a candidate set. Here I give 4 examples,
  1. ("Before", "Before") $\longrightarrow$ { "Before" }
  2. ("Before", "After") $\longrightarrow$ { all categories }
  3. ("Before", "Simultaneously") $\longrightarrow$ { "Before" }
  4. ("Before", "Vague") $\longrightarrow$ { "Before", "Vague" }

  We can infer the remaining transitivity rules accordingly.

Then our goal is to introduce the prior knowledge over the pretrained model. For the symmetry rule, given a event pair e_1 and e_2, a label c and its symmetric label $c' = symmetry(c)$, we want the output distribution satisfies

$$P(c|x, e_1, e_2) = P(c'|x, e_2, e_1) \tag{6.1}$$

Then we design a corresponding loss function regularizer as

$$L_{sym} = \sum_{c \in C} |P(c|x, e_1, e_2) - P(c'|x, e_2, e_1)| \tag{6.2}$$

Similarly, for the transitivity rule, given an event triple, we want the output distribution to satisfies

$$P(c_1|x, e_1, e_2) \times P(c_2|x, e_2, e_3) \leq \sum_{c' \in C'} P(c'|x, e_1, e_3) \tag{6.3}$$

where C' is the transitivity set of the relation pair (c_1, c_2). Accordingly, we can design a regularizer as,

$$L_{trans} = \sum_{c_1, c_2 \in C} max(P(c_1|x, e_1, e_2) \times P(c_2|x, e_2, e_3) - \sum_{c' \in C'} P(c'|x, e_1, e_3), 0) \tag{6.4}$$

Finally, we combine the two regularizers with the cross-entropy loss of the classification layer, which can be written as,

$$L = L_{cross\_entory} + \lambda_s L_{sym} + \lambda_t L_{trans} \tag{6.5}$$

where λ_s and λ_t are two tunable hyperparameters. With the two regularizers, we can force the classifier to fit the human prior knowledge about the event sequence.

6.2.2 Data Augmentation to Produce Pseudo Labeled Training Instances

Another popular method to improve model robustness is data augmentation. Given a data sample (x, e_1, e_2, c), we want to design an augmentation method to produce the augmented sample $(q(x), e_1, e_2, c)$. The most popular augmentation method in natural language processing is round translation [122]. But the translation will break or drop the event information, (e_1, e_2). After trying different strategies in experiments, we design a token-level augmentation strategy by leveraging the methods proposed in [64] and [114]. The augmentation method is applying the following operators by order,

- Randomly replace words with a synonym.
- Randomly insert words. The inserted words are synonyms of one word in the original text.
- Randomly drop words.
- Randomly shuffle words. We limit the position shift to less than 2.

6.2.3 Semi-supervised Label Propagation Based on Augmented Data Graph

Next, we will discuss how to use unlabeled data to improve model performance. Besides the advantage that seeing more data will make our model more robust and prevent overfitting, unlabeled data can also help us address the domain shift problem. If the domain of test data is different from the training data, we can use the unlabeled data from the test domain, which usually are cheap, to let the model learn the prior information of the test domain. The first task is to generate the unlabeled event sequence data. We use the OpenIE system [102] to extract events from the news corpus. Given the unlabeled data, the next question is how to utilize it to improve our model. Here we test two methods, self-training, and semi-supervised learning.

The most straightforward method is the self-training method [69]. It first trains a teacher model with the human-annotated data, then produces the pseudo-labels on the unlabeled data based on the teacher model. Finally, the model retrains a student model from scratch based on the annotated data. The self-training method is simple, but it shows strong empirical performance with careful hyperparameters tuning in various classic benchmarks [32, 118]. Its problem is that the teacher model would propagate its error to the student model, and we could not correct it.

Another solution is the semi-supervised learning method. Instead of separating the processes of annotating the unlabeled data and training the student classifier in the self-training. The semi-supervised learning model can optimize them jointly with the label propagation technique. Here, We apply the state-of-the-art semi-supervised method, Unsupervised Data Augmentation (UDA) [117]. Its core idea is building an augmented data graph that links every sample with several of its augmented samples. Then we train a model that can produce consistent prediction results on the augmented data graph. The objective function can be written as,

$$L_{UDA} = \sum_{c \in C} KL(P(c|q(x), e_1, e_2)||P(c|x, e_1, e_2)) \tag{6.6}$$

Then we also combine this loss with the cross-entropy loss from the supervised data.

$$L = \lambda_U L_{UDA} + L_{cross\_entropy} \tag{6.7}$$

where λ_U is a tunable coefficient. Notice that in both self-training and semi-supervised learning, we still can apply the human prior knowledge regularization mentioned in Section 6.2.1 in the loss function. We drop it here to keep the notation simple.

	Acc	F1	Unlabel
LSTM (Ning et al.(2018))	61.6	66.6	✗
LSTM (Goyal and Durrett (2019))	68.6	74.2	✗
Bert (Ning et al.(2019))	71.7	76.7	✗
Roberta (Ballesteros, Miguel, et al. (2020))	73.5	78.9	✗
Roberta (Ballesteros, Miguel, et al. (2020))	75.5	81.6	✓
Logistic Regression	59.74	63.85	✗
Hawkes Process	58.06	62.20	✗
LSTM	65.71	70.42	✗
Neural Hawkes Process	65.31	69.93	✗
Our Roberta	74.51	79.21	✗
Our Roberta with Proposed Learning Strategy	76.58†	81.67†	✗
+ semi-supervised learning	**77.66**†	**82.47**†	✓

Table 6.1: Experiment results on MATRES dataset. The first section contains the results from previous publications. The second section contains the results from our implementation. The "unlabeled" column is referred to as whether to use unlabeled data during the training. The results with † mean that the p-values in the significance tests (one-sample t-tests) for comparing our method with the second best (not ours) in the table are smaller than 5%.

6.3 Main Results

6.3.1 Experiment setting

In the experiment section, we focus on the MATRES dataset [84]. It is an event temporal relation dataset, whose label space is the same as the description in section 6.1.1. Its training set contains 256 contexts and 12,740 event pair relations. Its test set contains 20 contexts and 837 event pair relations. We split 10% of training data as the development set.

To evaluate the model prediction, we use two metrics, accuracy and F1 score. For the F1 score, we treat the "Vague" class as the null class. In our experiment, we always use the Roberta Base model as the pretrained language model.

For the unlabeled data used in this experiment, we use the News Crawl 2007 and 2008 from the WMT task. After extracting the events by the OpenIE system, we obtain 4M context and 40M event pairs, which is 1000x larger than the MATRES training set.

6.3.2 Baselines

To comprehensively study the performance of traditional and advanced baselines, we implement the following methods and report in our experiment results.

- Logistic Regression: Given the event pair (e_1, e_2), we use a linear classifier to predict its temporal relationship label. The linear classifier is trained by the cross-entropy function.
- Hawkes Process: Given the event pair (e_1, e_2), we use the Hawkes process to estimate the

density of different labels. Then choose the label with the highest density estimation as our prediction.

- LSTM: Given the event pair (e_1, e_2) and context, we use LSTM to produce the contextual embedding of the two events. The prediction layer and learning strategy are the same as the logistic Regression.
- Neural Hawkes Process: Similar to LSTM, we use LSTM to produce contextual embedding and use them to predict the coefficient of the Hawkes Process.
- Roberta: Similar to LSTM, replace the embedding model by the Roberta model [77].

6.3.3 Experiment Results

Here, we present experiment results of the baseline methods and the proposed methods in Table 6.1. We also include the results reported in previous publications as the reference. First, Compared with the model with softmax function output distribution, Logistic Regression, and LSTM, Hawkes process family models, Hawkes Process and Neural Hawkes Process, didn't provide improvements. We hypothesize the main reason is the strong assumption about the exponential distribution family, which did not fit our problem setting. Second, by comparing the baseline models, we found that the pretrained language model (Our Roberta) produces the best performance among the baseline method, which means that utilizing the context information is important in this task.

Then by comparing the proposed method with the best-reported results without using additional unlabeled event sequences, we observe that the proposed learning strategy significantly improves both accuracy and F1 metrics. Furthermore, when using the unlabeled data, we achieve the new state-of-the-art performance in this task, 77.66 accuracy and 82.47 F1 score. But, considering the large-scale of the unlabeled dataset, semi-supervised learning would cost much more computation resources compared to purely supervised learning. It is a trade-off between computation and accuracy. On the other hand, the cost of adding regularization and data augmentation is small due to the limit size of labeled data. Finally, the improvements over previous baselines have passed the one-tail significance test.

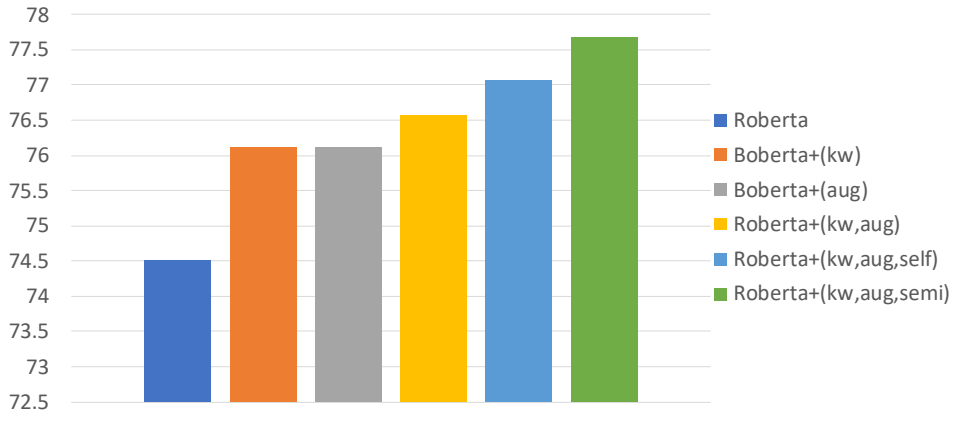
6.3.4 Ablation Study

Next, we conduct the ablation study for the proposed learning strategy. The ablation results are recorded in Figure 6.1. Based on the results, we observe that all 3 proposed improvements (introducing human prior, data augmentation, and utilizing unlabeled data) contribute to the final performance. Compared with the self-training method, the semi-supervised learning approach has better performance for utilizing the unlabeled data.

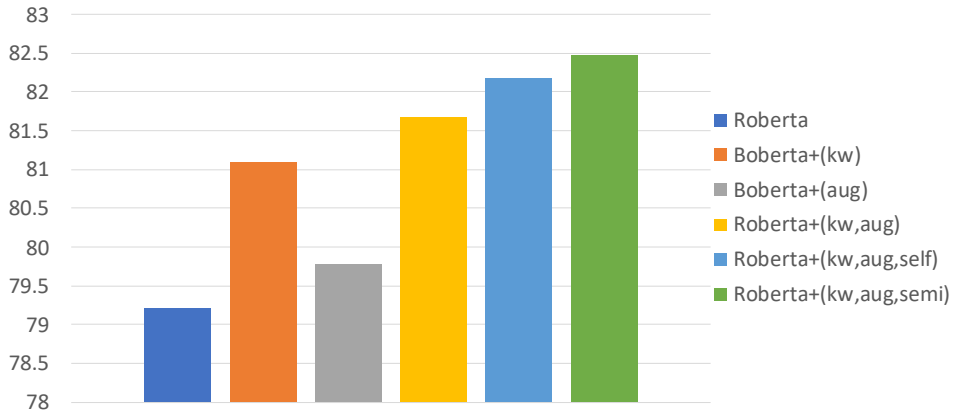
6.4 Conclusion

In this chapter, we study an important machine learning task, event temporal relationship classification. We carefully compare several kinds of important baselines. We found the main issue is

the overfitting problem caused by the data sparsity nature of this task. To alleviate that, we propose several improvements, including introducing new regularization to introduce human prior, design data augmentation to generate pseudo training labels, and utilizing the unlabeled data. Finally, in both supervised and semi-supervised settings, the proposed models improve current state-of-the-art results by a large margin.



(a) Accuracy vs method



(b) F1 score vs method

Figure 6.1: Ablation study of the proposed strategy. "kw" means using regularization to introduce human prior. "aug" means using data augmentation to generate pseudo labels. "self" means using self-training. "semi" means using semi-supervised learning.

Chapter 7

Conclusion

This thesis focuses on sequential data, which is one of the most important types of data in real-world applications, including natural language processing, speech analysis, electricity usage, prices of financial products, traffic occupation, and so on. The various types of sequential data bring us different challenges. For example, modeling correlation among variables in the multivariate time series modeling and the range of data value issue in the continues time series data forecasting. The main goal of this thesis is to enhance the state-of-the-art in this field by developing novel neural network frameworks and learning algorithms to solve these sequential tasks according to their unique formats and characteristics. The main contributions can be summarized from five aspects below.

1. **A novel graph convolution architecture for spatiotemporal sequence modeling:** Inspired by the great success of convolution networks on image data, we want to extend such techniques for spatiotemporal modeling. But standard 2D convolution methods require the data to have a grid-like structure, which is too restrictive for the spatiotemporal domain. To address this issue, We propose Depthwise Separable Graph Convolution (DSGC), which combines the strengths of CNN in 2D convolution and the flexibility of Graph Convolution Network (GCN), enabling automated learning of spatial correlations with irregular spatiotemporal structures. In our experiments, DSGC shows better empirical performance compared to representative Graph Convolution methods on spatiotemporal forecasting benchmarks.
2. **A factorized recurrent neural network for multivariate time series modeling:** The DSGC method utilizes spatial correlations among variables. However, how to generalize the method to the dependencies beyond spatial correlations is an open question. For example, the status of each key (being pressed or unpressed) in a piano can be treated as a temporal sequence, and the states of all the keys in a piano score form a multivariate time series, where the dependencies are more complicated than a spatial correlation. We propose to use an autoregressive prediction layer along the variables in each time step, i.e., to predict the value of the ith variable at time step t based on all values of variables with smaller index at time step t. Compared with the previous state-of-the-art methods such as adding stochastic latent variables to RNN, our approach consistently improves the density estimation results by 40% on average.

3. **Modeling periodic patterns with RNN for temporal sequence forecasting:** While neural networks have achieved good performance in the various sequential tasks, none of the previous methods can integrate the periodic prior into their neural models. On the other hand, periodic time series are common in real life, such as traffic occupation, electricity usage, and solar-energy farm output. This thesis address this modeling challenge by proposing a Recurrent-Skip layer that explicitly introduces the periodic prior into the neural network model. We also add a linear bypass to the neural network, which makes the model robust to the out-of-domain time series data in real-world applications. In experiments, we consistently achieve better results in the temporal forecasting tasks compared with the methods introduced by previous publications.
4. **Efficient Transformer for sequence classification:** Since the huge success of the language model style pretraining models, such as Bert and Roberta, they have become the dominant solution for sequence classification tasks. The main shortcoming of the pretraining method is that the pretraining process consumes a huge amount of data and leads to expensive training costs. Here we identify an overlook redundancy of the Transformer pretraining process. In the downstream classification tasks, we only need one hidden representation to represent the whole sequence, while the Transformer always maintains a full sequence representation during pretraining and finetuning. According to this observation, we propose a novel Transformer architecture, Funnel-Transformer, which maintains a compressed sequence representation by injecting the pooling operation into Transformer architecture. The experiments show that the Funnel-Transformer can reduce the pretraining time by 30% without losing accuracy.
5. **Modeling temporal events with sparse labels:** Event temporal modeling is an important real-world application. However, event modeling is difficult because events are often not explicitly annotated in natural language tests. The event modeling systems usually rely on machine learning algorithms to detect the occurrences of events and the temporal relationship among them. It leads to insufficient human-annotated labels for both neural network training and testing. To alleviate the label sparsity issue, we design three improvements over the finetuning pretrained model method: (1) introduce the regularization by human prior knowledge about event temporal relationship (2) data augmentation (3) utilize the unlabeled data by semi-supervised technique. With the help of them, we boost the state-of-the-art result by a large margin.

Beyond the topics covered in this thesis, there are many interesting topics and unsolved problems, including:

Transfer the Works between the NLP Domain and Other Time Series Domains: Since the appearance of deep learning, NLP research is the pioneer of research related to sequence data. The attention mechanism and the Transformer model are all proposed for solving NLP problems. An important question is that: when we are solving tasks with other kinds of sequential data, what can we learn from the NLP models? According to our experience, an important commonality of NLP and other time series problems is modeling the **general** sequential dependency of data. For example, a shared core task of NLP domain and time series domains is prediction, which is

written as an autoregressive problem:

$$\underset{\theta}{\text{maximize}} \qquad P_\theta(x_t|x_{<t}) \tag{7.1}$$

In the NLP domain, this problem is named as language modeling. In the time series domains, it is named as forecasting. Because the NLP problems are the most extensively studied among all sequential tasks. It usually represents the advanced way to model sequential dependency. So when we are studying problems with time series data, we can trust the NLP model to capture the **general** sequential dependency, and focus on the other features that the language data doesn't have, such as periodic patterns and multivariate correlations. On the other hand, if we can provide a better way to model **general** temporal dependency in other domains, it should also be effective and be examined on the standard NLP tasks. For example, if we found a better way to model general temporal dependency and be helpful in the time series forecasting task, we should test that idea in the language modeling problem. The benefit to examine the general ideas in the NLP domain is that it can almost eliminate the influence of the hyper-parameters and problematic optimization to make sure that the correct conclusions are drawn. However, as demonstrated in this thesis, if the targeted domain exhibits a specific structure or human prior knowledge, we can propose a specific solution for this task.

Optimization Issue of Stochastic Neural Network: In Chapter 3, we reveal the problem of optimizing the sequential stochastic neural network. We find there is a large gap between its theoretical advantage and empirical performance. In [53], the authors propose the re-parameterization trick to enable the back-propagation over the deep neural networks with the stochastic latent variables. But in most of all benchmark datasets, such as the image and natural language model, the stochastic models did not achieve better results compared to the deterministic models. In our experiments, a human-designed posterior distribution can achieve similar or better performance compared to the optimized posterior distribution. It indicates the optimization of the stochastic latent variable is problematic. In summary, how to optimize the posterior distribution of the stochastic neural network and let it demonstrate the theoretical advantage is still an open problem.

Modeling the Variable Dependency of Temporal Data in Parallel: In Chapter 3, we design an autoregressive layer to capture the correlation of variables in multivariate temporal modeling tasks. Although this approach provides us a simple and powerful way to model correlations, it is very expensive in the aspect of computation time. Because the output of dimension i relies on the previous dimension, it prevents us to make predictions in a parallel manner. So how to model the correlation among variables with both efficient and powerful algorithms is still an open question.

Reducing the Sequential Redundancy in the Other Sequence Related Problems: In Chapter 5, we design an efficient Transformer for sequence classification tasks. Besides that, there are many other sequence related problems with sequential redundancy. For example, the Sequence to Sequence task (Seq2Seq) is another large category of applications with sequential data. We hypothesize that the sequential redundancy of the classification task also exists in the Seq2Seq tasks. For example, in the machine translation task, a classic sequence to sequence problem, the decoder may need two kinds of information. One is the global information, which conveys the semantic information from the source sentence. Another is more token-level information, such as the noun of a specific object. For the first kind of information, we can use a

similar idea as the funnel-transformer to compress the encoder representation. But for the second kind of information, we still need a token-level representation. How to make these two parts balanced is the key challenge in this direction.

Another related problem is speech. Compared with language data, the speech data contain more noise signals, which means there is more redundancy information. Furthermore, speech related applications usually require fast inference speed. How to utilize the Funnel-Transformer in the speech domain is a promising research direction.

Transformer Model for Time Series Data: During my PhD research career, the Transformer architecture took over the Recurrent Neural Network (RNN) to become the dominated solution for the tasks with sequential data. The major advantages of Transformer is that its global attention operation can capture longer sequential dependency compared with the recurrent neural network, and it does not suffer from the gradient vanish issue of RNN. Its disadvantage is that the computation and memory cost of attention operation grows quadratically with the length of sequence. In our thesis, some proposed improvements are implemented over the recurrent neural network backbone, including LSTNet (Section 4) and Factorized RNN (Section 3). In most cases, we can directly replace the RNN backbone with the Transformer backbone, and I expect them to achieve better results. In several cases such as the skip-recurrent link in LSTNet, it is specifically designed for recurrent neural networks. It can not directly migrate to Transformer. But we still can leverage the proposed idea, injecting the periodic pattern into the Transformer, to improve the Transformer model.

Bibliography

[1] Emre Aksan and Otmar Hilliges. Stcn: Stochastic temporal convolutional networks. *arXiv preprint arXiv:1902.06568*, 2019. 4, 23

[2] James Allan. *Topic detection and tracking: event-based information organization*, volume 12. Springer Science & Business Media, 2012. 71

[3] James Atwood and Don Towsley. Diffusion-convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 1993–2001, 2016. 15

[4] Mohammad Taha Bahadori, Qi Rose Yu, and Yan Liu. Fast multivariate spatio-temporal analysis via low rank tensor learning. In *Advances in neural information processing systems*, pages 3491–3499, 2014. 15

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 39

[6] Justin Bayer and Christian Osendorfer. Learning stochastic recurrent networks. *arXiv preprint arXiv:1411.7610*, 2014. 4, 23, 25

[7] Davide Boscaini, Jonathan Masci, Emanuele Rodolà, and Michael Bronstein. Learning shape correspondence with anisotropic convolutional neural networks. In *Advances in Neural Information Processing Systems*, pages 3189–3197, 2016. 7, 12

[8] Nicolas Boulanger-Lewandowski, Yoshua Bengio, and Pascal Vincent. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392*, 2012. 24

[9] George EP Box and David A Pierce. Distribution of residual autocorrelations in autoregressive-integrated moving average time series models. *Journal of the American statistical Association*, 65(332):1509–1526, 1970. 5, 36

[10] George EP Box, Gwilym M Jenkins, Gregory C Reinsel, and Greta M Ljung. *Time series analysis: forecasting and control*. John Wiley & Sons, 2015. 35

[11] Philip Bramsen, Pawan Deshpande, Yoong Keok Lee, and Regina Barzilay. Inducing temporal graphs. In *Proceedings of the 2006 Conference on Empirical Methods in Natural Language Processing*, pages 189–198, 2006. 72

[12] Michael Bronstein, Joan Bruna, Arthur Szlam, Xavier Bresson, and Yann LeCun. Geometric deep learning on graphs and manifolds. In *NIPS Tutorial*, 2017. 17

[13] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. Geometric deep learning: going beyond euclidean data. *IEEE Signal Processing Maga-*

zine, 34(4):18–42, 2017. 7, 8

[14] Joan Bruna, Wojciech Zaremba, Arthur Szlam, and Yann LeCun. Spectral networks and locally connected networks on graphs. *arXiv preprint arXiv:1312.6203*, 2013. 4, 12

[15] Nicolas Carion, F. Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander M Kirillov, and Sergey Zagoruyko. End-to-end object detection with transformers. *ArXiv*, abs/2005.12872, 2020. 66

[16] Nathanael Chambers, Taylor Cassidy, Bill McDowell, and Steven Bethard. Dense event ordering with a multi-pass architecture. *Transactions of the Association for Computational Linguistics*, 2:273–284, 2014. 71

[17] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *arXiv preprint arXiv:1606.01865*, 2016. 35

[18] Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. Adabert: Task-adaptive bert compression with differentiable neural architecture search. *arXiv preprint arXiv:2001.04246*, 2020. 51

[19] Xi Chen, Nikhil Mishra, Mostafa Rohaninejad, and Pieter Abbeel. Pixelsnail: An improved autoregressive generative model. *arXiv preprint arXiv:1712.09763*, 2017. 23

[20] François Chollet. Xception: Deep learning with depthwise separable convolutions. *arXiv preprint arXiv:1610.02357*, 2016. 4, 8, 10, 15

[21] Junyoung Chung, Caglar Gulcehre, KyungHyun Cho, and Yoshua Bengio. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555*, 2014. 38

[22] Junyoung Chung, Kyle Kastner, Laurent Dinh, Kratarth Goel, Aaron C Courville, and Yoshua Bengio. A recurrent latent variable model for sequential data. In *Advances in neural information processing systems*, pages 2980–2988, 2015. 4, 23, 25, 26

[23] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*, 2020. 51, 53, 61, 64

[24] Jerome Connor, Les E Atlas, and Douglas R Martin. Recurrent networks and narma modeling. In *NIPS*, pages 301–308, 1991. 36

[25] Ekin D Cubuk, Barret Zoph, Dandelion Mane, Vijay Vasudevan, and Quoc V Le. Autoaugment: Learning augmentation policies from data. *arXiv preprint arXiv:1805.09501*, 2018. 72

[26] Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019. 23, 57, 61, 65

[27] Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V Le. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *arXiv preprint arXiv:2006.03236*, 2020. iii, 1, 5

[28] Sakyasingha Dasgupta and Takayuki Osogami. Nonlinear dynamic boltzmann machines for time-series prediction. *AAAI-17. Extended research report available at goo. gl/Vd0wna*, 2016. 36

[29] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Advances in Neural Information Processing Systems*, pages 3844–3852, 2016. ix, 4, 12, 15, 16, 21

[30] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018. 3, 5, 6, 51, 52, 61, 65, 71

[31] Laurent Dinh, Jascha Sohl-Dickstein, and Samy Bengio. Density estimation using real nvp. *arXiv preprint arXiv:1605.08803*, 2016. 23

[32] Sergey Edunov, Myle Ott, Michael Auli, and David Grangier. Understanding back-translation at scale. *arXiv preprint arXiv:1808.09381*, 2018. 74

[33] Valentin Flunkert, David Salinas, and Jan Gasthaus. Deepar: Probabilistic forecasting with autoregressive recurrent networks. *arXiv preprint arXiv:1704.04110*, 2017. 23

[34] Marco Fraccaro, Søren Kaae Sønderby, Ulrich Paquet, and Ole Winther. Sequential neural models with stochastic layers. In *Advances in neural information processing systems*, pages 2199–2207, 2016. 4, 23, 25

[35] R Frigola-Alcade. *Bayesian Time Series Learning with Gaussian Processes*. PhD thesis, PhD thesis, University of Cambridge, 2015. 41

[36] Mathieu Germain, Karol Gregor, Iain Murray, and Hugo Larochelle. Made: Masked autoencoder for distribution estimation. In *International Conference on Machine Learning*, pages 881–889, 2015. 31

[37] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. *arXiv preprint arXiv:1704.01212*, 2017. 7, 11, 13, 15

[38] Anirudh Goyal Alias Parth Goyal, Alessandro Sordoni, Marc-Alexandre Côté, Nan Rosemary Ke, and Yoshua Bengio. Z-forcing: Training stochastic recurrent networks. In *Advances in neural information processing systems*, pages 6713–6723, 2017. 4, 23, 25

[39] Saurabh Goyal, Anamitra Roy Choudhary, Venkatesan Chakaravarthy, Saurabh ManishRaje, Yogish Sabharwal, and Ashish Verma. Power-bert: Accelerating bert inference for classification tasks. *arXiv preprint arXiv:2001.08950*, 2020. 65

[40] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, 2013. 23

[41] Karol Gregor, Ivo Danihelka, Alex Graves, Danilo Jimenez Rezende, and Daan Wierstra. Draw: A recurrent neural network for image generation. *arXiv preprint arXiv:1502.04623*, 2015. 23

[42] Karol Gregor, Frederic Besse, Danilo Jimenez Rezende, Ivo Danihelka, and Daan Wierstra. Towards conceptual compression. In *Advances In Neural Information Processing Systems*, pages 3549–3557, 2016. 23

[43] James Douglas Hamilton. *Time series analysis*, volume 2. Princeton university press Princeton, 1994. 35

[44] Nils Y Hammerla, Shane Halloran, and Thomas Ploetz. Deep, convolutional, and recurrent models for human activity recognition using wearables. *arXiv preprint arXiv:1604.08880*, 2016. 36

[45] Alan G Hawkes. Spectra of some self-exciting and mutually exciting point processes. *Biometrika*, 58(1):83–90, 1971. 6, 71

[46] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 38

[47] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 4, 8

[48] Jie Hu, Li Shen, and Gang Sun. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017. 8, 17, 19

[49] Gao Huang, Zhuang Liu, Kilian Q Weinberger, and Laurens van der Maaten. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016. 8, 13, 17, 19

[50] Ashu Jain and Avadhnam Madhav Kumar. Hybrid neural network models for hydrologic time series forecasting. *Applied Soft Computing*, 7(2):585–592, 2007. 36

[51] Thorsten Joachims. A probabilistic analysis of the rocchio algorithm with tfidf for text categorization. Technical report, Carnegie-mellon univ pittsburgh pa dept of computer science, 1996. 15

[52] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014. 20, 41, 44

[53] Diederik P Kingma and Max Welling. Auto-encoding variational bayes. *arXiv preprint arXiv:1312.6114*, 2013. 23, 81

[54] Durk P Kingma, Tim Salimans, Rafal Jozefowicz, Xi Chen, Ilya Sutskever, and Max Welling. Improved variational inference with inverse autoregressive flow. In *Advances in neural information processing systems*, pages 4743–4751, 2016. 23

[55] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016. 4, 7, 9, 15

[56] Lingpeng Kong, Cyprien de Masson d'Autume, Wang Ling, Lei Yu, Zihang Dai, and Dani Yogatama. A mutual information maximization perspective of language representation learning. *arXiv preprint arXiv:1910.08350*, 2019. 51

[57] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. 2009. 14

[58] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 3, 4

[59] Guokun Lai, Wei-Cheng Chang, Yiming Yang, and Hanxiao Liu. Modeling long-and short-term temporal patterns with deep neural networks. *arXiv preprint arXiv:1703.07015*, 2017. iii, 1, 4, 5, 21

[60] Guokun Lai, Hanxiao Liu, and Yiming Yang. Learning depthwise separable graph convolution from data manifold. *arXiv preprint arXiv:1710.11577*, 2017. iii, 1, 4

[61] Guokun Lai, Qizhe Xie, Hanxiao Liu, Yiming Yang, and Eduard Hovy. Race: Large-scale reading comprehension dataset from examinations. *arXiv preprint arXiv:1704.04683*, 2017. 61

[62] Guokun Lai, Bohan Li, Guoqing Zheng, and Yiming Yang. Stochastic wavenet: A generative latent variable model for sequential data. *arXiv preprint arXiv:1806.06116*, 2018. 4, 23

[63] Guokun Lai, Zihang Dai, Yiming Yang, and Shinjae Yoo. Re-examination of the role of latent variables in sequence modeling. In *Advances in Neural Information Processing Systems*, pages 7812–7822, 2019. iii, 1, 4

[64] Guillaume Lample and Alexis Conneau. Cross-lingual language model pretraining. *arXiv preprint arXiv:1901.07291*, 2019. 73

[65] Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*, 2019. 51, 64

[66] Colin Lea, René Vidal, Austin Reiter, and Gregory D Hager. Temporal convolutional networks: A unified approach to action segmentation. In *Computer Vision–ECCV 2016 Workshops*, pages 47–54. Springer, 2016. 36

[67] Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995. 7, 9

[68] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *Nature*, 521(7553): 436–444, 2015. 7

[69] Dong-Hyun Lee. Pseudo-label: The simple and efficient semi-supervised learning method for deep neural networks. In *Workshop on Challenges in Representation Learning, ICML*, volume 3, page 2, 2013. 72, 74

[70] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Ves Stoyanov, and Luke Zettlemoyer. Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *arXiv preprint arXiv:1910.13461*, 2019. 51

[71] Jiahan Li and Weiye Chen. Forecasting macroeconomic time series: Lasso-based approaches and their forecast combinations with dynamic factor models. *International Journal of Forecasting*, 30(4):996–1015, 2014. 35

[72] Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Graph convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017. 7, 8

[73] Rui Lin, Shujie Liu, Muyun Yang, Mu Li, Ming Zhou, and Sheng Li. Hierarchical recurrent neural network for document modeling. In *Proceedings of the 2015 Conference on*

Empirical Methods in Natural Language Processing, pages 899–907, 2015. 64

[74] Xiao Ling and Daniel Weld. Temporal information extraction. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 24, 2010. 71

[75] Zachary C Lipton, David C Kale, Charles Elkan, and Randall Wetzell. Learning to diagnose with lstm recurrent neural networks. *arXiv preprint arXiv:1511.03677*, 2015. 35

[76] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. Multi-task deep neural networks for natural language understanding. *arXiv preprint arXiv:1901.11504*, 2019. 51

[77] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019. 6, 51, 64, 76

[78] Juha Makkonen, Helena Ahonen-Myka, and Marko Salmenkivi. Topic detection and tracking with spatio-temporal evidence. In *European Conference on Information Retrieval*, pages 251–265. Springer, 2003. 71

[79] Jonathan Masci, Davide Boscaini, Michael Bronstein, and Pierre Vandergheynst. Geodesic convolutional neural networks on riemannian manifolds. In *Proceedings of the IEEE international conference on computer vision workshops*, pages 37–45, 2015. 7, 12

[80] Hongyuan Mei and Jason Eisner. The neural hawkes process: A neurally self-modulating multivariate point process. *arXiv preprint arXiv:1612.09328*, 2016. 6, 71

[81] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013. 15

[82] Federico Monti, Davide Boscaini, Jonathan Masci, Emanuele Rodolà, Jan Svoboda, and Michael M Bronstein. Geometric deep learning on graphs and manifolds using mixture model cnns. *arXiv preprint arXiv:1611.08402*, 2016. 7, 8, 12, 15

[83] Daniel Neil, Michael Pfeiffer, and Shih-Chii Liu. Phased lstm: Accelerating recurrent network training for long or event-based sequences. In *Advances in Neural Information Processing Systems*, pages 3882–3890, 2016. 36

[84] Qiang Ning, Hao Wu, and Dan Roth. A multi-axis annotation scheme for event temporal relations. In *ACL*, 7 2018. URL `http://cogcomp.org/papers/NingWuRo18.pdf`. 71, 75

[85] Qiang Ning, Zhili Feng, and Dan Roth. A structured learning approach to temporal relation extraction. *arXiv preprint arXiv:1906.04943*, 2019. 72

[86] Qiang Ning, Sanjay Subramanian, and Dan Roth. An improved neural baseline for temporal relation extraction. *arXiv preprint arXiv:1909.00429*, 2019. 71

[87] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016. 23

[88] Niki Parmar, Ashish Vaswani, Jakob Uszkoreit, Łukasz Kaiser, Noam Shazeer, and Alexander Ku. Image transformer. *arXiv preprint arXiv:1802.05751*, 2018. 23

[89] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton

Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018. 51

[90] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*, 2019. 51

[91] Danilo Jimenez Rezende and Shakir Mohamed. Variational inference with normalizing flows. *arXiv preprint arXiv:1505.05770*, 2015. 23

[92] Danilo Jimenez Rezende, Shakir Mohamed, and Daan Wierstra. Stochastic backpropagation and approximate inference in deep generative models. *arXiv preprint arXiv:1401.4082*, 2014. 23

[93] Stephen Roberts, M Osborne, M Ebden, Steven Reece, N Gibson, and S Aigrain. Gaussian processes for time-series modelling. *Phil. Trans. R. Soc. A*, 371(1984):20110550, 2013. 41

[94] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P Kingma. Pixelcnn++: Improving the pixelcnn with discretized logistic mixture likelihood and other modifications. *arXiv preprint arXiv:1701.05517*, 2017. 23

[95] Kristof Schütt, Pieter-Jan Kindermans, Huziel Enoc Sauceda Felix, Stefan Chmiela, Alexandre Tkatchenko, and Klaus-Robert Müller. Schnet: A continuous-filter convolutional neural network for modeling quantum interactions. In *Advances in Neural Information Processing Systems*, pages 992–1002, 2017. 13

[96] David I Shuman, Sunil K Narang, Pascal Frossard, Antonio Ortega, and Pierre Vandergheynst. The emerging field of signal processing on graphs: Extending high-dimensional data analysis to networks and other irregular domains. *IEEE Signal Processing Magazine*, 30(3):83–98, 2013. 7, 12

[97] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 14, 15

[98] David R So, Chen Liang, and Quoc V Le. The evolved transformer. *arXiv preprint arXiv:1901.11117*, 2019. 51

[99] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mass: Masked sequence to sequence pre-training for language generation. *arXiv preprint arXiv:1905.02450*, 2019. 51

[100] Kaitao Song, Xu Tan, Tao Qin, Jianfeng Lu, and Tie-Yan Liu. Mpnet: Masked and permuted pre-training for language understanding. *arXiv preprint arXiv:2004.09297*, 2020. 51, 64

[101] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. Highway networks. *arXiv preprint arXiv:1505.00387*, 2015. 39

[102] Gabriel Stanovsky, Julian Michael, Luke Zettlemoyer, and Ido Dagan. Supervised open information extraction. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 885–895, 2018. 74

[103] Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. Mobilebert: a compact task-agnostic bert for resource-limited devices. *arXiv preprint arXiv:2004.02984*, 2020. 51

[104] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2818–2826, 2016. 4, 8, 13, 17, 19

[105] Benigno Uria, Marc-Alexandre Côté, Karol Gregor, Iain Murray, and Hugo Larochelle. Neural autoregressive distribution estimation. *The Journal of Machine Learning Research*, 17(1):7184–7220, 2016. 23, 30

[106] Naushad UzZaman, Hector Llorens, Leon Derczynski, James Allen, Marc Verhagen, and James Pustejovsky. Semeval-2013 task 1: Tempeval-3: Evaluating time expressions, events, and temporal relations. In *Second Joint Conference on Lexical and Computational Semantics (* SEM), Volume 2: Proceedings of the Seventh International Workshop on Semantic Evaluation (SemEval 2013)*, pages 1–9, 2013. 71

[107] Aäron Van Den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR abs/1609.03499*, 2016. 3

[108] Aaron van den Oord, Nal Kalchbrenner, Lasse Espeholt, Oriol Vinyals, Alex Graves, et al. Conditional image generation with pixelcnn decoders. In *Advances in Neural Information Processing Systems*, pages 4790–4798, 2016. 23

[109] Vladimir Vapnik, Steven E Golowich, Alex Smola, et al. Support vector method for function approximation, regression estimation, and signal processing. *Advances in neural information processing systems*, pages 281–287, 1997. 40, 41

[110] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Advances in Neural Information Processing Systems*, pages 5998–6008, 2017. 5, 51, 52

[111] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017. 8, 13

[112] Marc Verhagen. Times between the lines. *Brandeis University, Massachusetts*, 2004. 72

[113] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R Bowman. Glue: A multi-task benchmark and analysis platform for natural language understanding. *arXiv preprint arXiv:1804.07461*, 2018. 61

[114] Jason Wei and Kai Zou. Eda: Easy data augmentation techniques for boosting performance on text classification tasks. *arXiv preprint arXiv:1901.11196*, 2019. 73

[115] Felix Wu, Angela Fan, Alexei Baevski, Yann N Dauphin, and Michael Auli. Pay less attention with lightweight and dynamic convolutions. *arXiv preprint arXiv:1901.10430*, 2019. 51

[116] Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. Lite transformer with long-short range attention. *arXiv preprint arXiv:2004.11886*, 2020. 51

[117] Qizhe Xie, Zihang Dai, Eduard Hovy, Minh-Thang Luong, and Quoc V Le. Unsupervised data augmentation. *arXiv preprint arXiv:1904.12848*, 2019. 72, 74

[118] Qizhe Xie, Minh-Thang Luong, Eduard Hovy, and Quoc V Le. Self-training with noisy student improves imagenet classification. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10687–10698, 2020. 74

[119] SHI Xingjian, Zhourong Chen, Hao Wang, Dit-Yan Yeung, Wai-Kin Wong, and Wang-chun Woo. Convolutional lstm network: A machine learning approach for precipitation nowcasting. In *Advances in neural information processing systems*, pages 802–810, 2015. 7, 8

[120] Jian Bo Yang, Minh Nhut Nguyen, Phyo Phyo San, Xiao Li Li, and Shonali Krishnaswamy. Deep convolutional neural networks on multichannel time series for human activity recognition. In *Proceedings of the 24th International Joint Conference on Artificial Intelligence (IJCAI), Buenos Aires, Argentina*, pages 25–31, 2015. 36

[121] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. In *Advances in neural information processing systems*, pages 5754–5764, 2019. 51, 60, 61, 64

[122] Adams Wei Yu, David Dohan, Minh-Thang Luong, Rui Zhao, Kai Chen, Mohammad Norouzi, and Quoc V Le. Qanet: Combining local convolution with global self-attention for reading comprehension. *arXiv preprint arXiv:1804.09541*, 2018. 73

[123] Hsiang-Fu Yu, Nikhil Rao, and Inderjit S Dhillon. Temporal regularized matrix factorization for high-dimensional time series prediction. In *Advances in Neural Information Processing Systems*, pages 847–855, 2016. 35, 41

[124] Rose Yu, Yaguang Li, Cyrus Shahabi, Ugur Demiryurek, and Yan Liu. Deep learning: A generic approach for extreme condition traffic forecasting. In *Proceedings of the 2017 SIAM International Conference on Data Mining*, pages 777–785. SIAM, 2017. 36

[125] G Peter Zhang. Time series forecasting using a hybrid arima and neural network model. *Neurocomputing*, 50:159–175, 2003. 35, 36, 41

[126] Guoqiang Zhang, B Eddy Patuwo, and Michael Y Hu. Forecasting with artificial neural networks:: The state of the art. *International journal of forecasting*, 14(1):35–62, 1998. 36

[127] Xiang Zhang, Junbo Zhao, and Yann LeCun. Character-level convolutional networks for text classification. In *Advances in neural information processing systems*, pages 649–657, 2015. 61

[128] Yongfeng Zhang, Min Zhang, Yi Zhang, Guokun Lai, Yiqun Liu, Honghui Zhang, and Shaoping Ma. Daily-aware personalized recommendation based on feature-level time series analysis. In *Proceedings of the 24th international conference on world wide web*, pages 1373–1383. International World Wide Web Conferences Steering Committee, 2015. 5

[129] Xiaojin Zhu, Zoubin Ghahramani, and John D Lafferty. Semi-supervised learning using

gaussian fields and harmonic functions. In *Proceedings of the 20th International conference on Machine learning (ICML-03)*, pages 912–919, 2003. 7, 9

[130] Yu Zhu, Hao Li, Yikang Liao, Beidou Wang, Ziyu Guan, Haifeng Liu, and Deng Cai. What to do next: modeling user behaviors by time-lstm. In *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, pages 3602–3608, 2017. 36