UiT

THE ARCTIC UNIVERSITY OF NORWAY

Faculty of Science and Technology

Department of Physics and Technology

Time Series Forecasting with Recurrent Neural Networks in Presence of Missing Data

—

Changkyu Choi

FYS-3900 Master's thesis in physics 60 SP - November 2018

Abstract

In many applications, time series forecasting plays an irreplaceable role in time-varying systems such as energy markets, financial markets, and so on. Predicting the dynamic of time-varying systems is essential but is a difficult task because it depends on not only the nature of the system but also on external influences, such as environmental conditions and social and economic status.

Recurrent Neural Networks (RNNs) are a special class of neural networks characterized by the recurrent internal connections, which enable to model the nonlinear dynamical system. Recently, they have been applied in the various forecasting tasks and reported that they outperform the forecast accuracy compared with conventional time series forecasting models. However, there is a limited study of time series forecasting using RNNs in the presence of missing data.

In this thesis, we propose a novel model that utilize Dilated RNN(DRNN) and a modified attention mechanism, focusing on the problem of time series forecasting with missing data. The proposed model outperforms existing models such as AutoRegressive Integrated Moving Average(ARIMA) and Gated Recurrent Unit(GRU), with respect to the forecast accuracy on benchmark datasets.

Besides, we provide a formal description of the learning procedure of RNNs, referred as truncated BPTT(k_2, k_1), and explain how to construct mini-batches of the training dataset for the forecasting tasks with RNNs, that has not been presented before this work. Discussions and future directions are suggested in five different perspectives at the end.

Acknowledgements

I would first like to show the gratitude to my supervisors, Filippo Maria Bianchi, Ph.D. and Professor Robert Jenssen, Ph.D. for the counsel and guidance throughout my time working on this thesis. Without your dedication, I wouldn't be able to complete the thesis. From my supervisors, I was able to learn not only the knowledge in the ML research domain but also the positive mindset and attitude required to be an academic researcher.

I would also like to appreciate my colleagues in UiT Machine Learning Group for their academic and social contributions. Associate professor Stian is the first Norwegian who pronounce my first name correctly, that I never forget. My neighbor Kristoffer, who is a fan of Newcastle and have started the master study at the same time and since then, has always been a good friend of mine, is the one who sincerely answers my humble questions throughout the studies. I hope Newcastle will rank higher in PL. The talk with Karl Øyvind is always fun because we are fans of Manchester United. We want United to sign a good central defender in this winter. Michael helps me in writing the project description and cover letter for my Ph.D. application. The conversation with Sigurd about Korean foods helps me to relieve the longing for my home country. Jonas, Thomas, and Jørgen, I am a huge fan of your jokes both on- and offline which made me feel that Machine Learning Group is now as comfortable as home. I also thank my former office mate Stine, current office mate Andre, and everyone whom I couldn't mention the name on it.

Above all, I appreciate Yoon with all of my heart for love and support.

Changkyu Choi

Contents

List of Figures

Abbreviations

ARIMA	**A**uto **R**egressive **I**ntegrated **M**oving **A**erage
ARMA	**A**uto **R**egressive **M**oving **A**erage
BPTT	**B**ack **P**ropoagate **T**hrough **T**ime
DRNN	**D**ilated **R**ecurrent **N**eural **N**etwork
ERNN	**E**lman **R**ecurrent **N**eural **N**etwork
FFNN	**F**eed **F**orward **N**eural **N**etwork
GEFCom	**G**lobal **E**nergy **F**orecast **Com**petition
GRU	**G**ated **R**ecurrent **U**nit
LSTM	**L**ong **S**hort **T**erm **M**emory
LTLF	**L**ong **T**erm **L**oad **F**orecasting
LVCF	**L**ast **V**alue **C**arried **F**orward
MSE	**M**ean **S**quare **E**rror
RNN	**R**ecurrent **N**eural **N**etwork
SARIMA	**S**easonal **A**uto **R**egressive **I**ntegrated **M**oving **A**erage
STLF	**S**hort **T**erm **L**oad **F**orecasting
SVM	**S**upport **V**ector **M**achine
SVR	**S**upport **V**ector **R**egression

Chapter 1

Introduction

1.1 Short-Term Load Forecasting of Electricity Demand in Time Series

As a type of energy resource, electricity has a unique characteristic which should be generated as soon as demanded because it cannot be stored [5]. In modern society, the irregularity of electricity demands becomes increasing due to a growth of population, an appearance of the new business, an increase of personal electric appliance, and so on. Accordingly, the importance of demand forecasting is emphasized in order to efficiently manage and distribute the resource which is temporally and quantitatively finite. Indeed, forecasting the future demand of resources within a distribution network of energy is fundamental for managing the limited availability of the assets [4]. The accurate forecast is a crucial factor in the planning of the electricity industry and the operation of electric power systems. It leads to substantial savings in operating and maintenance costs, increased reliability of power supply and delivery system, and correct decisions for future development [5].

However, load forecasting is a difficult task for some reasons. First, the load time series have multiple scales of time dependencies. For example, the load at a given hour is dependent on the load at not only the previous hour, but also the same hour of the last day, and the same hour of the day in the previous week [6]. Second, it also depends on the exogenous variables, such as environmental conditions including variations of climate, human activities and so forth [7].

These reasons motivate the research of forecasting models capable of improving this financial and social influence, by increasing the load forecasting accuracy even by a small percent [8–12]. Hence exploring reliable models of load forecasting for the electricity demand has been an active topic of research for decades. The major methods to perform STLF include exponential smoothing [13], ARIMA [14], neural networks [15], and support vector machine (SVM) [16].

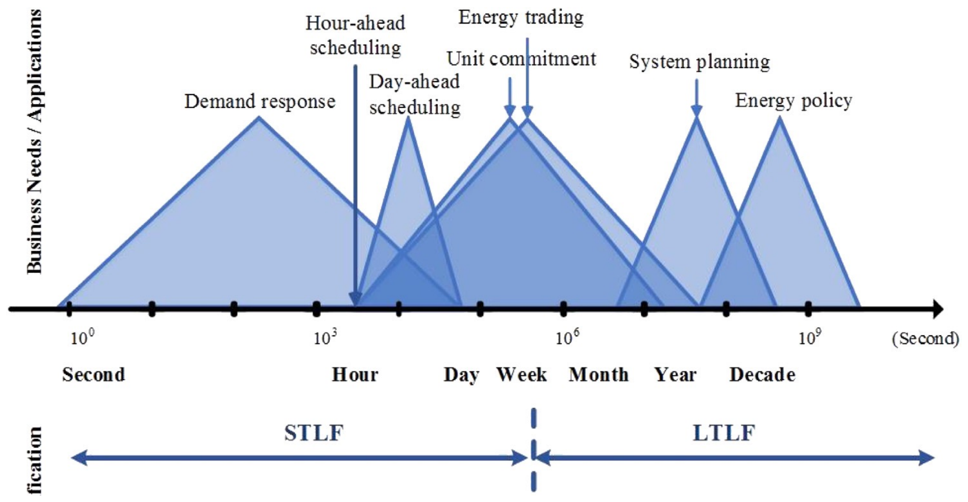


FIGURE 1.1: Classification of the forecasting application according to the forecast time in electric power generation. A coarse classification may lead to two categories, short-term load forecasting (STLF) and long term load forecasting (LTLF), with a cut-off horizon of two weeks. Day-ahead scheduling, which forecasts from an hour to a day, has been an important research topic for STLF. Source : Probabilistic electric load forecasting: A tutorial review, Hong and Fan [1]

Figure 1.1 depicts the load forecasting applications and classification. There is a tacit standard for classifying the range of load forecasts based on the forecasting horizons. A rough classification may lead to two categories, short-term load forecasting (STLF) and long term load forecasting (LTLF), with a cut-off horizon of two weeks [17].

This thesis focuses on short-term load forecasting (STLF), specifically day-ahead scheduling in Figure 1.1, because it has been a challenging issue for electric power companies to forecast day-ahead demand due to the unpredictable factors in the representation of the demand. An improved STLF accuracy contributes the power company to provide safe and stable electricity to end users [5]. Hong and Fan [1] report that the literature on STLF is much more extensive than that on LTLF.

1.2 STLF with Recurrent Neural Networks

Over the past several decades, various forecasting models have been proposed to improve STLF accuracy. Classical models, such as the auto-regressive model including ARIMA [18] and exponential smoothing [19] both based on statistical background, have been popularly applied. However, the accuracy of the forecast is limited to a given degree because it is difficult to precisely model nonlinear and random-like demand patterns by making strong statistical assumptions that are realistic or suitable for different scenarios.

After the advent of recurrent neural networks (RNNs), an essential family of neural networks, RNNs become a standard framework for STLF because of their dominant performance and high expressivity. The networks can learn functions of arbitrary complexity and deal with time series data possessing properties such as nonlinear interactions between latent variables without making too strong statistical assumptions [4].

RNNs are characterized by recurrent internal connections, which enable to capture time dependencies and to model a dynamical (that generates the observed variables) system up to a given degree of accuracy [20]. As an RNN processes sequential information, it performs the same operations on every element of the input sequence. Its output, at each time step, depends on previous inputs and past computations. Accordingly, the networks integrate past and current information and can predict future values [21, 22]. This allows the network to develop a memory of previous events, which is implicitly encoded in its hidden state variables. Different types of RNNs, such as long short term memory (LSTM) and gated recurrent unit (GRU) have been applied in various STLF tasks, such as electricity load forecasting [23], traffic speed prediction [24], traffic peak forecasting [25] and improve the accuracy compared with classical time series forecasting models.

1.3 Challenges : STLF with RNNs towards Missing Data

Missing data are a problem because algorithms for STLF cannot explicitly deal with them [26]. Therefore, one must fill the missing data beforehand, using imputation which is a source of bias. In order to replace missing values in time series, manual imputation, which is filling up the missing values by plausible values instead of removing out, is employed [27]. Mean substitution and last value carried forward [28] are typical techniques of manual imputation.

However, manual imputation can be still a source of bias for RNNs if values are missed for successively long time steps. In practice, missing values very often tend to be observed consecutively over a particular period of time. In GEFCom 2012 dataset [2], for example, there are several time windows of one week where data are missing. As RNNs transfer information through the recurrent connections with time delay 1, the bias provoked by one of the imputed values in the window is accumulated and transferred to the next time step until the window is over. The transferred bias can result in the gradual deterioration of the forecast reliability along the time steps within the window. Figure 1.2 depicts the gradual deterioration of the forecast for three successive missing values $\{x_{t-1}, x_t, x_{t+1}\}$.

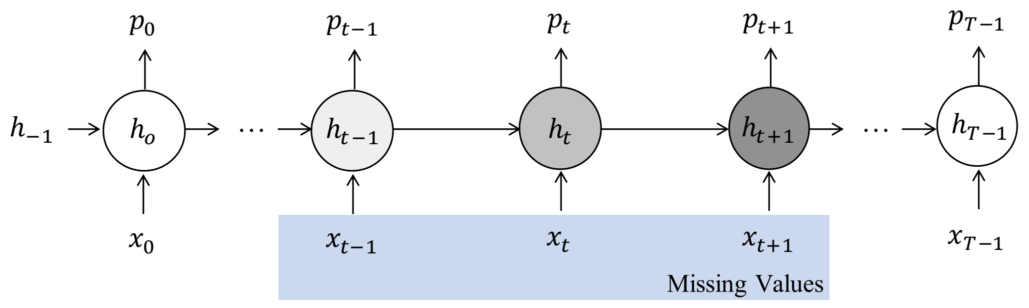


FIGURE 1.2: Unfolded graph of a single layer RNN for the STLF task of length-T time series. Three successive values $\{x_{t-1}, x_t, x_{t+1}\}$ in the blue window are missed thus manually imputed. As the RNN processes the input time series in a sequential manner, bias caused by manual imputation are accumulated and transferred through the internal self-connection. The color of the RNN units represents the degree of the bias.

1.4 Proposed Approach : Dilated RNNs with a Modified Attention Mechanism

In this thesis, we propose a RNN framework to deal with missing data. Specifically, it aims to lower the effect of bias caused by the manual imputation over the missing window. The framework consists of dilated RNNs [29] with a modified attention mechanism [30].

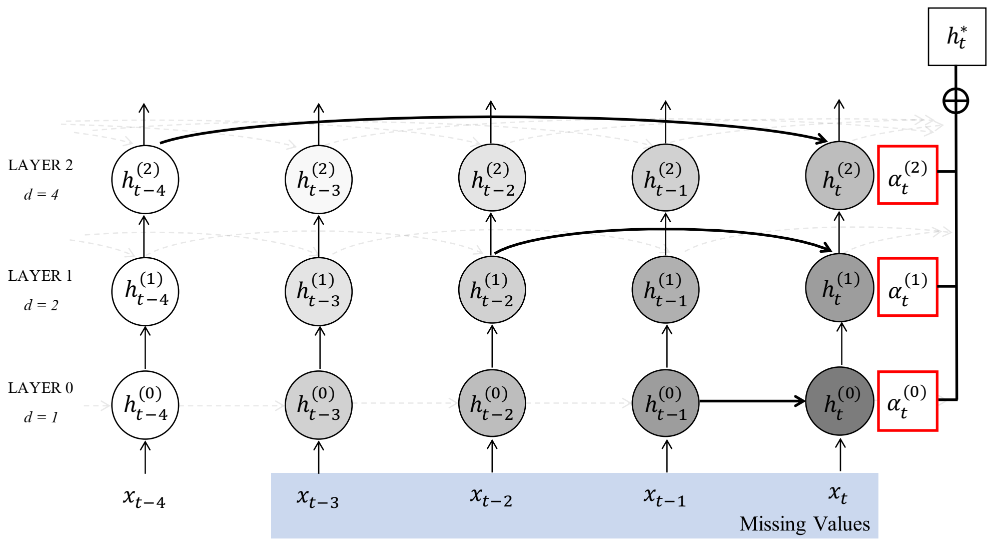


FIGURE 1.3: Unfolded graph of the proposed framework : dilated RNN(3) with modified attention mechanism. Four successive values $\{x_{t-3}, x_{t-2}, x_{t-1}, x_t\}$ in the blue window are missed thus manually imputed. In layer 0, bias caused by manual imputation are accumulated and transferred from left to right through its recurrent connection with dilation 1. But the bias from the imputed values is counterbalanced when it is processed by the upper layer, holding longer dilation in the RNN structure, since the RNN with longer dilation updates its state less often within the time window with missing values. A weighted average of the states h_t^* represents the state at the time step t over the layers. Attention weights $\alpha_t^{(l)}$ enable the RNN to learn the layer which provides the most reliable state each time, in the presence of missing values. The color of the RNN units represents the degree of the bias from the imputed values.

Dilated RNNs are a stack of multiple single layer RNN with different length of dilated recurrent skip connections, referred to 'dilations' [29]. Recurrent units in each layer of dilated RNNs have a dilation with a different length which provides more flexibility and capability of modeling different time scales to capture longer dependencies in time. In the networks, bias from the imputed values is counterbalanced by the upper layers, holding longer dilation in the RNN structure. The RNN with longer dilation updates its state $h^{(l)t}$ less often than one with shorter dilation. Therefore, the RNN with longer dilation is less biased in the presence of long windows where data are missing.

Attention mechanism is introduced by Bahdanau et al. [30] in the field of neural machine translation. Neural machine translation is based on the framework of RNN Encoder Decoder, proposed by Cho et al. [31]. The attention mechanism is introduced to focus on specific parts of the input sequence while computing output sequence using a weighted average. In this thesis, we modify the conventional attention mechanism to utilize it in dilated RNN structure. Modified attention mechanism focuses on a specific layer of the dilated RNNs at each time step using a weighted average of the states from different layers to compute the output. A weighted average of the states h_t^* represents the state at the time step t over the layers. Attention weights $\alpha_t^{(l)}$ enable the RNN to learn the layer which provides the most reliable state each time, especially in the presence of missing values, and decides to which specic layer to allocate importance. Weights usually take values in the interval $\left[0, 1\right]$, and they sum to 1. Figure 1.3 depicts the proposed framework of the thesis.

This thesis focuses on the comparison of forecasting accuracy between the novel framework and existing methods on two independent time series, including real world dataset.

1.5 Contributions

The major contribution of the thesis is that we develop a novel framework based on dilated RNNs and attention mechanism for the task of STLF from missing values in time series. As a second contribution, we provide a formal description of truncated BPTT(k_2, k_1), and explain how to construct the training dataset for the forecasting tasks with RNNs. As RNNs have been mostly used for classification, there is a lack of knowledge of how to train them for the forecasting tasks. To the best of our knowledge, it has not been described formally before this work.

1.6 Notations

Unless otherwise stated, the following notation will be used throughout this thesis:

- Scalars will be written in lowercase, for example, x
- Vectors will be written in lowercase bold, for example, $\mathbf{x}$
- Matrices will be written in uppercase bold, for example, $\mathbf{X}$
- time index t will be written as a subscript of any character, for example, $\mathbf{x}_t$
- layer index l will be written in a parenthesis of superscript of any character, for example, $\mathbf{x}_t^{(l)}$

1.7 Structure of Thesis

This thesis consists of ten chapters, including this introductory chapter.

Chapter 2 presents an introduction to the problem of STLF and the properties of electricity demand time series. The chapter continues by reviewing several methods previously applied for STLF based on statistical and machine learning approaches.

Chapter 3 introduces RNNs as a particular class of neural networks specialized in the processing of sequential data, with a detailed explanation of training procedure including forward and backward propagation through time. Expected issues while training and the solutions are discussed.

Chapter 4 provides, as the important contribution of the thesis, the formal description of mini-batch training based on truncated BPTT(k_2, k_1) learning process and explains how to train the RNNs for the forecasting tasks using the mini-batch training.

Chapter 5 provides advanced cell architecture of RNNs, such as long short-term memory (LSTM) and gated recurrent unit (GRU), followed by the operational principle of the gated cell structure.

Chapter 6 begins by the description of deep RNN structure, and then introduces the concept of recurrent skip connection. The structure of dilated RNNs are introduced by comparing conventional stacked RNNs.

Chapter 7 introduce the procedure of missing data analysis with RNNs. Begun by the type of missing data, the chapter provides approaches on how to deal with the missing values in time series and how to incorporate the missing patterns into RNN analysis.

In Chapter 8, we provide, as a main contribution of the thesis, a detailed introduction to the novel framework we propose in this thesis. To explain the novel framework, introduction to conventional attention mechanism is preceded. Details of the experiments, such as datasets, setting, and results and following discussions are given in Chapter 9.

Chapter 10 gives conclusions and future directions of the thesis.

Chapter 2

Introduction to the Problem of Short-Term Load Forecasting

In many applications, short-term load forecasting plays an irreplaceable role in time-varying systems such as energy markets, financial markets, business management, planning [32] and basic operation systems including fuel scheduling, and unit maintenance [33]. Predicting the dynamics of time-varying systems is important but is a difficult task because it depends on not only the nature of the system but also on external influences, such as environmental conditions including variations of climate, social and economic status [7]. Therefore, exploring reliable models of short-term load forecasting (STLF) for the time-varying systems has been an active topic of research.

During the past years, a wide variety of forecasting models has been suggested for STLF to improve the forecasting accuracy. Two important classes of methods for STLF are statistical approaches and machine learning approaches, though the boundary between the two is becoming more and more ambiguous, as a result of multidisciplinary influences in the scientific community [6]. Recently, recurrent neural networks (RNNs), an important family of neural networks within the extent of machine learning models, have emerged and applied in the STLF task, such as electricity load forecasting [23], traffic speed prediction [24], traffic peak forecasting [25] and so forth. These studies commonly report that RNNs improve the accuracy of STLF compared with classical time series forecasting models.

In this chapter, we discuss the various models that have been applied to STLF tasks, focusing on the RNN models applied to electricity load forecasting.

2.1 Short-Term Load Forecasting for Electricity Load

Electricity as a product has very unique characteristics compared to a material product because electricity energy cannot be stored as it should be provided as soon as it is demanded [5]. This property places importance on load forecasting, specifically short-term load forecasting. STLF can reduce risk of over- and under-contracts on balancing markets due to load prediction errors [34]. Moreover, it keeps energy markets efficient and provides a better understanding of the dynamics of the monitored system [35]. On the other hand, an inaccurate STLF could give rise to either a load overestimation, which brings to the excess of supply and consequently more costs for market participants, or a load underestimation, which results in failures of providing enough resources needed. Both draw serious inconvenience to energy based service end users [36, 37]. These reasons motivate the research of forecasting models capable of reducing this financial and social costs, by increasing the load forecasting accuracy even by a small percent [8–12].

Electricity load time series is characterized by several properties, namely, multiple time dependencies, weather effects, and calendar effects. These dependencies are often complex and highly nonlinear so that they make the accurate forecast difficult [6].

2.1.1 Multiple Time Dependencies

The load time series has multiple scales of time dependencies. For example, the load at a given hour is dependent on the load at not only previous hour but also the same hour of the previous day and the same hour of the day in the previous week. Figure 2.1, sourced by Dang-Ha et al. [3], shows hourly measured electricity load data at zone 1 of the GEFCom 2012 dataset [2]. In the figure, three strong time dependencies are observed, namely, within a day (intraday), within a week (intraweek) and across different seasons. Time dependencies of intraday and intraweek originate from the routines of human. For intraday cycles, the load peaks at breakfast time and before dinner. For intraweek cycles, the load on the weekend is usually lower than on the weekdays. Seasonal time dependency is closely related to the temperature. In Figure 2.1, the load is higher in

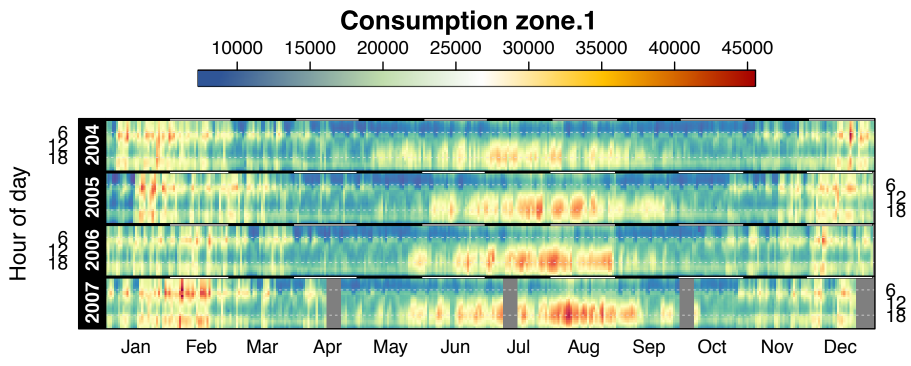


FIGURE 2.1: Hourly load profiles in KWh of zone 1 for 4 years from GEFCom 2012 dataset [2]. Three strong time dependencies are observed, within a day, within a week and across different seasons. Source : Local short-term electricity load forecasting: Automatic approaches, Dang-Ha et al. [3]

summer and winter compared with other seasons. It implies that electricity consumption is increased for heating in the winter time while for cooling in the summer time.

2.1.2 Weather Effects

As briefly mentioned, weather conditions have always been an important variable to be considered in electricity load forecasting [3]. Temperature is a factor that strongly influences the load among many meteorological factors like humidity, wind, rainfall, cloud cover, thunderstorm and so forth. Hong and Shahidehpour [17] report that the temperature factor can explain more than 70% of the load variance in the GEFCom 2012 dataset [2]. Hence, time series of the temperature can be considered as exogenous input of the STLF models.

2.1.3 Calendar Effects

As the load of electricity consumption is closely related to human behavior, special calendar events (holidays, festival days and so on) can demand uncommon load of electricity. Those situations represent outliers and could be treated differently to improve the model accuracy [3]. Fan and Hyndman [38] include public holidays in the statistical STLF model with annual, weekly, and daily seasonal patterns to forecast electricity load in Australia. The calendar effects are considered critical in other domains, for example, transport industry like Uber which represents car sharing business. Accurate forecasting of trips during special events can increase efficiency of driver allocation, resulting in a decrease of waiting time for the riders. To forecast the traffic load of Uber in special events, Laptev et al. [25] utilizes RNNs with a new architecture leveraging an autoencoder for feature extraction. The autoencoder is out of scope in this thesis.

2.1.4 Other Effects

There are still obvious factors that affects a load of electricity consumption, for example, geographical locations, human comfortable temperature, heating/cooling technology, type of consumers or purpose of electricity use (industrial or residential) and so on [3]. These various factors make electricity load patterns become more complex and irregular, that impedes the accuracy of the forecast.

As Almeshaiei and Soltan [5] argue that an ideal forecasting model for a case may perform poorly for another one, it is very important for researchers to understand that a universally best technique simply does not exist [1]. Note that the forecasting accuracy may also differ significantly for different utilities, different zones within a utility, and different time periods. Therefore, researchers should focus on discovering efficient and effective modifications that suit the specific case, based on general techniques.

In the following sections, techniques mostly applied for STLF tasks, are reviewed in terms of two categories : (a) statistical approaches, such as autoregressive models, and exponential smoothing models (b) machine learning approaches, such as Support Vector Machine (SVM), Feedforward Neural Networks (FFNNs) and Recurrent Neural Networks (RNNs).

2.2 Statistical approaches

In this section, we will review statistical approaches used in STLF tasks, mainly, autoregressive models, and exponential smoothing models. Both autoregressive and exponential smoothing models represented for many years the baseline among systems for time series prediction [39].

Such models require to properly select the lagged inputs to identify the correct model orders, a procedure which demands a certain amount of skill and expertise [40]. Moreover, autoregressive models make explicit assumptions about the nature of system under analysis.

Therefore, their use is limited to those settings in which such assumptions hold and where a-priori knowledge on the system is available [41]. Taylor [18] showed that for long forecasting horizons a very basic averaging model, like Autoregressive Integrated Moving Average (ARIMA) or exponential smoothing, can outperform more sophisticated alternatives. However, in many complicated systems the properties of linearity and even stationarity of the analyzed time series are not guaranteed. Nonetheless, given their simplicity, autoregressive models have been largely employed as practical implementations of forecast systems.

2.2.1 Autoregressive Models

Autoregressive (AR) model is one of frameworks representing a random process X_t varying in terms of time t. Thus, it is broadly used to explain the time-varying systems, for example trend in financial markets, and so on. The autoregressive model specifies that the output variable depends linearly on its own previous values and on a stochastic term. Thus the model is expressed in the form of a stochastic difference equation. Equation 2.1 denotes p^{th} order of autoregressive model referring $AR(p)$, where c, ϕ_i and $\epsilon_t \sim N(0, \sigma^2)$ are parameters of the model and white noise, respectively.

$$X_t = c + \sum_{i=1}^{p} \phi_i X_{t-i} + \epsilon_t \tag{2.1}$$

This model is based on the assumption of (weak) stationarity. This means that the stationary time series is assumed to have constant mean and variance, and autocovariance only dependent on the time lag τ. Equation 2.2 shows the formal expression of stationarity for any integer t, s and τ, Note that the order of the model p is fixed a-priori while the parameters are adapted on the data at hand [42, 43].

$$\begin{aligned} E[X_t] &= \mu \\ Var[X_t] &= \sigma_y^2 \\ Cov[X_{t+\tau}, X_t] &= Cov[X_{s+\tau}, X_s] \end{aligned} \tag{2.2}$$

Among the different types of AR models, Autoregressive Moving Average (ARMA) model is often used in the STLF task. ARMA models provide a parsimonious description of a stochastic process in terms of two polynomials, one an autoregression and the other a moving average [40, 44–46].

The q^{th} order of moving average process, $MA(q)$ is defined in Equation 2.3, where μ, θ_i and $\epsilon_t, \epsilon_{t-1} \cdots \sim N(0, \sigma^2)$ are expectation of X_t, parameters of the model and white noise terms.

$$X_t = \mu + \epsilon_t + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \tag{2.3}$$

Thus $ARMA(p, q)$, the sum of $AR(p)$ and $MA(q)$ is denoted in Equation 2.4, where μ is often assumed 0.

$$X_t = c + \epsilon_t + \sum_{i=1}^{p} \phi_i X_{t-i} + \sum_{i=1}^{q} \theta_i \epsilon_{t-i} \tag{2.4}$$

In practice, load data are often nonstationary. To comply with stationarity assumption for autoregressive frameworks, Autoregressive Integrated Moving Average (ARIMA) model is suggested. ARIMA model is a generalization of Autoregressive Moving Average (ARMA) model.

The ARIMA model was adopted in STLF back in 1987 [47] and still remains a popular baseline. The extension of ARIMA models is also used in STLF tasks, such as ARIMAX [48, 49], generalization of ARIMA models including exogenous variables and SARIMA [47, 50] which includes seasonality of time series.

ARIMA applies differentiation of order d on the random process X_t. Equation 2.5 denotes an $ARIMA(p, d, q)$ model on the random process X_t, where B denotes a back-shift operator, $X_{t-n} = B^n X_t$.

$$(1 - \sum_{i=1}^{p} \phi_i B^i)(1 - B)^d X_t = c + (1 + \sum_{i=1}^{q} \theta_i B^i)\epsilon_t \tag{2.5}$$

One big challenge of ARIMA is that model selection, determined by hyperparameters (p, d, q), is hard to automate and still requires human expertise [51, 52]. It gets harder if the data includes seasonality because seasonal hyperparameters should be additionally considered for seasonal ARIMA (SARIMA) (p, d, q, P, D, Q).

Model hyperparameters are usually derived from the Box-Jenkins test, which examines the autocorrelation (ACF) and partial autocorrelation function (PACF) of the time series to select the candidates of the hyperparameter set. The value of Akaike information criterion (AIC) [53] is compared among the candidates. The most parsimonious model with the lowest AIC will be the model hyperparameters. As (seasonal) ARIMA is based on the linearity and parsimonious rule, it usually achieves lower performance if the data is complex.

2.2.2 Exponential Smoothing Models

Exponential smoothing assigns exponentially decreasing weights to the observation as they get older. In other words, recent observations are given relatively more importance in forecasting than the older observations [54]. Depending on the characteristics of the dataset, the number of smoothing parameters are determined by an initial analysis.

For an electricity load time series which has trend and single seasonality, exponential smoothing models can be applied with three smoothing parameters where each parameter corresponds to (deseasonal) level s_t, trend b_t and seasonal component c_t, respectively. In the case of additive seasonality, these models consider the h time step ahead forecasting value f_{t+h} to be an aggregated value of the three components, shown in Equation 2.6, where L is a seasonal length.

$$f_{t+h} = s_t + h \cdot b_t + c_{t+h-L} \tag{2.6}$$

For each components, smoothing parameters are defined, α for deseasonal level, β for trend and γ for seasonal component, $0 < \alpha, \beta, \gamma < 1$. These components have recursive relationship to generate each components for the next time step [19]. Equation 2.7 denotes the formal expression of the recursive relationship of each components for an observation x_t, referring each overall smoothing, trend smoothing and seasonal smoothing. The optimal smoothing parameters α, β and γ are estimated in such a way that the mean square error between actual value and estimated value is minimized.

$$\begin{aligned} s_t &= \alpha(x_t - c_{t-L}) + (1-\alpha)(s_{t-1} + b_{t-1}) && : \text{Overall smoothing} \\ b_t &= \beta(s_t - s_{t-1}) + (1-\beta)b_{t-1} && : \text{Trend smoothing} \\ c_t &= \gamma(x_t - s_t) + (1-\gamma)c_{t-L} && : \text{Seasonal smoothing} \end{aligned} \tag{2.7}$$

As seen in Equation 2.7, initial values for each components should be assigned. In fact, determining initial values can become a source of bias in forecasting because the initial values of each component will have an unreasonably large effect on early forecasts if the models put substantial weights on past observations. Kalekar [19] provides the detail of various techniques regarding initial value selection in order to reduce the effect of the bias.

2.3 Machine Learning Approaches

Machine learning approaches try to discover consistent patterns from data, instead of modeling the underlying physical processes heuristically. A mapping between the input and the ground truth is estimated by the function approximation framework and then used for the forecast [3]. The basic formulation is represented in Equation 2.8, where x_t and y_t are input and the forecasted value at a time step t and the function $F(\cdot)$ is a non-linear function, which could be estimated by machine learning approaches, such as Support Vector Machines (SVMs), Feedforward neural networks (FFNNs) and RNNs.

$$y_t = F(y_{t-1}, y_{t-2} \cdots, x_t, x_{t-1}, x_{t-2} \cdots) \tag{2.8}$$

2.3.1 Support Vector Machines

Support Vector Machines (SVMs) are learning models that analyze data and recognize patterns, often being used for classification analysis. SVM has been shown to be very resistant to the problem of over-fitting, and could achieve good performances for solving time series forecasting problems [1]. Details of SVMs are represented in the article of Sapankevych and Sankar [55] and the appendix Wickstrøm [56].

Support Vector Regression (SVR), an splitting of SVM in the continuum have been applied in time series forecasting [55] by sequential input values in time windows of fixed length. The approach can only succeed if there are no critical temporal dependencies exceeding the windows length, making the SVR unable to learn an internal state representation for sequence learning tasks involving time lags of arbitrary length.

2.3.2 Feedforward Neural Networks

Neural networks are a prominent example of data-driven models to learn arbitrary functions. They are widely used in many practical application, such as pattern classification, function approximation, optimization and forecast [57, 58].

Feedforward neural networks (FFNNs) [59] play a role like universal function approximators. Many studies employ FFNNs in STLF tasks to forecast one or a fixed number of future values [60–66].

FFNNs are consisted of three types of layers, input, hidden and output layer. Each layer have multiple number of neurons. Neurons are highly interconnected from bottom (input layer) to top (output layer). Each neuron performs a simple computation, defined in Equation 2.9, where j and k denote index of neuron and its input index, $\{o_j, i_k\}$ are output and input of the neuron, $\{w_{jk}, b_j\}$ are weight / bias parameters of the neuron and f_j is an activation function.

$$o_j = f_j(\sum_k w_{jk} i_k + b_j) \tag{2.9}$$

The architecture of FFNNs can be modified depending on the purpose of researchers, for example, inserting multiple hidden layers. Convolutional neural networks (CNNs) are a

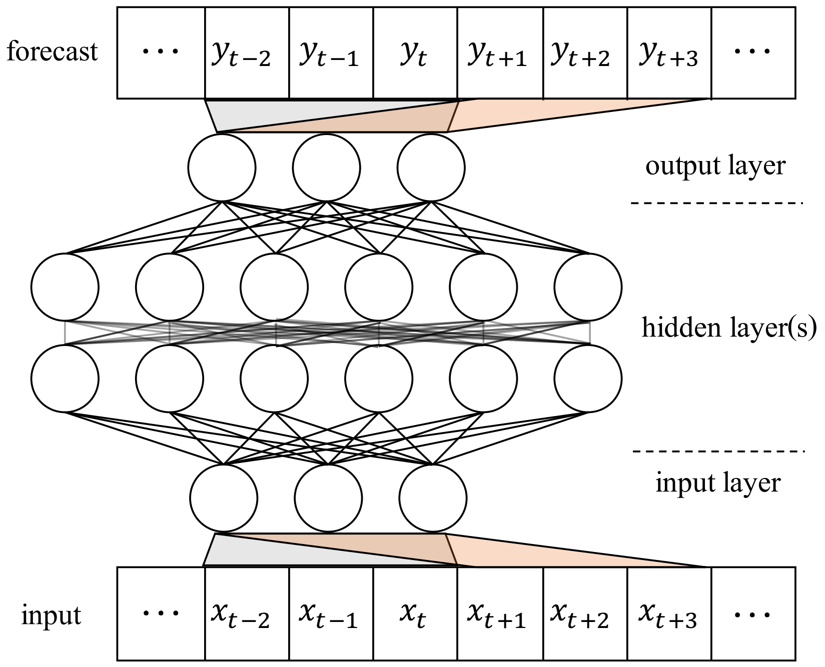

forecast
$\cdots$ y_{t-2} y_{t-1} y_t y_{t+1} y_{t+2} y_{t+3} $\cdots$
output layer
hidden layer(s)
input layer
input
$\cdots$ x_{t-2} x_{t-1} x_t x_{t+1} x_{t+2} x_{t+3} $\cdots$

FIGURE 2.2: Feedforward neural network with 2 hidden layers. The networks process as inputs time windows of length 3 and provide as outputs time windows of length 3 sequentially.

type of FFNNs which are composed of alternating convolutional and subsampling layers [67]. But to be fed into the networks as inputs, a long time series should be divided into small windows of fixed size. Figure 2.2 shows a FFNN architecture with 2 hidden layers, which process as inputs time windows of length 3. The operation is repeated to forecast next values by shifting the time window across the whole time series [68].

While FFNNs have been proved to be effective in many circumstances [69–72], they do not consider temporal ordering as an explicit feature of the time series. Also, it is not suitable in cases where the time dependency in the time series is greater than the length of time windows. By the same reason, CNNs is not often applied for time series analysis.

On this account, a Recurrent Neural Networks (RNNs) are a more flexible model since it encodes the temporal context in its feedback connections, which are capable of capturing the time-varying dynamics of the underlying system [20, 73].

2.3.3 Recurrent Neural Networks

RNNs are a special class of neural networks characterized by internal self-connections, which enable to model, in principle, any nonlinear dynamical system, up to a given degree of accuracy [20]. RNNs and their variants have been used in many contexts where the temporal dependency in the data is an important feature. Applications of RNNs include sequence transformation [74], language modeling [75–78], speech recognition [79], learning word embeddings [80], audio modeling [81], handwriting recognition [82, 83], image generation [84], and so on.

As an RNN processes sequential information, it performs the same operations on every element of the input sequence. Its output, at each time step, depends on previous inputs and past computations. This allows the network to develop a memory of previous events, which is implicitly encoded in its hidden state variables. This is certainly different from traditional FFNNs, where it is assumed that all inputs are independent of each other.

Theoretically, RNNs can remember arbitrarily long sequences. However, their memory is in practice limited by their finite size and, specifically, by vanishing/exploding gradient problem while training of their parameters. To overcome memory limitations, advanced cell structures are suggested, referring Long-Short Term Memory (LSTM) [85] and Gated Recurrent Unit (GRU) [86], both utilize gated structures in contrast to ordinary RNN cells (Elman RNN, ERNN). The advanced cell structure enables the RNNs to increase its capability of storing information for longer periods of time.

Contrarily to other linear models adopted for STLF tasks, RNNs can learn functions of arbitrary complexity and they can deal with time series data possessing properties such as saturation or exponential effects and nonlinear interactions between latent variables [4]. However, if the temporal dependencies of data are mostly contained in a finite and small time interval, the use of RNNs can be unnecessary. In these cases, performances, in terms of computational resources and accuracy, are generally lower than the ones of time-window approaches, like ARIMA, exponential smoothing, SVM, and FFNNs.

On the other hand, in many STLF tasks, the time series to be predicted are characterized by long term dependencies, whose extent may vary in time or be unknown in advance. In all these situations, the use of RNNs turn out to be the best solution.

Recent studies have suggested the design of novel RNN architectures, where stacking multiple layers of single layer (shallow) RNN with different length of time skip connection (or dilation) [29, 87]. The novel architectures help RNNs to learn long term dependencies efficiently, in terms of computational resources and accuracy. Details of RNN architectures are discussed in the next several chapters.

Chapter 3

Recurrent Neural Network Properties and Training

3.1 Properties

In general, RNNs are a learning model that updates new state h_t using previous state h_{t-1} and current input x_t recursively. It can also be described by a network, which is composed of multiple cells connecting in series along the time axis. Each cell in the network computes state at a certain time step. Figure 3.1 illustrates forward propagation of RNN in two ways. Cyclic shape in the left is called 'folded graph' while acyclic shape in the right is called 'unfolded graph' which expands cyclic shape in time. Note that parameters in the unfolded representation share their values over the cells and the values are updated simultaneously within the optimization procedure.

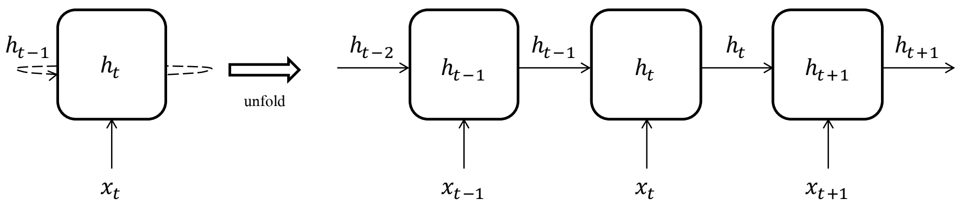


FIGURE 3.1: Folded graph of RNN (left) and the unfolded in time (right) during forward propagation. The new state h_t sequentially is updated by the current input x_t and the previous state h_{t-1}.

Depending on the approach to update the new state h_t, architectures of RNN differ such as Elman RNN (ERNN), Long short-term memory (LSTM)[85] and Gated recurrent unit

(GRU)[88]. Further analysis in this chapter will focus on ERNN since it has the most simple and basic architecture among them.

3.2 Training ERNN

In general, training means a process that a model learns the optimal parameters with a training set by minimizing defined error function which depends on trainable parameters. Training is composed of 3 steps, forward propagation, backpropagation and parameter update respectively [4]. In RNN, backpropagation step is called backpropagation through time (BPTT) [89] as the gradient of the error must be propagated through the unfolded version of the RNN graph.

3.2.1 Forward propagation

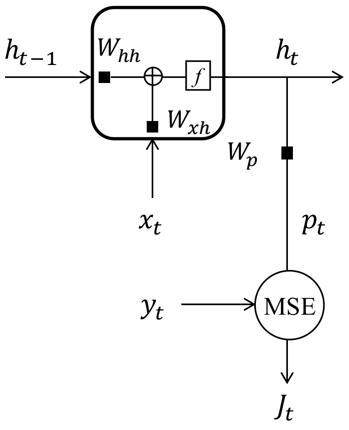


FIGURE 3.2: Forward propagation of a RNN at a time step t. The state $\mathbf{h}_t$, the forecast $\mathbf{p}_t$ and the error J_t are updated with parameters unchanged, such as weights $\{\mathbf{W}_{hh}, \mathbf{W}_{xh}, \mathbf{W}_p\}$ and bias $\{\mathbf{b}_h, \mathbf{b}_x, \mathbf{b}_p\}$, during forward propagation.

$$\text{Target} : \mathbf{y}_t = \mathbf{x}_{t+\Delta t} \tag{3.1}$$

At each time step t, forward propagation of RNN updates values of the state $\mathbf{h}_t$, the forecast $\mathbf{p}_t$ and corresponding error J_t with respect to the input $\mathbf{x}_t$ and target $\mathbf{y}_t$. Note that the parameters, such as weights $\{\mathbf{W}_{hh}, \mathbf{W}_{xh}, \mathbf{W}_p\}$ and bias $\{\mathbf{b}_h, \mathbf{b}_x, \mathbf{b}_p\}$, remain unchanged during the forward propagation. To start forward propagation with sequential training data, initial state $\mathbf{h}_{-1}$ is required to compute the first state $\mathbf{h}_0$. In general, initial state is set to zero [4]. Figure 3.2 illustrates forward propagation process within a cell at a time step t. Equation 3.1 denotes the expression of the target $\mathbf{y}_t$ for STLF tasks, where Δt is the number of time step that the RNN forecasts ahead.

3.2.1.1 Trainable parameters

Trainable parameters can be grouped into two depending on the purpose, such as cell parameters and prediction parameters. The purpose of cell parameters are to update state $\mathbf{h}_t$ using previous state $\mathbf{h}_{t-1}$ and current input $\mathbf{x}_t$. On the other hand, prediction parameters exist to compute the forecast $\mathbf{p}_t$ using the updated state $\mathbf{h}_t$. Using dimension of $\mathbf{x}_t \in \mathbf{R}^m$, $\mathbf{h}_t \in \mathbf{R}^s$ and $\mathbf{p}_t \in \mathbf{R}^m$, the dimension of the trainable parameters is specified.

$$\text{Cell parameters}: \ \mathbf{W}_{xh} \in \mathbf{R}^{s\times m},\ \mathbf{W}_{hh} \in \mathbf{R}^{s\times s},\ \mathbf{b}_h \in \mathbf{R}^s,\ \mathbf{b}_x \in \mathbf{R}^s$$

$$\text{Prediction parameters}: \ \mathbf{W}_p \in \mathbf{R}^{m\times s},\ \mathbf{b}_p \in \mathbf{R}^m$$

3.2.1.2 State $\mathbf{h}_t$

At each time step, new state $\mathbf{h}_t$ is updated by current input $\mathbf{x}_t$ and previous state $\mathbf{h}_{t-1}$. A set of parameters $\{\mathbf{W}_{hh},\ \mathbf{W}_{xh}, \mathbf{b}_h, \mathbf{b}_x\}$ participate in the update of state. Equation 3.2 specifies the relationship, where $f(\cdot)$ is an activation function such that hyperbolic tangent or Relu. Note that all the elements in input $\mathbf{x}_t$ are usually normalized to have z-scores $\mathbf{x}_t \sim N(0, 1)$ or values within [0, 1]. In some cases, the bias $\mathbf{b}_x$ are included to $\mathbf{b}_h$ thus removed in the formal expression.

$$\begin{aligned}\mathbf{h}_t &= f(\mathbf{W}_{hh}\cdot\mathbf{h}_{t-1} + \mathbf{W}_{xh}\cdot\mathbf{x}_t + \mathbf{b}_h + \mathbf{b}_x)\\ &= f(\mathbf{W}_{hh}\cdot\mathbf{h}_{t-1} + \mathbf{W}_{xh}\cdot\mathbf{x}_t + \mathbf{b}_h)\end{aligned} \tag{3.2}$$

3.2.1.3 Forecast $\mathbf{p}_t$

The forecast $\mathbf{p}_t$ is computed with current state $\mathbf{h(t)}$ and prediction parameters $\mathbf{W}_p$ and $\mathbf{b}_p$. It is specified in Equation 3.3, where function $g(\cdot)$ is usually linear.

$$\mathbf{p}_t = g(\mathbf{W}_p \cdot \mathbf{h}_t + \mathbf{b}_p) \tag{3.3}$$

3.2.1.4 Error J and Error at a time J_t

Error J is defined by mean square error (MSE) of the forecast $\mathbf{p}_t$ and target $\mathbf{y}_t$ for all time steps. Analogously, J_t, an error at a time step t, is quadratic error of forecast at the time step. Note that J and J_t are both scalar values. Equation 3.4 denotes the formal expression of J and J_t.

$$\begin{aligned} \text{MSE} : J &= \frac{1}{T}\sum_{t=0}^{T-1} J_t = \frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{m}\{(\mathbf{p}_t)_i - (\mathbf{y}_t)_i\}^2 \\ J_t &= \sum_{i=1}^{m}\{(\mathbf{p}_t)_i - (\mathbf{y}_t)_i\}^2 \end{aligned} \tag{3.4}$$

3.2.2 Backpropagation

The purpose of backpropagation is to compute gradients that will be used for updating parameters. Gradients are derived from error J but have different calculation methods depending on the type of parameters, such as cell parameters which are used for updating the state $\mathbf{h}_t$ and prediction parameters used for computing the forecast $\mathbf{p}_t$. This is because parameters at each group participate in computing the error with different scheme. The chain rule is applied to compute gradients following the inverse direction of the forward scheme during backpropagation.

3.2.2.1 Gradients of J in terms of prediction parameters

As two parameters, $\mathbf{W}_p$ and $\mathbf{b}_p$ participate in computing the forecast using the updated state, they are not engaged in updating the state. Thus, the depth of backpropagation is bounded on the same time step when computing gradient of these parameters. That is, the gradients in terms of the prediction parameters don't backpropagate through time. Fig.3.3 shows how the error at a time J_t is backpropagating to the prediction parameters.

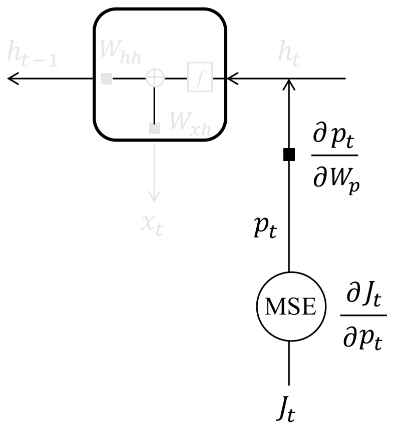

h_{t-1}
W_{hh}
f
h_t
W_{xh}
x_t
$\frac{\partial p_t}{\partial W_p}$
p_t
MSE
$\frac{\partial J_t}{\partial p_t}$
J_t

FIGURE 3.3: Backpropagation scheme for prediction parameters $\{\mathbf{W}_p, \mathbf{b}_p\}$. As the parameters are not engaged in updating states within the cell during forward propagation, the gradients of the error J_t in terms of prediction parameters are bounded within the time step. That is, the gradients do not backpropagate through time. Gradients in terms of the prediction parameters are computed by the chain rule.

The gradient of J in terms of $\mathbf{W}_p$ can be represented by the mean value of the partial derivative of J_t with respect to $\mathbf{W}_p$ over time. By the chain rule, the partial derivative of J_t with respect to $\mathbf{W}_p$ becomes a product of two partial derivatives, the partial derivative of J_t with respect to $\mathbf{p}_t$ and the partial derivative of $\mathbf{p}_t$ with respect to $\mathbf{W}_p$. The former partial derivative is simplified by $2(\mathbf{p}_t - \mathbf{y}_t) \in \mathbf{R}^m$ and the latter one is $\mathbf{h}_t \in \mathbf{R}^s$, assuming that the function $g(\cdot)$ in Equation 3.3 is linear. Considering the dimension of the gradient, cross product is operated between two vectors. Equation 3.5 denotes the formal expression of the gradients, where $\otimes$ represents cross product operation.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}_p} &= \frac{1}{T}\sum_{t=0}^{T-1}\frac{\partial J_t}{\partial \mathbf{W}_p}\\ &= \frac{1}{T}\sum_{t=0}^{T-1}\frac{\partial J_t}{\partial \mathbf{p}_t}\frac{\partial \mathbf{p}_t}{\partial \mathbf{W}_p}\\ &= \frac{2}{T}\sum_{t=0}^{T-1}(\mathbf{p}_t - \mathbf{y}_t)\otimes \mathbf{h}_t\end{aligned} \tag{3.5}$$

Likewise, gradient of the error J with respect to $\mathbf{b}_p$ is,

$$\frac{\partial J}{\partial \mathbf{b}_p} = \frac{2}{T}\sum_{t=0}^{T-1}(\mathbf{p}_t - \mathbf{y}_t) \tag{3.6}$$

3.2.2.2 Gradients of J in terms of cell parameters

The purpose of a RNN cell is to update state $\mathbf{h}_t$ using previous state $\mathbf{h}_{t-1}$ and current input $\mathbf{x}_t$. But properties of the updated state $\mathbf{h}_t$ varies depending on the RNN cell architecture which means a computation process to update a new state $\mathbf{h}_t$ within a RNN cell. There are several RNN cell architectures including LSTM [85], GRU [88] and their variant [90]. Parameters in the RNN cell, referred to cell parameters in the literature, also vary depending on the RNN cell architecture.

The following discussion is based on the cell parameters or ERNN, such as $\{\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{b}_h\}$. Unlike prediction parameters, cell parameters are involved in updating the state $\mathbf{h}_t$ which propagates through time. Cell parameters compute current state $\mathbf{h}_{t-1}$ using previous state $\mathbf{h}_{t-1}$ which is computed by the same parameters and two-step previous state $\mathbf{h}_{t-2}$. This sequential chain lasts until it reaches to the first cell which computes $\mathbf{h_0}$.

Accordingly, the gradients in terms of cell parameters backpropagate through the path that the state $\mathbf{h}_t$ propagates through time. Theoretically, the error can backpropagate to the first cell, which is used to compute the first state $\mathbf{h}_0$. By the chain rule, the procedure that an error at a time step J_t backpropagates to cell parameters in every cell behind the time t can be specified.

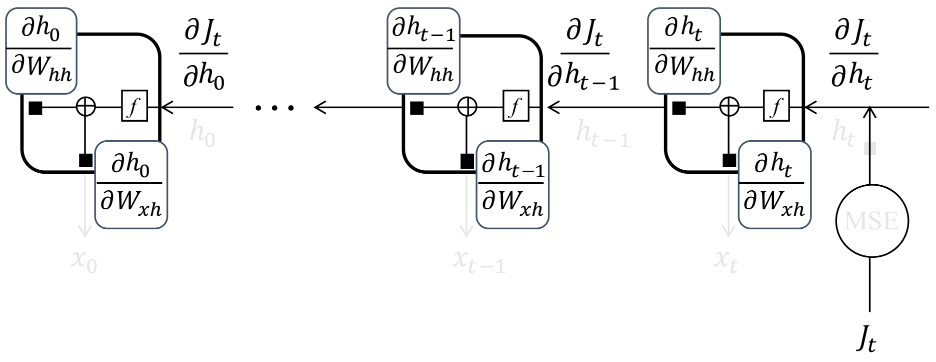

$\frac{\partial h_0}{\partial W_{hh}}$
$\frac{\partial J_t}{\partial h_0}$
$\frac{\partial h_0}{\partial W_{xh}}$
f
h_0
x_0
$\cdots$
$\frac{\partial h_{t-1}}{\partial W_{hh}}$
$\frac{\partial J_t}{\partial h_{t-1}}$
$\frac{\partial h_{t-1}}{\partial W_{xh}}$
h_{t-1}
x_{t-1}
$\frac{\partial h_t}{\partial W_{hh}}$
$\frac{\partial J_t}{\partial h_t}$
$\frac{\partial h_t}{\partial W_{xh}}$
h_t
x_t
MSE
J_t

FIGURE 3.4: Schema of how the error J_t backpropagates to the first cell(unit) through recurrent connection with length 1, which carries gradients of the cell parameters.

Figure 3.4 illustrates how an error at time t, J_t, backpropagates to the cell at time 0. At each time step, the gradient in terms of the state is expressed by the partial derivative of J_t with respect to the state $\mathbf{h}_{t-k}$, where $k = 0, 1, \cdots t$ represents the number of time step that the error backpropagates. The gradients in terms of the cell parameters are derived from the partial derivative at each time step. It implies that the gradients of J_t

in terms of cell parameters differ at each time step, unlike the same kind of parameters have identical values through time in the forward propagation scheme. Hence, for each parameter, the overall gradient is computed by the sum of gradients at every time step in order to maintain the gradient identical through time.

For the parameter $\mathbf{W}_{xh}$ within a cell at time step $t-k$, the gradient of an error J_t is expressed in Equation 3.7 by the chain rule,

$$\begin{aligned}\text{At } t-k^{th} \text{ cell : } \frac{\partial J_t}{\partial \mathbf{W}_{xh}} &= \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{xh}} \\ &= \frac{\partial J_t}{\partial \mathbf{h}_t} \frac{\partial \mathbf{h}_t}{\partial \mathbf{h}_{t-1}} \frac{\partial \mathbf{h}_{t-1}}{\partial \mathbf{h}_{t-2}} \cdots \frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{xh}} \\ &= \frac{\partial J_t}{\partial \mathbf{h}_t} (\prod_{\tau=0}^{k-1} \frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}}) \frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{xh}}\end{aligned} \tag{3.7}$$

The overall gradient in Equation 3.8 is the sum of the gradients at every time step induced by one error J_t. The second last term of Equation 3.8 within braces, products of the partial derivative, is noteworthy because it causes vanishing or exploding gradient problem depending on its value, and that will be discussed later.

$$\text{Overall : } \frac{\partial J_t}{\partial \mathbf{W}_{xh}} = \sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_t} (\prod_{\tau=0}^{k-1} \frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}}) \frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{xh}} \tag{3.8}$$

The gradient of total error J is the average of gradient of an error J_t for T-length sequential data, denoted in Equation 3.9.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}_{xh}} &= \frac{1}{T} \sum_{t=0}^{T-1} \frac{\partial J_t}{\partial \mathbf{W}_{xh}} \\ &= \frac{1}{T} \sum_{t=0}^{T-1} \sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_t} (\prod_{\tau=0}^{k-1} \frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}}) \frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{xh}}\end{aligned} \tag{3.9}$$

Three partial derivative terms in Equation 3.9,

$$\frac{\partial J_t}{\partial \mathbf{h}_t}, \quad \prod_{\tau=0}^{k-1} \frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}}, \quad \frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{xh}}$$

can be expanded using the expressions in forward propagation.

The first term means the gradient of an error J_t in terms of the updated state $\mathbf{h}_t$ at the same time step. Using expressions in Equation 3.3 and 3.4, the term can be expanded in Equation 3.10 ,where $g(\cdot)$ is linear.

$$\frac{\partial J_t}{\partial \mathbf{h}_t} = \frac{\partial J_t}{\partial \mathbf{p}_t}\frac{\partial \mathbf{p}_t}{\partial \mathbf{h}_t} = 2(\mathbf{p}_t - \mathbf{y}_t)\cdot \mathbf{W}_p$$

$$\begin{aligned} \frac{\partial J_t}{\partial \mathbf{p}_t} &= 2(\mathbf{p}_t - \mathbf{y}_t) \\ \frac{\partial \mathbf{p}_t}{\partial \mathbf{h}_t} &= \mathbf{W}_p \end{aligned} \quad \Leftrightarrow \quad \begin{aligned} J_t &= \sum_{i=1}^{m}\{(\mathbf{p}_t)_i - (\mathbf{y}_t)_i\}^2 \quad (3.4) \\ \mathbf{p}_t &= g(\mathbf{W}_p\cdot \mathbf{h}_t + \mathbf{b}_p) \quad (3.3) \end{aligned} \tag{3.10}$$

The second term, the product of the partial derivatives, implies the transmission of the gradient of J_t which backpropagates from time step t to $t-k$. As the gradient backpropagates in sequential manner, the process can be factorized by one time step backpropagation. Considering the formula of updating state $\mathbf{h}_t$ in Equation 3.2 with activation function f as hyperbolic tangent, the product of factorized partial derivative is specified in Equation 3.11 with element-wise product operator $\odot$.

$$\begin{aligned} \prod_{\tau=0}^{k-1}\frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}} &= \prod_{\tau=0}^{k-1}(1-\mathbf{h}_{t-\tau}^2)\cdot \mathbf{W}_{hh} \\ &= \{(1-\mathbf{h}_t^2)\cdot \mathbf{W}_{hh}\}\odot\{(1-\mathbf{h}_{t-1}^2)\cdot \mathbf{W}_{hh}\}\cdots \\ &\qquad \odot\{(1-\mathbf{h}_{t-k+2}^2)\cdot \mathbf{W}_{hh}\}\odot\{(1-\mathbf{h}_{t-k+1}^2)\cdot \mathbf{W}_{hh}\} \end{aligned} \tag{3.11}$$

$$\begin{aligned} &\frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}} = (1-\mathbf{h}_{t-\tau}^2)\cdot \mathbf{W}_{hh} \\ &\Leftrightarrow \quad \mathbf{h}_{t-\tau} = tanh(\mathbf{W}_{hh}\cdot \mathbf{h}_{t-\tau-1} + \mathbf{W}_{xh}\cdot \mathbf{x}_{t-\tau} + \mathbf{b}_h) \quad (3.2) \end{aligned}$$

The third term shows how the cell parameter $\mathbf{W}_{xh}$ is influenced by the backpropagated gradient at time step $t-k$. Using the expression of 3.2, the partial derivative is expanded in Equation 3.12, where $\otimes$ represents cross product operation.

$$\begin{aligned}\frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{xh}} &= (1-\mathbf{h}_{t-k}^2)\otimes \mathbf{x}_{t-k}\\ \Leftrightarrow \quad & \mathbf{h}_{t-k} = tanh(\mathbf{W}_{hh}\cdot \mathbf{h}_{t-k-1} + \mathbf{W}_{xh}\cdot \mathbf{x}_{t-k} + \mathbf{b}_h) \quad (3.2)\end{aligned} \tag{3.12}$$

Thus, Equation 3.9 can be specified in the expression of 3.13 using Equation 3.10, 3.11, and 3.12.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}_{xh}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_t}(\prod_{\tau=0}^{k-1}\frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}})\frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{xh}} \quad (3.9)\\ &= \frac{2}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\{(\mathbf{p}_t-\mathbf{y}_t)\cdot \mathbf{W}_p\}\{\prod_{\tau=0}^{k-1}(1-\mathbf{h}_{t-\tau}^2)\cdot \mathbf{W}_{hh}\}\{(1-\mathbf{h}_{t-k}^2)\otimes \mathbf{x}_{t-k}\}\end{aligned} \tag{3.13}$$

Likewise, gradients in terms of $\mathbf{W}_{hh}$ and $\mathbf{b}_h$ are given in Equation 3.14 and 3.15 respectively.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}_{hh}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_t}(\prod_{\tau=0}^{k-1}\frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}})\frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{W}_{hh}}\\ &= \frac{2}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\{(\mathbf{p}_t-\mathbf{y}_t)\cdot \mathbf{W}_p\}\{\prod_{\tau=0}^{k-1}(1-\mathbf{h}_{t-\tau}^2)\cdot \mathbf{W}_{hh}\}\{(1-\mathbf{h}_{t-k}^2)\otimes \mathbf{h}_{t-k-1}\}\end{aligned} \tag{3.14}$$

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{b}_{h}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_t}(\prod_{\tau=0}^{k-1}\frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}})\frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{b}_{h}}\\ &= \frac{2}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\{(\mathbf{p}_t-\mathbf{y}_t)\cdot \mathbf{W}_p\}\{\prod_{\tau=0}^{k-1}(1-\mathbf{h}_{t-\tau}^2)\cdot \mathbf{W}_{hh}\}(1-\mathbf{h}_{t-k}^2)\end{aligned} \tag{3.15}$$

3.2.3 Parameters Update using Gradient Descent Algorithm

For parameters such that $\mathbf{W}_{hh}$, $\mathbf{W}_{xh}$, $\mathbf{W}_{p}$, $\mathbf{b}_{h}$ and $\mathbf{b}_{p}$, BPTT identifies each partial derivatives with respect to the error J. Identified gradients are used to update parameters by gradient decent algorithm [1] with learning rate μ.

$$\mathbf{W}_{new} = \mathbf{W}_{old} - \mu \frac{\partial J}{\partial \mathbf{W}_{old}} \tag{3.16}$$

3.2.4 Vanishing or Exploding Gradient

To ensure local stability, the network must operate in a ordered regime [91, 92]. However, the product of partial derivatives in Equation 3.11 can cause the gradient to vanish or explode, that deteriorate training procedure. Two factors, length of the time step that error backpropagates and the value of the partial derivative are engaged in the problem [93]. While an error backpropagates from the cell corresponding to the current time step t to the first cell corresponding to $t = 0$, the partial derivative term is multiplied by itself at every time step.

$$f^{'}(t, \tau) = \frac{\partial \mathbf{h}_{t-\tau}}{\partial \mathbf{h}_{t-\tau-1}} \tag{3.17}$$

For the partial derivative term $f^{'}(t, \tau)$ in Equation 3.17, if the error backpropagates through sufficiently long time steps upon condition $|f^{'}(t, \tau)| < 1$, the gradient vanishes to a very small value near zero before computing the appropriate gradient of the parameters in the first cell. On the other hand, an error that backpropagates upon condition $|f^{'}(t, \tau)|$ > 1 many time steps, it explodes to very large value, that causes the parameters never reach their optimal values, which hinder error function converge upon its minimum. In general, the network enters into a chaotic regime, where its computational capability is hindered [94].

[1]Details will be discussed in the Appendix.

Chapter 4

Mini-batch Training in RNNs for STLF

Mini-batch training is a well-known method in neural networks for the advantages in terms of efficiency and robustness. However, to the best of my knowledge, mini-batch training has not been specifically described for STLF tasks using RNNs. This section explains mini-batch training based on the learning algorithm called truncated backpropagation through time (truncated BPTT).

4.1 Truncated BPTT (k_2, k_1)

To avoid vanishing or exploding gradient problem, the time steps that RNNs backpropagate at a time in an unfolded graph is restricted. Roughly, a long sequence should be divided into several chunks with same length that RNNs can learn if the time series is longer than its limit. RNNs repeat forward and backpropagation for chunks that are sequentially fed. It is important to keep the sequential continuity between chunks because the last state at a chunk carries information of the sequence processed so far to the next chunk. Accordingly, shuffling the order of the chunks is not possible, unlike traditional Neural Networks. This learning procedure is called truncated backpropagation through time (truncated BPTT) [95].

Figure 4.1 illustrates a simple scheme of truncated BPTT for a sequence with length T. The figure shows that a long sequence with length T which is divided into two

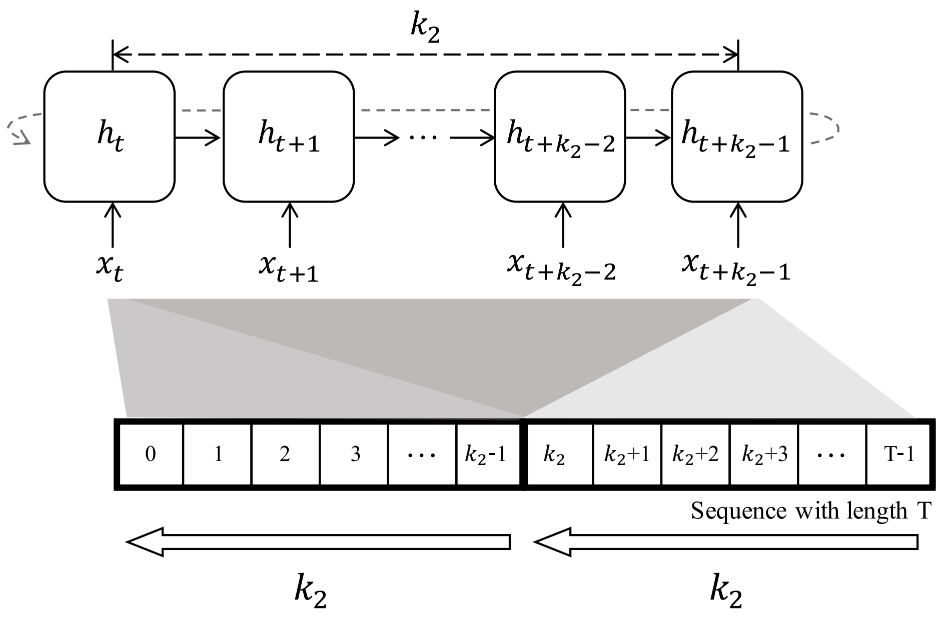


FIGURE 4.1: Folded graph of truncated BPTT(k_2, k_2). The RNNs are fed by truncated sequences with length k_2. The truncated sequences should be fed into network in sequential manner because the last state at a chunk carries information of the sequence processed so far to the next chunk.

chunks with length k_2. The chunks are fed into the RNNs sequentially. The state $\mathbf{h}_{k_2-1}$ computed by the last input of the first chunk $\mathbf{x}_{k_2-1}$ is transferred to the initial state of the second chunk to compute $\mathbf{h}_{k_2}$.

Two hyperparameters, the length of backward pass k_2 and the length of forward pass k_1, are determined a-priori for truncated BPTT.

4.1.1 Length of Backward Pass k_2

The length of the backward pass k_2 is the time steps at which the RNNs backpropagate. The length of k_2 should be determined after considering whether the RNNs enable to backpropagate to k_2 time steps without vanishing or exploding gradient. In general, the RNNs using cells of gated architecture, such as LSTM and GRU, can backpropagate without vanishing gradient farther than the RNNs using Elman cell.

The length that RNNs can backpropagate also depends on the property of the input to learn, such as the expected maximum extent of time dependencies in the sequence or complexity. For example, in a periodic time series with period t, it may be unnecessary or even detrimental to set $k_2 > t$ [4]. Meanwhile, a too small k_2 can increase unnecessary computational cost because it makes the RNNs update parameters too often. The too often parameter update can make the RNNs concentrate more on the local minima within the chunk than global minima which can be obtained based on the long-term dependencies in some cases. Accordingly, it requires more iteration of training until the RNNs converge to the global minima. Therefore, the length of the backward pass k_2 must be carefully tuned to achieve effective training.

4.1.2 Length of Forward Pass k_1

A chunk in Figure 4.1 can be interpreted as a sampled sequence by time window with length k_2, where the window moves forward for every k_2 time steps. The length of the window represents the length of backward pass. The length that the window moves forward is defined as the length of forward pass. Hence, the learning procedure is referred to truncated BPTT(k_2, k_2), where the first and second arguments are the length of backward and forward pass respectively.

Even though truncated BPTT(k_2, k_2) enables the RNNs to learn a long sequence, it still has a drawback. For an arbitrary sequence, truncated BPTT(k_2, k_2) does not guarantee that the gradients for all chunks can be backpropagated to the length of backward pass k_2 without vanishing or exploding. That is, for some chunk in the sequence, the gradient may not be fully backpropagated, that harms the fidelity of learning.

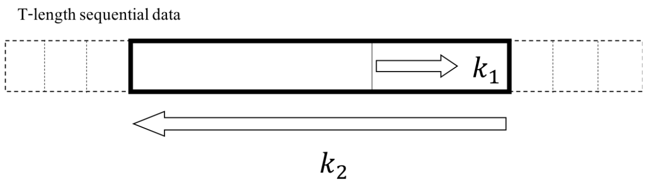


FIGURE 4.2: Procedure of a chunk generation for truncated BPTT(k_2, k_1). The RNNs learns from the chunks with length k_2 and a new chunk created for each k_1 time steps.

The length of forward pass k_1, where $k_2 > k_1 \geq 1$, is introduced to improve the drawback. Truncated BPTT(k_2, k_1) backpropagates to the length of k_2 for every k_1 time steps. Figure 4.2 shows how a long sequence is transformed to chunks to be learned by truncated BPTT(k_2, k_1). Note that a chunk of truncated BPTT(k_2, k_1) have overlapped information of length $k_2 - k_1$ with neighboring chunks, unlike the chunk of truncated BPTT(k_2, k_2). This redundancy, obtained from the overlapped information, alleviates the impact that occurs in the drawback where the gradient is not fully backpropagated.

Truncated BPTT(k_2, 1), referred to true truncated BPTT(k_2), learns the sequence in the most detail but requires expensive computational cost that may be unnecessary. Williams and Peng [95] reports that truncated BPTT($2h$, h) is a good trade-off between accuracy and computational cost, giving comparable accuracy with Truncated BPTT(k_2, 1) and speed advantage.

4.2 Mini-batch Training with Truncated BPTT (k_2, k_1)

Algorithm 1 Pseudocode of mini-batch training with truncated BPTT(k_2,k_1)

```
for i in range(number of iterations) do
  for j in range(number of mini-batches) do
    Forward propagation:compute states and errors of a mini-batch
    Backpropagation:estimate gradients of parameters
    Parameter updates
    Deliver the last states to the next mini-batch
  end for
end for
```

In the following, a technique of mini-batch training for which implements truncated BPTT(k_2, k_1) [95] is described. Mini-batch training is an algorithm that splits the training dataset into small batches that are used to calculate model error and update model parameters. It reduces variance of the gradient of the error and improves the quality and efficiency at computing optimal parameters. The pseudocode explains the procedure of mini-batch training with truncated BPTT(k_2,k_1).

4.2.1 Dimension of a Mini-batch

In the previous section, a chunk of a sequence according to a truncated BPTT (k_2, k_1) is defined as a sequence sampled by a time window of length k_2, where the window moves forward to k_1 time steps at a time. For a sequence $\mathbf{x}_t \in \mathbf{R}^m$, a mini-batch which has three dimension (s_B, k_2, m), where s_B is batch size, is a set of chunks stacked in parallel. Figure 4.3 illustrate the formation of a mini-batch for truncated BPTT(k_2, k_1).

Batch size s_B is defined by the number of multiples of k_1 between zero to k_2. Mathematically, it is a rounded value of $\frac{k_2}{k_1}$ to its upper integer. In practice, batch size s_B shown in Equation 4.1, is always integer as k_2 is defined by the multiple of k_1 .

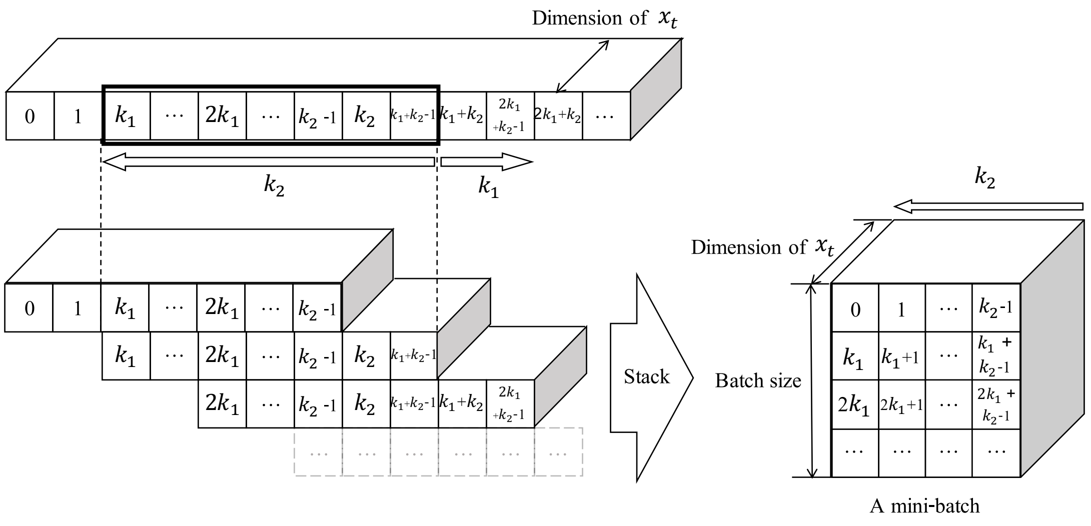


FIGURE 4.3: Formation of a mini-batch for truncated BPTT(k_2, k_1). A mini-batch with dimension (s_B, k_2, m) is a set of chunks stacked in parallel.

$$k_2 = s_B \cdot k_1 \tag{4.1}$$

With respect to the chunks in the first mini-batch, each chunk starts with time step index of k_1 multiples, 0, k_1, $2k_1$, $\cdots$, $(s_B - 1)k_1$. Analogously, the examples in the second mini-batch starts with time step index k_2, $k_2 + k_1$, $k_2 + 2k_1$, $\cdots$, $k_2 + (s_B - 1)k_1$, shown in Figure 4.4.

4.2.2 Number of Mini-batches and Zero Padding

$$\begin{gathered} n_B^{temp} = floor(\frac{T}{k_2}) \\ n_B = \begin{cases} n_B^{temp} + 1 & \text{if } T > n_B^{temp} \cdot k_2 + (s_B - 1)k_1 \\ n_B^{temp} & \text{if } T \leq n_B^{temp} \cdot k_2 + (s_B - 1)k_1 \end{cases} \\ n_Z = n_B k_2 + (s_B - 1)k_1 - T \end{gathered} \tag{4.2}$$

In practice, some part of the sequence can be lost while transforming the sequence into a series of mini-batches depending on the value of k_2 and k_1. It can be improved by padding zeros into the sequence. To compute the number of zeros to be padded, the number of mini-batches n_B should be defined first. The number of batches n_B for a sequence with length T is roughly defined by a rounded value of $\frac{T}{k_2}$ to its lower integer or adding one to the value depending on the condition in Equation 4.2 which denotes the number of batches n_B and the number of zeros to be padded n_Z based on the assumption that k_1 is a divisor of k_2.

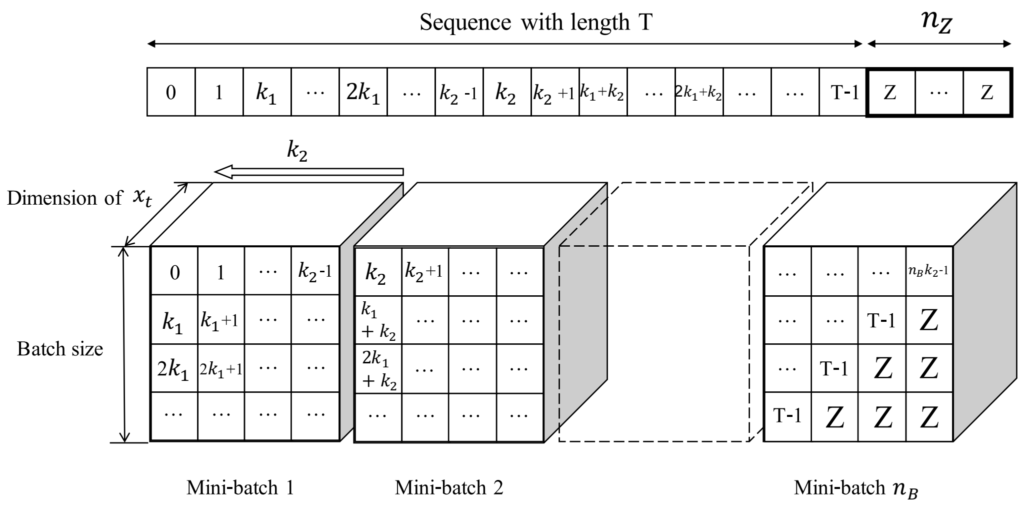


FIGURE 4.4: A series of mini-batches with zero padded sequential data. Each mini-batch starts from the multiples of k_2 and each row within a mini-batch starts the multiples of k_1.

Figure 4.4 illustrates the new sequence generated by concatenating the zero vector with length n_Z to the end of given sequence with length T and mini-batches with the new sequence. As the zero padding part is allocated to the end of the sequence, the last batch n_B contains zeros. Note that identical zero padding and batch generating method must be applied to the target $\mathbf{y}_t$ when training. To eliminate the effect of zero padding, it can be simply done by ignoring the last mini-batch.

4.3 Forward Propagation within a Mini-batch

For STLF tasks, forward propagation updates the state $\mathbf{h}_t$, the forecast $\mathbf{p}_t$, and errors J_t at each time step over the mini-batch. Figure 4.5 describes how a mini-batch of input, $\mathbf{X}_b$ computes the forecast $\mathbf{P}_b$ by truncated BPTT(k_2,k_1). An input mini-batch $\mathbf{X}_b$ is split into k_2 layers along the axis of sequential time step and each layer is fed into the RNNs. RNNs compute state $\mathbf{H}_b$ using the input $\mathbf{X}_b$.

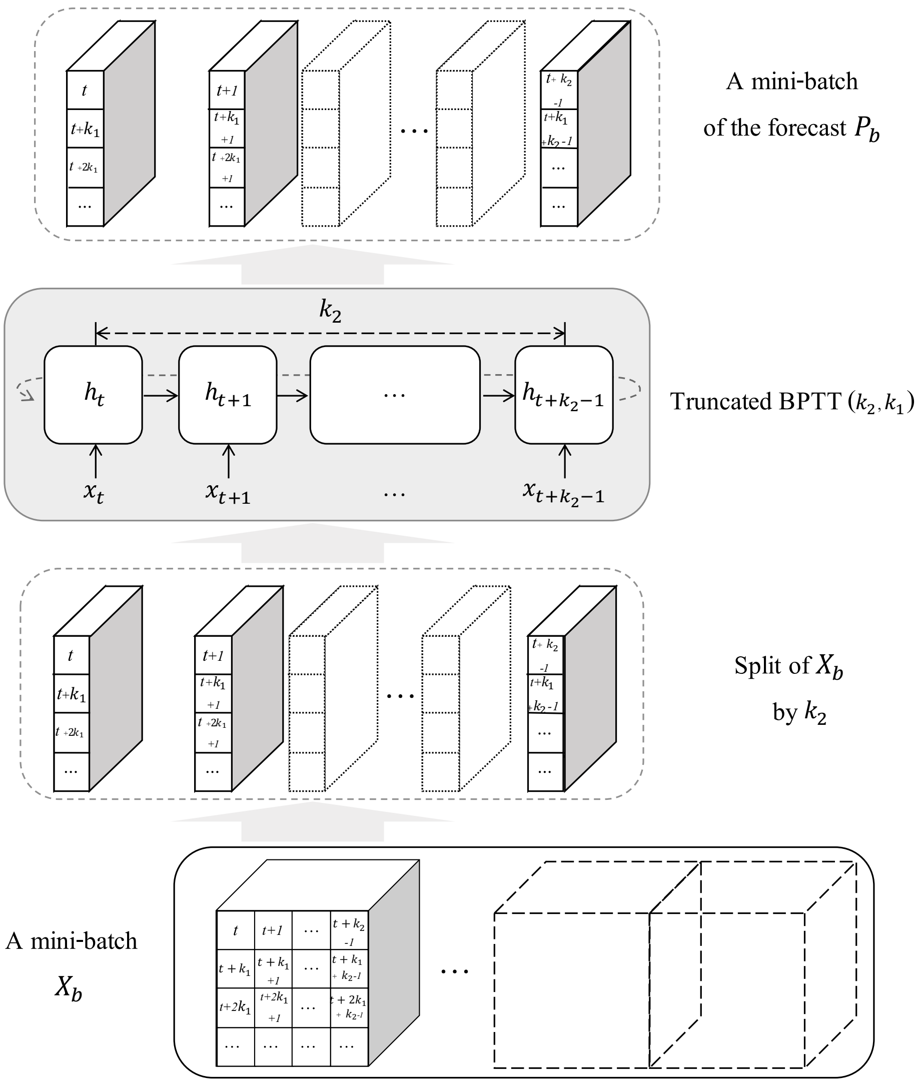


FIGURE 4.5: Forward propagation with a mini-batch for truncated BPTT(k_2, k_1). For input a mini-batch $\mathbf{X}_b$, it is split along the axis of time step k_2 and fed into the RNNs. Truncated BPTT(k_2, k_1) returns a mini-batch of state $\mathbf{H}_b$ which computes forecast mini-batch $\mathbf{P}_b$ by FFNNs.

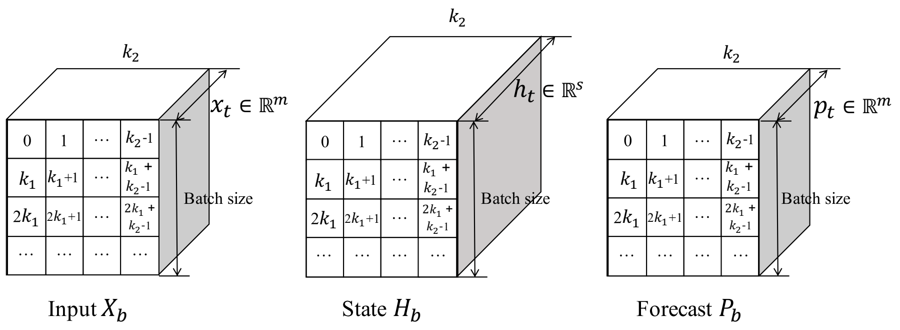


FIGURE 4.6: Dimension of mini-batches that are used in truncated BPTT(k_2, k_1) : input $\mathbf{X}_b$, state $\mathbf{H}_b$ and the forecast $\mathbf{P}_b$

The forecast mini-batch $\mathbf{P}_b$, having the same dimension with the input mini-batch $\mathbf{X}_b$, is obtained from the state $\mathbf{H}_b$. Figure 4.6 depicts the each dimension of the mini-batch of input $\mathbf{X}_b$, state $\mathbf{H}_b$ and the forecast $\mathbf{P}_b$. Note that the propagation is available only along the axis of sequential time step, neither along the axis of batch size nor along the dimension axis. The state from the last cell will be kept to carry information processed so far to the next mini-batch.

4.3.1 Mini-batch Error J_b

Mini-batch error J_b is defined by the total sum of mean square error of all elements along three axes of the mini-batch (s_B, k_2, m).

$$J_b = \frac{1}{k_2 \cdot s_B} \sum_{j \in s_B} \sum_{t=0}^{k_2-1} \sum_{i=1}^{m} \{(\mathbf{p}_t)_{ji} - (\mathbf{y}_t)_{ji}\}^2 \tag{4.3}$$

4.4 Backpropagation within a Mini-batch

4.4.1 Gradients of J_b in terms of Prediction Parameters

Truncated BPTT(k_2, k_1) follows the same procedure of traditional BPTT explained above except for the length of backward pass k_2 and mini-batch. The gradient of batch error J_b in terms of prediction parameters $\{\mathbf{W}_p, \mathbf{b}_p\}$ are computed by the average of gradients not only along the backward pass axis but also along the batch size axis, which is specified in Equation 4.4.

$$\begin{aligned} \frac{\partial J_b}{\partial \mathbf{W}_p} &= \frac{2}{k_2 \cdot s_B} \sum_{j \in s_B} \sum_{t=0}^{k_2-1} \{(\mathbf{p}_t)_j - (\mathbf{y}_t)_j\} \otimes (\mathbf{h}_t)_j \\ \frac{\partial J_b}{\partial \mathbf{b}_p} &= \frac{2}{k_2 \cdot s_B} \sum_{j \in s_B} \sum_{t=0}^{k_2-1} \{(\mathbf{p}_t)_j - (\mathbf{y}_t)_j\} \end{aligned} \tag{4.4}$$

4.4.2 Gradients ofJ_b in terms of Cell Parameters

Gradients of batch error J_b in terms of cell parameters $\{\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{b}_h\}$ can be shown by the same procedure with conventional BPTT. The gradients of cell parameters backpropagate through time but the maximum depth of the time is bounded by k_2. That is, the gradients don't backpropagate over the mini-batches.

$$\begin{aligned} \frac{\partial J_b}{\partial \mathbf{W}_{xh}} &= \frac{2}{k_2 \cdot s_B} \sum_{j \in s_B} \sum_{t=0}^{k_2-1} \sum_{k=0}^{t} \left[\{(\mathbf{p}_t)_j - (\mathbf{y}_t)_j\} \cdot \mathbf{W}_p\}\right] \\ &\qquad \left[\prod_{\tau=0}^{k-1} \{1 - (\mathbf{h}_{t-\tau}^2)_j\} \cdot \mathbf{W}_{hh}\right] \left[\{1 - (\mathbf{h}_{t-k}^2)_j\} \otimes (\mathbf{x}_{t-k})_j\right] \\ \frac{\partial J_b}{\partial \mathbf{W}_{hh}} &= \frac{2}{k_2 \cdot s_B} \sum_{j \in s_B} \sum_{t=0}^{k_2-1} \sum_{k=0}^{t} \left[\{(\mathbf{p}_t)_j - (\mathbf{y}_t)_j\} \cdot \mathbf{W}_p\}\right] \\ &\qquad \left[\prod_{\tau=0}^{k-1} \{1 - (\mathbf{h}_{t-\tau}^2)_j\} \cdot \mathbf{W}_{hh}\right] \left[\{1 - (\mathbf{h}_{t-k}^2)_j\} \otimes (\mathbf{x}_{h-k-1})_j\right] \\ \frac{\partial J_b}{\partial \mathbf{b}_h} &= \frac{2}{k_2 \cdot s_B} \sum_{j \in s_B} \sum_{t=0}^{k_2-1} \sum_{k=0}^{t} \left[\{(\mathbf{p}_t)_j - (\mathbf{y}_t)_j\} \cdot \mathbf{W}_p\}\right] \\ &\qquad \left[\prod_{\tau=0}^{k-1} \{1 - (\mathbf{h}_{t-\tau}^2)_j\} \cdot \mathbf{W}_{hh}\right] \{1 - (\mathbf{h}_{t-k}^2)_j\} \end{aligned} \tag{4.5}$$

Chapter 5

Advanced RNN Architectures

This chapter describes two advanced RNN units, long short-term memory (LSTM) and gated recurrent unit (GRU), which improve vanishing gradient problem of ERNN.

The architecture of ERNN cell, specifically non-linear activation function $f(\cdot)$ of state $\mathbf{h}_t$, causes vanishing or exploding gradient problem when the gradient backpropagates through time because it generates the product of partial derivative of $\mathbf{h}_{t-\tau}$ with respect to $\mathbf{h}_{t-\tau-1}$ for every backward time step. Truncated BPTT(k_2, k_1)[95] is one solution for this problem. However, due to the truncation, the RNNs don't backpropagate longer than k_2. It may cause another problem where the RNNs can not capture the long-term dependencies greater than k_2 in the training.

LSTM and GRU both provide longer range where the RNNs can learn without vanishing or exploding gradient by changing the architecture the RNN cell. The new approaches both utilized gated structure that enables the networks update information without non-linear activation function.

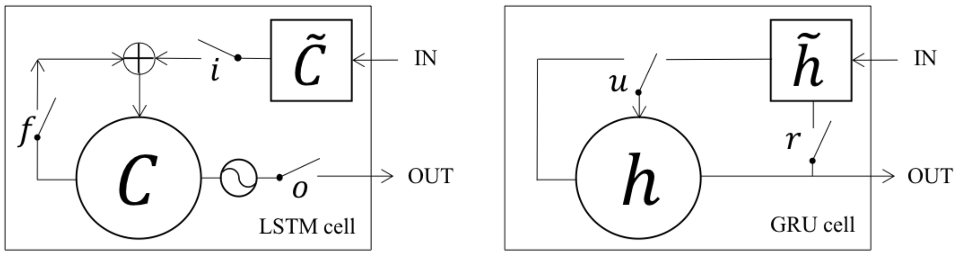


Figure 5.1: Schema of two RNN cells, LSTM (left) and GRU (right). GRU has a simpler architecture with the less number of gates than LSTM.

5.1 Long Short-term Memory (LSTM)

For LSTM, cell state $\mathbf{C}_t$ and three gate layers, such as forget gate, input gate and output gate, are newly introduced[85]. Cell state inherits information from previous cell and conveys it to the next cell. The gates determine the amount of information to to inherit, update and convey to the next cell by interacting with the cell state. Figure 5.2 shows the unfolded graph of a RNN with LSTM cells.

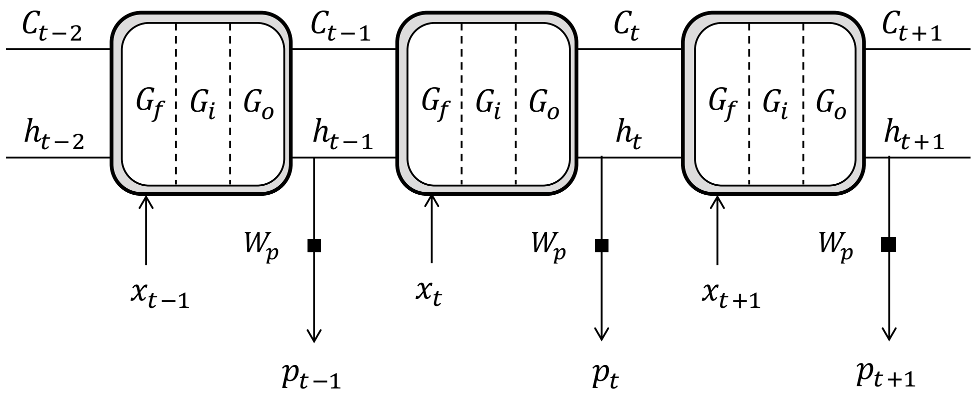


FIGURE 5.2: Unfolded graph of a RNN with LSTM cell that consist of three gates.

5.2 Training LSTM

5.2.1 Forward Propagation

Figure 5.3 depicts the architecture of LSTM cell when the RNNs propagate forward. There is an important bypass on top of the cell where cell state $\mathbf{C}_t$ flows. As the bypass doesn't have any activation functions, such as hyperbolic tangent or sigmoid $\sigma(\cdot)$, it is less influenced by the vanishing or exploding gradient caused by the product of partial derivatives in the learning procedure.

5.2.1.1 Variables and trainable parameters

The following table shows all trainable parameters and variables of LSTM cell with dimensional information.

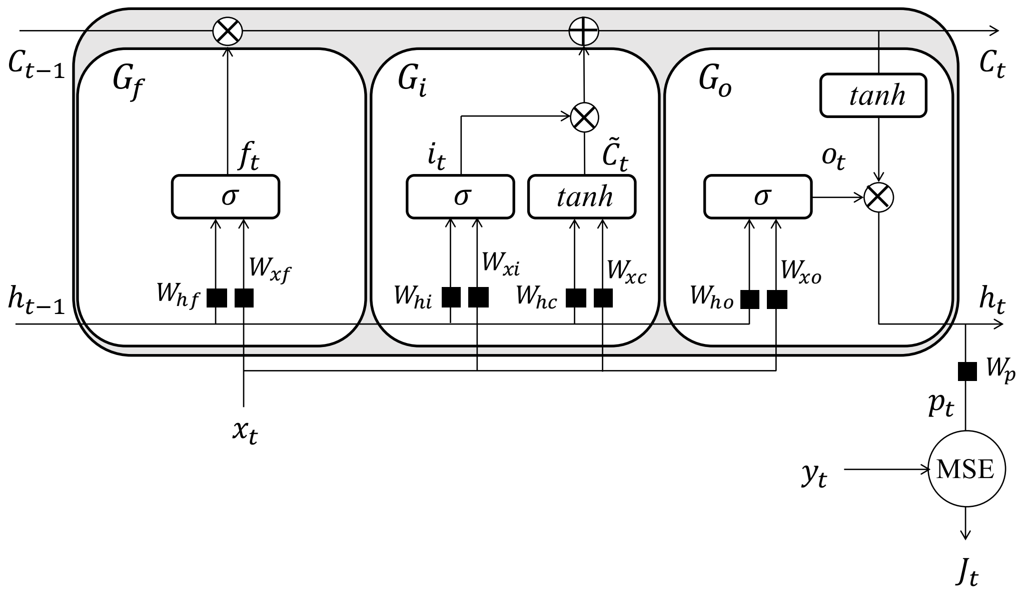


FIGURE 5.3: LSTM cell architecture. Note that the bypass without non-linear activation function for cell state $\mathbf{C}_t$ enables to avoid vanishing or exploding gradient problem and backpropagate the gradients to the further past.

Gate type		Variable	Parameters		
Forget gate	G_f	$\mathbf{f}_t \in \mathbf{R}^s$	$\mathbf{W}_{xf} \in \mathbf{R}^{s\times m}$	$\mathbf{W}_{hf} \in \mathbf{R}^{s\times s}$	$\mathbf{b}_f \in \mathbf{R}^s$
Input gate	G_i	$\mathbf{i}_t \in \mathbf{R}^s$	$\mathbf{W}_{xi} \in \mathbf{R}^{s\times m}$	$\mathbf{W}_{hi} \in \mathbf{R}^{s\times s}$	$\mathbf{b}_i \in \mathbf{R}^s$
		$\tilde{\mathbf{C}}_t \in \mathbf{R}^s$	$\mathbf{W}_{xc} \in \mathbf{R}^{s\times m}$	$\mathbf{W}_{hc} \in \mathbf{R}^{s\times s}$	$\mathbf{b}_c \in \mathbf{R}^s$
Output gate	G_o	$\mathbf{o}_t \in \mathbf{R}^s$	$\mathbf{W}_{xo} \in \mathbf{R}^{s\times m}$	$\mathbf{W}_{ho} \in \mathbf{R}^{s\times s}$	$\mathbf{b}_o \in \mathbf{R}^s$
Prediction		$\mathbf{p}_t \in \mathbf{R}^m$	$\mathbf{W}_p \in \mathbf{R}^{m\times s}$		$\mathbf{b}_p \in \mathbf{R}^m$

TABLE 5.1: Variables and trainable parameters of LSTM cell

5.2.1.2 Cell State $\mathbf{C}_t$

Cell state, which conveys the processed information so far to the next cell, plays a similar role like state $\mathbf{h}_t$ in ERNN. However, the state $\mathbf{h}_t$ in ERNN has non-linear property due to the activation function $f(\cdot)$ while cell state $\mathbf{C}_t$ in LSTM is expressed by a linear combination as shown in Equation 5.1.

$$\mathbf{C}_t = \mathbf{f}_t \odot \mathbf{C}_{t-1} + \mathbf{i}_t \odot \tilde{\mathbf{C}}_t \tag{5.1}$$

The linear property brings a promising result during BPTT phase. As its partial derivative doesn't generate product terms, LSTM doesn't suffer from vanishing or exploding gradient problem as ERNN does.

Instead, cell state is regulated by three variables $\mathbf{f}_t$, $\mathbf{i}_t$ and $\tilde{\mathbf{C}}_t$ which come from forget and input gate. For extreme case in Equation 5.1 such that $\mathbf{f}_t = 1$ and $\mathbf{i}_t = 0$, $\mathbf{C}_t$ only inherits from its previous cell state $\mathbf{C}_{t-1}$. In other words, cell state keeps same information between t and $t-1$, which can be interpreted as short-term memory. Furthermore, by optimizing parameters regarding the variables, the short-term memory can last for long period of time. Thus, for ideally trained LSTM network with a sequence with length T, cell state can reserve closely identical information through T-cells, that is why this RNN model is called long short-term memory.

5.2.1.3 Forget Gate Variable $\mathbf{f}_t$

$$\mathbf{f}_t = \sigma(\mathbf{W}_{hf} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xf} \cdot \mathbf{x}_t + \mathbf{b}_f) \tag{5.2}$$

Forget gate returns the variable $\mathbf{f}_t$. The range of the element values of $\mathbf{f}_t$ is restricted within 0 and 1 due to the sigmoid function $\sigma(\cdot)$ in Equation 5.2. As shown in Equation 5.1, $\mathbf{f}_t$ determines the portion of the previous cell state $\mathbf{C}_{t-1}$ to be inherited by the current cell state $\mathbf{C}_t$ by the element-wise multiplication.

5.2.1.4 Input Gate Variable $\mathbf{i}_t$ and Candidate $\tilde{\mathbf{C}}_t$

$$\begin{aligned} \tilde{\mathbf{C}}_t &= tanh(\mathbf{W}_{hc} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xc} \cdot \mathbf{x}_t + \mathbf{b}_c) \\ \mathbf{i}_t &= \sigma(\mathbf{W}_{hi} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xi} \cdot \mathbf{x}_t + \mathbf{b}_i) \end{aligned} \tag{5.3}$$

Input gate generates a variable $\mathbf{i}_t$ and a candidate $\tilde{\mathbf{C}}_t$. The element-wise product of the variable and the candidate participates in updating cell state $\mathbf{C}_t$ by adding to the element-wise product of $\mathbf{f}_t$ and $\mathbf{C}_{t-1}$ as shown in Equation 5.1.

The candidate $\tilde{\mathbf{C}}_t$ plays a role to collect and update new information from the training data $\mathbf{x}_t$, like $\mathbf{h}_t$ in ERNN. However, unlike $\mathbf{h}_t$ of ERNN which conveys information to the next cell directly, the candidate $\tilde{\mathbf{C}}_t$ is adjusted by the variable $\mathbf{i}_t$ which element values spanned within 0 to 1. As definition of $\mathbf{i}_t$, shown in Equation 5.3, is identical to the definition of $\mathbf{f}_t$, it plays a similar role like $\mathbf{f}_t$, determining the portion of the candidate $\tilde{\mathbf{C}}_t$ to be updated to the cell state $\mathbf{C}_t$.

5.2.1.5 State $\mathbf{h}_t$ and Output Gate Variable $\mathbf{o}_t$

$$\begin{aligned} \mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{C}_t) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_{ho} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xo} \cdot \mathbf{x}_t + \mathbf{b}_o) \end{aligned} \tag{5.4}$$

Output gate generates state $\mathbf{h}_t$ and a variable $\mathbf{o}_t$ as shown in Equation 5.4. State $\mathbf{h}_t$ delivers information to the next cell using a different path as cell state does.

However, the information delivered by $\mathbf{h}_t$ is originated from the cell state $\mathbf{C}_t$ which keeps processed information so far. As the cell state has only linear property, state $\mathbf{h}_t$ requires activation function that enables the network to keep the non-linear property that allows the cells to be connected in series along the time. Therefore, hyperbolic tangent is applied to the cell state as an activation function, as shown in Equation 5.4.

The variable $\mathbf{o}_t$ determines the portion of $tanh(\mathbf{C}_t)$ to be delivered to the next cell by element-wise multiplication, as $\mathbf{f}_t$ and $\mathbf{i}_t$ do. These three variables can be interpreted as a dimming switch and their operating characteristic is defined by their parameters which are optimized during backpropagation phase.

5.2.1.6 Forecast $\mathbf{p}_t$ and error J

As the forecast $\mathbf{p}_t$ and error J_t are not included in LSTM cell, these expressions are identical to the ones for ERNN. Equation 5.5 is the repetition of 3.3 and 3.4.

$$\begin{aligned} \mathbf{p}_t &= g(\mathbf{W}_p \cdot \mathbf{h}_t + \mathbf{b}_p) \\ J &= \frac{1}{T}\sum_{t=0}^{T-1} J_t = \frac{1}{T}\sum_{t=0}^{T-1}\sum_{i=1}^{m}\{(\mathbf{p}_t)_i - (\mathbf{y}_t)_i\}^2 \end{aligned} \tag{5.5}$$

5.2.2 Back Propagation

For the LSTM network, parameters can be discriminated into two groups depending on the approach of computing gradients, prediction parameters, and cell parameters. Computing gradients of the prediction parameters such as $\mathbf{W}_p$, doesn't require BPTT. On the other hand, computing gradients of the cell parameters, such as $\mathbf{W}_{xf}$, $\mathbf{W}_{xi}$, $\mathbf{W}_{xc}$, $\mathbf{W}_{xo}$ etc, requires BPTT.

In terms of BPTT for the cell parameters, the gradients backpropagate through two paths, cell state $\mathbf{C}_{t-k}$ and state $\mathbf{h}_{t-k}$, where index $k = 0, 1, \cdots t$ denotes the number of time step that the gradients backpropagate. For the cell corresponding the $t - k^{th}$ time step which counts from the beginning, the gradients in terms of cell parameters are only dependent on the gradients in terms of cell state $\mathbf{C}_{t-k}$ and state $\mathbf{h}_{t-k}$.

In this section describes the gradients in terms of cell parameters following by the discussion of the gradients in terms of prediction parameters. Cell parameters are distinguished to two, depending on the paths where the gradients backpropagate, cell state $\mathbf{C}_{t-k}$ or state $\mathbf{h}_{t-k}$.

5.2.2.1 Gradients of J in terms of Prediction Parameters

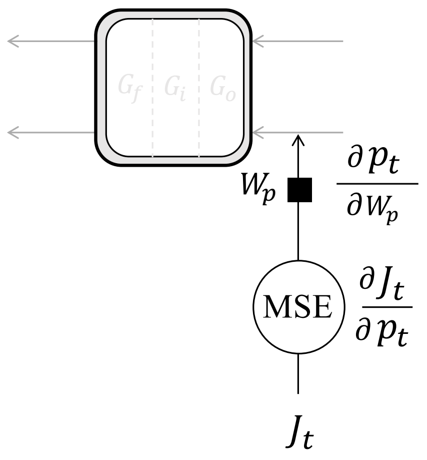

G_f
G_i
G_o
W_p
$\frac{\partial p_t}{\partial W_p}$
MSE
$\frac{\partial J_t}{\partial p_t}$
J_t

FIGURE 5.4: Gradient of J_t in terms of prediction parameters $\{\mathbf{W}_p, \mathbf{b}_p\}$. As prediction parameters locate out of the recurrent connections that enable the information over time, the error at a time step J_t only influences prediction parameters at the same time step, that is, the gradients of prediction parameters don't backpropagate through time.

Figure 5.4 shows how the error at one time step t is back-propagating to the prediction parameters. As illustrated in the figure, prediction parameters are located out of the LSTM cell. Thus, the parameters are not influenced by the gradients that backpropagate through time, unlike cell parameters and their backpropagation scheme is identical with the one of ERNN in Chapter 2. For a sequence with length T, the gradients of total error J in terms of the prediction parameters are given in Equation 5.6.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}_p} &= \frac{1}{T}\sum_{t=0}^{T-1}\frac{\partial J_t}{\partial \mathbf{W}_p} = \frac{2}{T}\sum_{t=0}^{T-1}(\mathbf{p}_t - \mathbf{y}_t) \otimes \mathbf{h}_t \\ \frac{\partial J}{\partial \mathbf{b}_p} &= \frac{1}{T}\sum_{t=0}^{T-1}\frac{\partial J_t}{\partial \mathbf{b}_p} = \frac{2}{T}\sum_{t=0}^{T-1}(\mathbf{p}_t - \mathbf{y}_t)\end{aligned} \tag{5.6}$$

5.2.2.2 Gradients of J in terms of Cell Parameters

For the $t-k^{th}$ cell from the beginning, gradients in terms of cell parameters can be discriminated into two groups depending on the path where the gradients backpropagate : Path A, gradients through state $\mathbf{h}_{t-k}$ (See Figure 5.5) and path B, gradients through cell state $\mathbf{C}_{t-k}$ (See Figure 5.6). The parameters in the output gate are involved in the path A, while the parameters in the input gate and forget gate are involved in the path B.

· **Path A : Gradients through $\mathbf{h}_{t-k}$**

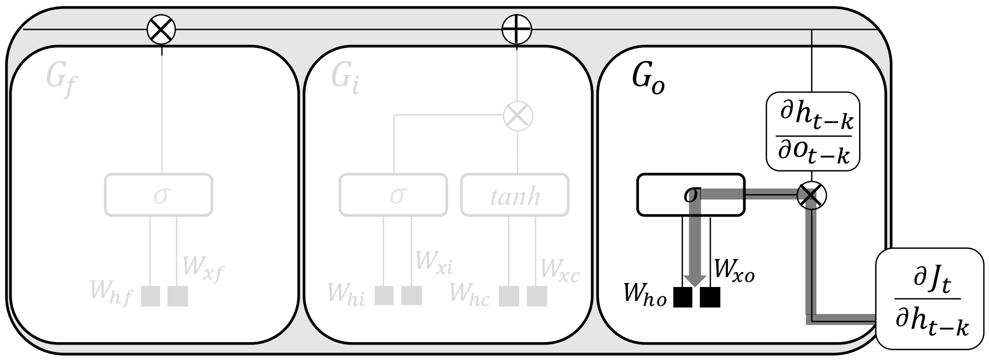


FIGURE 5.5: Back-propagating error through state $\mathbf{h}_{t-k}$. Output gate G_o only involves in this backpropagation. Gradients in terms of output gate parameters $\{\mathbf{W}_{xo},, \mathbf{W}_{ho}, \mathbf{b}_o\}$ can be expressed by the chain rule that is derived from the partial derivative of an error J_t with respect to $\mathbf{h}_{t-k}$ and $\mathbf{h}_{t-k}$ with respect to $\mathbf{o}_{t-k}$.

The gradients of output gate parameters, such as $\mathbf{W}_{xo}$, $\mathbf{W}_{ho}$ and $\mathbf{b}_o$, backpropagate through the path of $\mathbf{h}_{t-k}$ and $\mathbf{o}_{t-k}$. Overall gradient for a sequence with length T is expressed by the chain rule of three partial derivatives in Equation 5.7, where the last two partial derivatives are expanded in Equation 5.8. Discussion regarding the first partial derivative $\frac{\partial J_t}{\partial \mathbf{h}_{t-k}}$ will be followed later.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}_{xo}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}}\frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{o}_{t-k}}\frac{\partial \mathbf{o}_{t-k}}{\partial \mathbf{W}_{xo}} \\ &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot tanh(\mathbf{C}_{t-k}) \odot \mathbf{o}_{t-k} \odot (\mathbf{1}-\mathbf{o}_{t-k}) \otimes \mathbf{x}_{t-k}\end{aligned} \tag{5.7}$$

$$\begin{aligned}\frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{o}_{t-k}} &= tanh(\mathbf{C}_{t-k}) \\ \frac{\partial \mathbf{o}_{t-k}}{\partial \mathbf{W}_{xo}} &= \mathbf{o}_{t-k} \odot (\mathbf{1}-\mathbf{o}_{t-k}) \otimes \mathbf{x}_{t-k}\end{aligned} \quad \Longleftrightarrow \quad \begin{aligned}\mathbf{h}_t &= \mathbf{o}_t \odot tanh(\mathbf{C}_t) \\ \mathbf{o}_t &= \sigma(\mathbf{W}_{ho}\cdot\mathbf{h}_{t-1} + \mathbf{W}_{xo}\cdot\mathbf{x}_t + \mathbf{b}_o)\end{aligned} \tag{5.8}$$

For other parameters in the output gate $\{\mathbf{W}_{ho}, \mathbf{b}_o\}$, their gradients are in Equation 5.9.

$$\begin{aligned}\frac{\partial J}{\partial \mathbf{W}_{ho}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot tanh(\mathbf{C}_{t-k}) \odot \mathbf{o}_{t-k} \odot (\mathbf{1}-\mathbf{o}_{t-k}) \otimes \mathbf{h}_{t-k-1} \\ \frac{\partial J}{\partial \mathbf{b}_{o}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot tanh(\mathbf{C}_{t-k}) \odot \mathbf{o}_{t-k} \odot (\mathbf{1}-\mathbf{o}_{t-k})\end{aligned} \tag{5.9}$$

· **Path B : Gradients through $\mathbf{C}_{t-k}$**

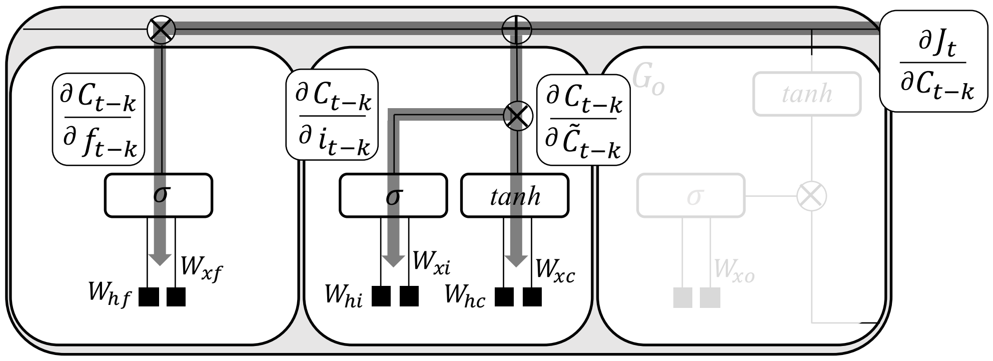

$\frac{\partial J_t}{\partial C_{t-k}}$
$\frac{\partial C_{t-k}}{\partial f_{t-k}}$
$\frac{\partial C_{t-k}}{\partial i_{t-k}}$
$\frac{\partial C_{t-k}}{\partial \tilde{C}_{t-k}}$
G_o
tanh
σ
σ
tanh
σ
W_{hf}
W_{xf}
W_{hi}
W_{xi}
W_{hc}
W_{xc}
W_{xo}

FIGURE 5.6: Back-propagating error through state $\mathbf{C}_{t-k}$. Forget and input gate, G_f and G_i, involve in the backpropagation. Gradients in terms of parameters can be expressed by the chain rule that is derived from the partial derivative of an error J_t with respect to $\mathbf{C}_{t-k}$ and others depending on the parameters.

As the parameters in forget and input gate contribute to update cell state in the forward propagation, the gradients backpropagate through the same path. But the effect of the gradient through the cell state differs between input and forget gates.

For the parameters in input gate $\{\mathbf{W}_{hi}, \mathbf{W}_{xi}, \mathbf{W}_{hc}, \mathbf{W}_{xc}, \mathbf{b}_i, \mathbf{b}_c\}$, the backpropagating gradients branch off into two, one for the variable $\mathbf{i}_{t-k}$ and one for the candidate $\tilde{\mathbf{C}}_{t-k}$. As they experience different activation functions and paths, their expressions differ. Overall gradients of input gate parameters for a sequence with length T is expressed in Equation 5.10 and 5.11.

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{W}_{xi}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}}\frac{\partial \mathbf{C}_{t-k}}{\partial \mathbf{i}_{t-k}}\frac{\partial \mathbf{i}_{t-k}}{\partial \mathbf{W}_{xi}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \tilde{\mathbf{C}}_{t-k} \odot \mathbf{i}_{t-k} \odot (\mathbf{1} - \mathbf{i}_{t-k}) \otimes \mathbf{x}_{t-k} \\
\frac{\partial J}{\partial \mathbf{W}_{hi}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}}\frac{\partial \mathbf{C}_{t-k}}{\partial \mathbf{i}_{t-k}}\frac{\partial \mathbf{i}_{t-k}}{\partial \mathbf{W}_{hi}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \tilde{\mathbf{C}}_{t-k} \odot \mathbf{i}_{t-k} \odot (\mathbf{1} - \mathbf{i}_{t-k}) \otimes \mathbf{h}_{t-k-1} \\
\frac{\partial J}{\partial \mathbf{b}_{i}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}}\frac{\partial \mathbf{C}_{t-k}}{\partial \mathbf{i}_{t-k}}\frac{\partial \mathbf{i}_{t-k}}{\partial \mathbf{b}_{i}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \tilde{\mathbf{C}}_{t-k} \odot \mathbf{i}_{t-k} \odot (\mathbf{1} - \mathbf{i}_{t-k})
\end{aligned} \tag{5.10}$$

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{W}_{xc}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}}\frac{\partial \mathbf{C}_{t-k}}{\partial \tilde{\mathbf{C}}_{t-k}}\frac{\partial \tilde{\mathbf{C}}_{t-k}}{\partial \mathbf{W}_{xc}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \mathbf{i}_{t-k} \odot (\mathbf{1} - \tilde{\mathbf{C}}_{t-k}^2) \otimes \mathbf{x}_{t-k} \\
\frac{\partial J}{\partial \mathbf{W}_{hc}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}}\frac{\partial \mathbf{C}_{t-k}}{\partial \tilde{\mathbf{C}}_{t-k}}\frac{\partial \tilde{\mathbf{C}}_{t-k}}{\partial \mathbf{W}_{hc}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \mathbf{i}_{t-k} \odot (\mathbf{1} - \tilde{\mathbf{C}}_{t-k}^2) \otimes \mathbf{h}_{t-k-1} \\
\frac{\partial J}{\partial \mathbf{b}_{c}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}}\frac{\partial \mathbf{C}_{t-k}}{\partial \tilde{\mathbf{C}}_{t-k}}\frac{\partial \tilde{\mathbf{C}}_{t-k}}{\partial \mathbf{b}_{c}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \mathbf{i}_{t-k} \odot (\mathbf{1} - \tilde{\mathbf{C}}_{t-k}^2)
\end{aligned} \tag{5.11}$$

For the parameters in forget gate $\{\mathbf{W}_{hf}, \mathbf{W}_{xf}, \mathbf{b}_f\}$, the gradients backpropagate through

$\mathbf{C}_t$ and $\mathbf{f}_t$. Overall gradients of forget gate parameters for a sequence with length T is expressed in Equation 5.12.

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{W}_{xf}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \frac{\partial \mathbf{C}_{t-k}}{\partial \mathbf{f}_{t-k}} \frac{\partial \mathbf{f}_{t-k}}{\partial \mathbf{W}_{xf}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \mathbf{C}_{t-k-1} \odot \mathbf{f}_{t-k} \odot (\mathbf{1} - \mathbf{f}_{t-k}) \otimes \mathbf{x}_{t-k} \\
\frac{\partial J}{\partial \mathbf{W}_{hf}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \frac{\partial \mathbf{C}_{t-k}}{\partial \mathbf{f}_{t-k}} \frac{\partial \mathbf{f}_{t-k}}{\partial \mathbf{W}_{hf}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \mathbf{C}_{t-k-1} \odot \mathbf{f}_{t-k} \odot (\mathbf{1} - \mathbf{f}_{t-k}) \otimes \mathbf{h}_{t-k-1} \\
\frac{\partial J}{\partial \mathbf{b}_{f}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \frac{\partial \mathbf{C}_{t-k}}{\partial \mathbf{f}_{t-k}} \frac{\partial \mathbf{f}_{t-k}}{\partial \mathbf{b}_{f}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{C}_{t-k}} \odot \mathbf{C}_{t-k-1} \odot \mathbf{f}_{t-k} \odot (\mathbf{1} - \mathbf{f}_{t-k})
\end{aligned} \tag{5.12}$$

5.2.2.3 Backpropagating Gradients between Neighboring Time Steps

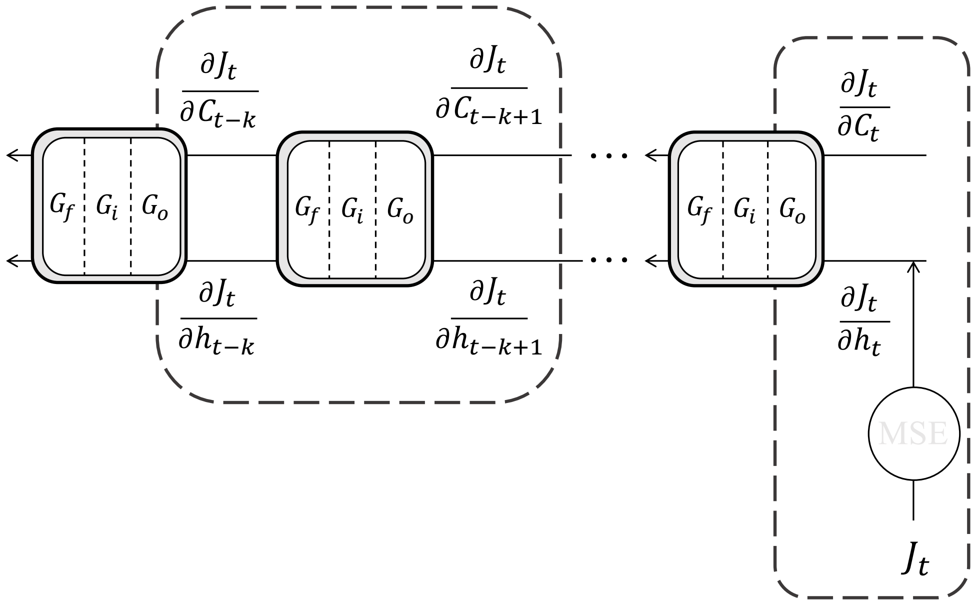

$\frac{\partial J_t}{\partial C_{t-k}}$
$\frac{\partial J_t}{\partial C_{t-k+1}}$
$\frac{\partial J_t}{\partial C_t}$
G_f
G_i
G_o
G_f
G_i
G_o
G_f
G_i
G_o
$\frac{\partial J_t}{\partial h_{t-k}}$
$\frac{\partial J_t}{\partial h_{t-k+1}}$
$\frac{\partial J_t}{\partial h_t}$
MSE
J_t

FIGURE 5.7: BPTT in terms of cell state and state over LSTM cells. J_t, an error at time t, backpropagate through the inner architecture of LSTM cell which has two different paths between the neighboring cells. The partial derivative of J_t with respect of $\mathbf{C}_{t-k}$ or $\mathbf{h}_{t-k}$ represents the effect of the error on the cell that k steps behind.

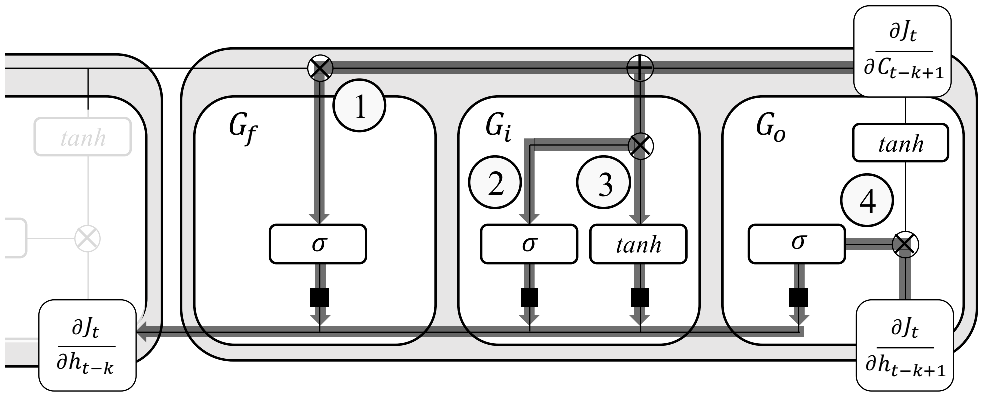
In the following, the discussion concentrates on the inner paths of LSTM cell, where the gradients backpropagate through time. As the gradients in terms of cell parameters depend on the partial derivatives of J_t with respect to state $\mathbf{C}_{t-k}$ or state $\mathbf{h}_{t-k}$, it is important to understand how the partial derivatives are computed. In unfolded graph, the gradients backpropagate cell by cell through two paths, cell state $\mathbf{C}_{t-k}$ and state $\mathbf{h}_{t-k}$ where $k = 1, \cdots t$, shown in Figure 5.7. The paths where the gradients backpropagate through the inner architecture of the LSTM cell between two neighboring time steps, $t-k$ and $t-k+1$ are discussed respectively.

· **Partial derivative of J_t with respect to $\mathbf{h}_t$ and $\mathbf{C}_t$**

$$\begin{aligned} \frac{\partial J_t}{\partial \mathbf{h}_t} &= \frac{\partial J_t}{\partial \mathbf{p}_t}\frac{\partial \mathbf{p}_t}{\partial \mathbf{h}_t} = 2(\mathbf{p}_t - \mathbf{y}_t)\cdot \mathbf{W}_p \\ \frac{\partial J_t}{\partial \mathbf{C}_t} &= \frac{\partial J_t}{\partial \mathbf{h}_t}\frac{\partial \mathbf{h}_t}{\partial \mathbf{C}_t} \\ &= 2\{(\mathbf{p}_t - \mathbf{y}_t)\cdot \mathbf{W}_p\} \odot \mathbf{o}_t \odot \{1 - tanh^2(\mathbf{C}_t)\} \end{aligned} \tag{5.13}$$

As shown in the right box of Figure 5.7, the partial derivatives of J_t with respect to $\mathbf{h}_t$ and $\mathbf{C}_t$ are simply computed as shown in Equation 5.13 because the error J_t doesn't backpropagate through time yet. Note that the partial derivative of J_t with respect to $\mathbf{C}_t$ is originated from the partial derivative with respect to $\mathbf{h}_t$.

· **Partial derivative of J_t with respect to $\mathbf{h}_{t-k}$**



$\frac{\partial J_t}{\partial C_{t-k+1}}$
tanh
G_f
1
G_i
2
3
G_o
tanh
4
σ
σ
tanh
σ
$\frac{\partial J_t}{\partial h_{t-k}}$
$\frac{\partial J_t}{\partial h_{t-k+1}}$

FIGURE 5.8: Four different paths that the gradients backpropagate to $\mathbf{h}_{t-k}$

Figure 5.8 depicts four different paths within the LSTM cell that the gradients back-propagate to $\mathbf{h}_{t-k}$. The paths ① passing by the forget gate, and ②, ③ both passing by the input gate, start from $\mathbf{C}_{t-k+1}$ while the path ④ passing by output gate, starts from $\mathbf{h}_{t-k+1}$. Accordingly, the partial derivative of J_t with respect to $\mathbf{h}_{t-k}$ can be expressed by the sum of the inflow from four paths (See Equation 5.14). Each path can be factorized by the chain rule (See Equation 5.15).

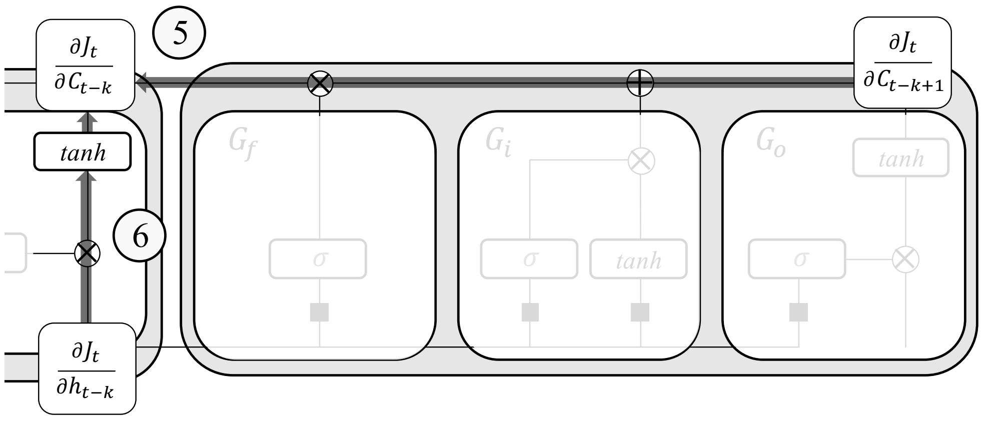
$$\begin{aligned}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} = & \frac{\partial J_t}{\partial \mathbf{C}_{t-k+1}} \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{\textcircled{1}} + \frac{\partial J_t}{\partial \mathbf{C}_{t-k+1}} \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{\textcircled{2}} \\ & + \frac{\partial J_t}{\partial \mathbf{C}_{t-k+1}} \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{\textcircled{3}} + \frac{\partial J_t}{\partial \mathbf{h}_{t-k+1}} \frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{\textcircled{4}}\end{aligned} \tag{5.14}$$

$$\begin{aligned}\frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{\textcircled{1}} &= \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{f}_{t-k+1}} \frac{\partial \mathbf{f}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \\ &= \mathbf{C}_{t-k} \odot \{\mathbf{f}_{t-k+1} \odot (\mathbf{1} - \mathbf{f}_{t-k+1}) \cdot \mathbf{W}_{hf}\} \\ \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{\textcircled{2}} &= \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{i}_{t-k+1}} \frac{\partial \mathbf{i}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \\ &= \tilde{\mathbf{C}}_{t-k+1} \odot \{\mathbf{i}_{t-k+1} \odot (\mathbf{1} - \mathbf{i}_{t-k+1}) \cdot \mathbf{W}_{hi}\} \\ \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{\textcircled{3}} &= \frac{\partial \mathbf{C}_{t-k+1}}{\partial \tilde{\mathbf{C}}_{t-k+1}} \frac{\partial \tilde{\mathbf{C}}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \\ &= \mathbf{i}_{t-k+1} \odot \{(\mathbf{1} - \tilde{\mathbf{C}}_{t-k+1}^2) \cdot \mathbf{W}_{hc}\} \\ \frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{\textcircled{4}} &= \frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{o}_{t-k+1}} \frac{\partial \mathbf{o}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \\ &= tanh(\mathbf{C}_{t-k+1}) \odot \{\mathbf{o}_{t-k+1} \odot (\mathbf{1} - \mathbf{o}_{t-k+1}) \cdot \mathbf{W}_{ho}\}\end{aligned} \tag{5.15}$$

Figure 5.8 and Equation 5.14 and 5.15 reveals that the partial derivative of J_t with respect to $\mathbf{h}_{t-k}$ includes the non-linear activation function so that gradients backpropagating through the paths have risk of vanishing or exploding like ERNN.

· Partial derivative of J_t with respect to $\mathbf{C}_{t-k}$

As shown in Figure 5.9, the gradients backpropagate to the cell state $\mathbf{C}_{t-k}$ through two paths, $\textcircled{5}$ from $\mathbf{C}_{t-k+1}$ and $\textcircled{6}$ from $\mathbf{h}_{t-k}$. The partial derivative of J_t with respect to $\mathbf{C}_{t-k}$ is the sum of gradients from two paths (See Equation 5.16), where the path $\textcircled{6}$



$\frac{\partial J_t}{\partial C_{t-k}}$
5
$\frac{\partial J_t}{\partial C_{t-k+1}}$
tanh
6
G_f
G_i
G_o
σ
σ
tanh
σ
tanh
$\frac{\partial J_t}{\partial h_{t-k}}$

FIGURE 5.9: Two paths that the gradients backpropagate to $\mathbf{C}_{t-k}$. Gradients that backpropagate through the path (5) don't get vanishing or exploding thanks to the lack of activation function.

stems from the partial derivative of $\mathbf{h}_{t-k}$ which is the sum of four path in the Figure 5.8.

$$\begin{aligned}
\frac{\partial J_t}{\partial \mathbf{C}_{t-k}} &= \frac{\partial J_t}{\partial \mathbf{C}_{t-k+1}} \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{C}_{t-k}} \textcircled{5} + \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{C}_{t-k}} \textcircled{6} \\
\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} &= \frac{\partial J_t}{\partial \mathbf{C}_{t-k+1}} \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \textcircled{1} + \frac{\partial J_t}{\partial \mathbf{C}_{t-k+1}} \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \textcircled{2} \\
&+ \frac{\partial J_t}{\partial \mathbf{C}_{t-k+1}} \frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \textcircled{3} + \frac{\partial J_t}{\partial \mathbf{h}_{t-k+1}} \frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \textcircled{4}
\end{aligned} \tag{5.16}$$

The path $\textcircled{5}$ has a special property. As it doesn't have non-linear activation function on the path unlike other the paths, the gradients that backpropagate through the path $\textcircled{5}$ don't suffer from the vanishing or exploding issue. The path $\textcircled{5}$ provides a long-term bypass that enables the RNNs with LSTM cell to learn long-term dependencies in the sequence.

On the other hand, the path $\textcircled{6}$ includes hyperbolic tangent which is a non-linear activation function. Partial derivative of $\mathbf{h}_{t-k}$ with respect to $\mathbf{C}_{t-k}$ returns $\mathbf{o}_{t-k} \odot \{1 - tanh^2(\mathbf{C}_{t-k})\}$, whose elements are equal to or less than one.

$$
\begin{aligned}
\overset{⑤}{\frac{\partial \mathbf{C}_{t-k+1}}{\partial \mathbf{C}_{t-k}}} &= \mathbf{f}_{t-k} \\
\overset{⑥}{\frac{\partial \mathbf{h}_{t-k}}{\partial \mathbf{C}_{t-k}}} &= \mathbf{o}_{t-k} \odot \{1 - tanh^2(\mathbf{C}_{t-k})\}
\end{aligned}
\tag{5.17}
$$

5.3 Gated recurrent unit (GRU)

In this section, gated recurrent unit (GRU) is suggested as an advanced RNN model from LSTM. Two notable differences compared with LSTM are found in Figure 5.10. At first, GRU has two gates, referred to reset and update gate respectively. It implies that GRU has fewer parameters to train, that allows to reduce computation load and time. The second, GRU doesn't have an independent memory cell, such as cell state $\mathbf{C}_t$ in LSTM. It suggests that the information is forward and backpropagates only through the state $\mathbf{h}_t$. Despite fewer gates and simpler architecture, it is known that GRU provides at least comparable prediction accuracy to LSTM and outperforms in terms of computation time.

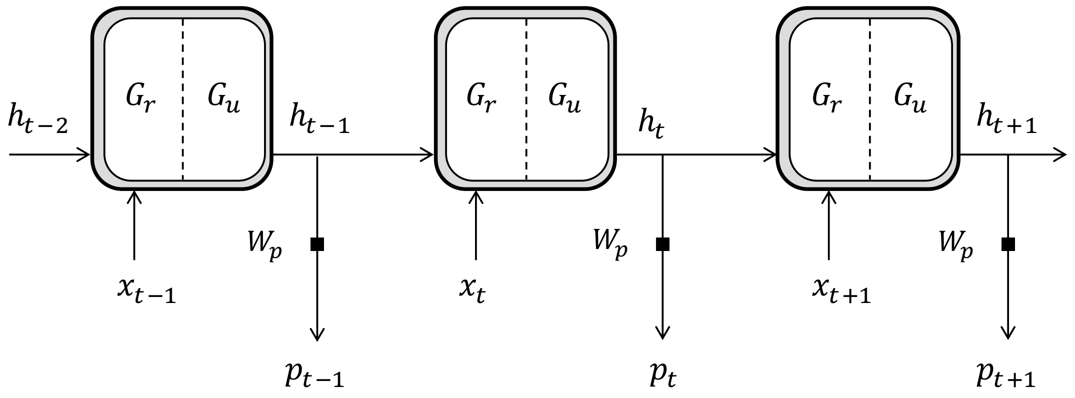


FIGURE 5.10: Unfolded graph of a RNN with GRU cell that consist of two gates.

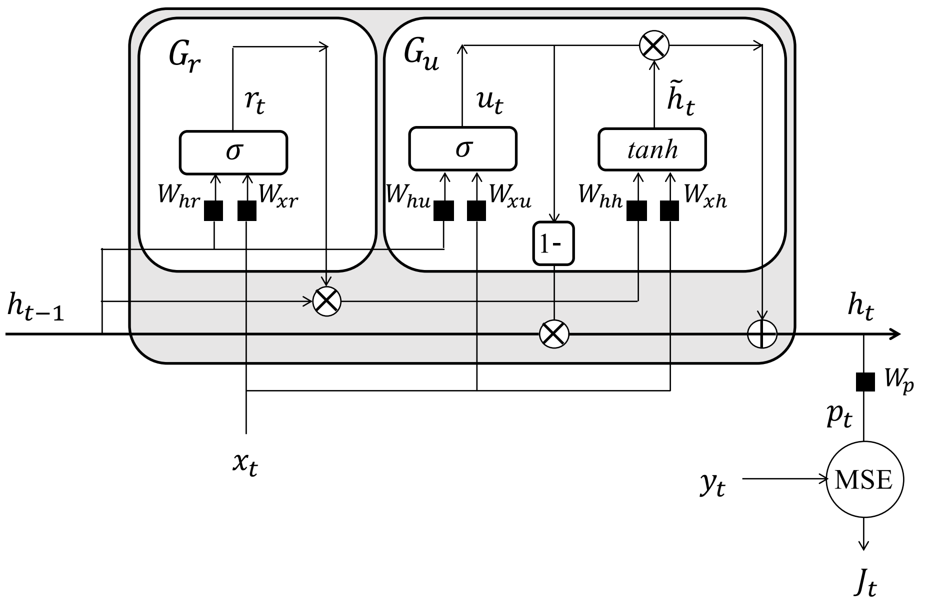


FIGURE 5.11: GRU cell architecture. Note that the bypass without non-linear activation function for state $\mathbf{h}_t$ enables to avoid vanishing or exploding gradient problem and backpropagate the gradients to the further past.

5.4 Training GRU

5.4.1 Forward propagation

Figure 5.11 depicts the architecture of GRU cell with two gate layers. Like LSTM, GRU architecture also has a bypass for state $\mathbf{h}_t$, which transfer information without experiencing activation function, such as sigmoid $\sigma(\cdot)$ or hyperbolic tangent. It enables the gradients backpropagate without vanishing or exploding so that the RNNs can capture long-term dependencies of the sequence which is not possible with ERNN.

5.4.1.1 Variables and trainable parameters

The following table shows all trainable parameters and variables of GRU cell with dimensional information. As prediction parameters works as same as they do for LSTM and ERNN, it is excluded from the following discussion.

Gate type		Variable	Parameters		
Reset gate	G_r	$\mathbf{r}_t \in \mathbf{R}^s$	$\mathbf{W}_{xr} \in \mathbf{R}^{s\times m}$	$\mathbf{W}_{hr} \in \mathbf{R}^{s\times s}$	$\mathbf{b}_r \in \mathbf{R}^s$
Update gate	G_u	$\mathbf{u}_t \in \mathbf{R}^s$	$\mathbf{W}_{xu} \in \mathbf{R}^{s\times m}$	$\mathbf{W}_{hu} \in \mathbf{R}^{s\times s}$	$\mathbf{b}_u \in \mathbf{R}^s$
		$\tilde{\mathbf{h}}_t \in \mathbf{R}^s$	$\mathbf{W}_{xh} \in \mathbf{R}^{s\times m}$	$\mathbf{W}_{hh} \in \mathbf{R}^{s\times s}$	$\mathbf{b}_h \in \mathbf{R}^s$
Prediction		$\mathbf{p}_t \in \mathbf{R}^m$	$\mathbf{W}_p \in \mathbf{R}^{m\times s}$		$\mathbf{b}_p \in \mathbf{R}^m$

TABLE 5.2: Variables and trainable parameters of GRU cell

5.4.1.2 State $\mathbf{h}_t$

$$\mathbf{h}_t = (\mathbf{1} - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t \tag{5.18}$$

State $\mathbf{h}_t$ conveys the information processed so far to the next cell by a linear combination shown in Equation 5.18. Due to the linear combination, GRU is free from the vanishing or exploding gradient problem when the gradients backpropagate through time.

The variable $\mathbf{u}_t$ participates in determining the portion of the information inherited from the previous state $\mathbf{h}_{t-1}$ also generated from the current candidate $\tilde{\mathbf{h}}_t$ like a bidirectional switch. For one extreme case, the linear combination in Equation 5.18 turns to a bypass through all cells when $\mathbf{u}_{t-k} = 0$, where $k = 0, 1, \cdots t$. The bypass can be interpreted as a memory which makes all state $\mathbf{h}_{t-k}$ keep the same information.

5.4.1.3 Update Gate Variable $\mathbf{u}_t$ and Candidate $\tilde{\mathbf{h}}_t$

$$\begin{aligned} \mathbf{u}_t &= \sigma(\mathbf{W}_{hu} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xu} \cdot \mathbf{x}_t + \mathbf{b}_u) \\ \tilde{\mathbf{h}}_t &= tanh\{\mathbf{W}_{hh} \cdot (\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h\} \end{aligned} \tag{5.19}$$

Update gate generates a variable $\mathbf{u}_t$ and the candidate $\tilde{\mathbf{h}}_t$. Mathematical expression of $\mathbf{u}_t$, shown in Equation 5.19, is identical to the expression of the variables of LSTM, $\mathbf{f}_t$, $\mathbf{i}_t$ and $\mathbf{o}_t$. However, the role of the variable $\mathbf{u}_t$ is slightly different.

In LSTM, two variables $\mathbf{f}_t$ and $\mathbf{i}_t$ respectively determines the portion of the information, to inherit or to update. In GRU, one variable $\mathbf{u}_t$ determines the portion in two ways. $(\mathbf{1} - \mathbf{u}_t)$ takes charge of the role of $\mathbf{f}_t$ while $\mathbf{u}_t$ takes $\mathbf{i}_t$'s role. Thus, update gate is interpreted as a coupled gate that forget and input gate in LSTM are combined.

Candidate $\tilde{\mathbf{h}}_t$ has a unique feature. Reset gate variable $\mathbf{r}_t$ determines the portion of the previous information $\mathbf{h}_{t-1}$ to include in generating the new candidate by element-wise multiplication, as shown in Equation 5.19. For an extreme case, the candidate $\tilde{\mathbf{h}}_t$ is generated by only the input data $\mathbf{x}_t$ when $\mathbf{r(t)} = \mathbf{0}$.

5.4.1.4 Reset Gate Variable $\mathbf{r}_t$

$$\mathbf{r}_t = \sigma(\mathbf{W}_{hr} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xr} \cdot \mathbf{x}_t + \mathbf{b}_r) \tag{5.20}$$

Reset gate variable $\mathbf{r}_t$ participates in computing new candidate $\tilde{\mathbf{h}}_t$. Compared with output gate variable $\mathbf{o}_t$ in LSTM, $\mathbf{o}_t$ filters the information in $\mathbf{C}_t$ and helps in generating state $\mathbf{h}_t$ which is fed into the next cell. Whereas reset gate variable $\mathbf{r}_t$ filters the information carried by the state from previous cell $\mathbf{h}_{t-1}$ only for the candidate $\tilde{\mathbf{h}}_t$.

5.4.2 Back Propagation

Like LSTM and ERNN, parameters are discriminated by two groups, prediction parameters and cell parameters. The gradients in terms of prediction parameters are computed in the same way as the gradients in LSTM and ERNN are.

Cell parameters are divided into three branches depending on the backpropagating paths, such as parameters of the candidate $\tilde{\mathbf{h}}_t$, parameters of update variable $\mathbf{u}_t$ and parameters of reset variable $\mathbf{r}_t$.

Lastly, discussion for the gradient in terms of the state $\mathbf{h}_{t-k}$ is followed. As the gradients backpropagate through state $\mathbf{h}_{t-k}, k = 0, 1, \cdots t$ over cells, the cell parameters in each cell also experience error from the state in the same cell. To explore it, gradients in terms of the state at two time steps, t and $t-1$, are compared. From the insight found by the comparison, generalization of the gradients is discussed at the end.

5.4.2.1 Gradients of J in terms of Parameters regarding Candidate $\tilde{\mathbf{h}}_{t-k}$

The candidate parameters $\{\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{b}_h\}$ shown in Figure 5.12, receive the gradients through the state $\mathbf{h}_{t-k}$ and $\tilde{\mathbf{h}}_{t-k}$. The chain rule provides a formal expression of the gradients of J in terms of the parameters, shown in Equation 5.21.

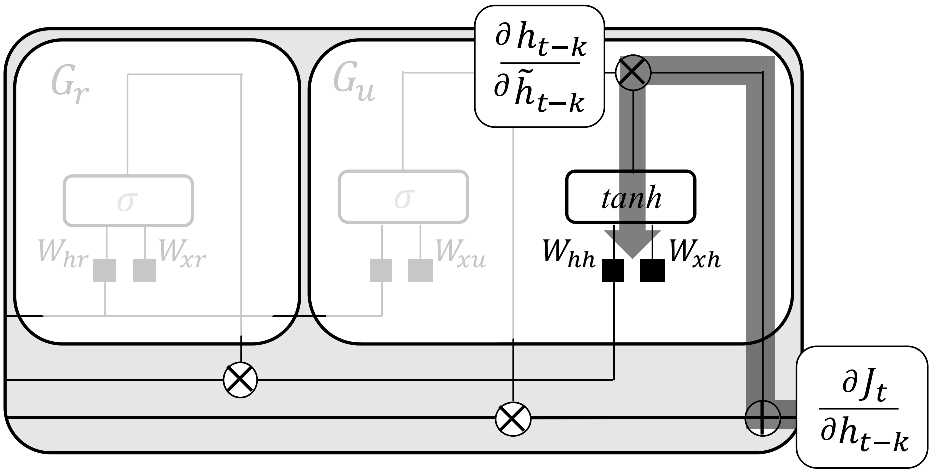
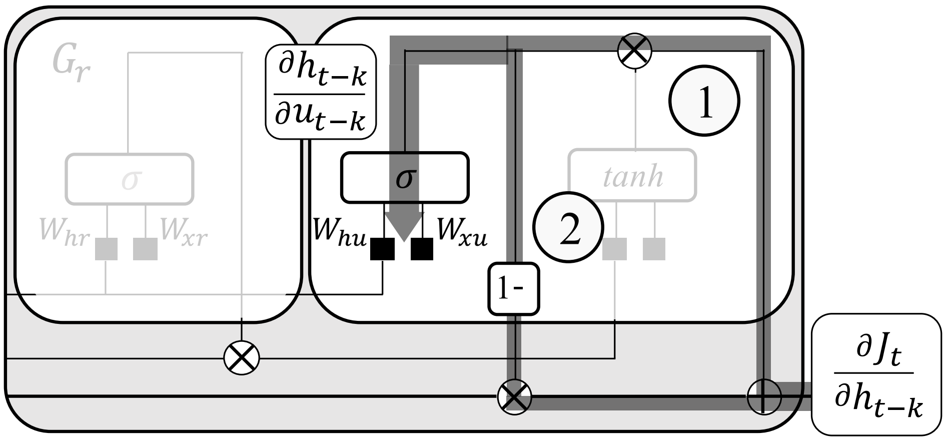

G_r
σ
W_{hr}
W_{xr}
G_u
σ
W_{xu}
$\frac{\partial h_{t-k}}{\partial \tilde{h}_{t-k}}$
tanh
W_{hh}
W_{xh}
$\frac{\partial J_t}{\partial h_{t-k}}$

FIGURE 5.12: Gradients in terms of candidate parameters $\{\mathbf{W}_{xh}, \mathbf{W}_{hh}, \mathbf{b}_h\}$ can be expressed by the chain rule that is derived from the partial derivative of an error J_t with respect to $\mathbf{h}_{t-k}$ and $\mathbf{h}_{t-k}$ with respect to $\tilde{\mathbf{h}}_{t-k}$.

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{W}_{xh}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \frac{\partial \mathbf{h}_{t-k}}{\partial \tilde{\mathbf{h}}_{t-k}} \frac{\partial \tilde{\mathbf{h}}_{t-k}}{\partial \mathbf{W}_{xh}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot \mathbf{u}_{t-k} \odot (\mathbf{1} - \tilde{\mathbf{h}}_{t-k}^2) \otimes \mathbf{x}_{t-k} \\
\frac{\partial J}{\partial \mathbf{W}_{hh}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \frac{\partial \mathbf{h}_{t-k}}{\partial \tilde{\mathbf{h}}_{t-k}} \frac{\partial \tilde{\mathbf{h}}_{t-k}}{\partial \mathbf{W}_{hh}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot \mathbf{u}_{t-k} \odot (\mathbf{1} - \tilde{\mathbf{h}}_{t-k}^2) \otimes \mathbf{h}_{t-k-1} \\
\frac{\partial J}{\partial \mathbf{b}_{h}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \frac{\partial \mathbf{h}_{t-k}}{\partial \tilde{\mathbf{h}}_{t-k}} \frac{\partial \tilde{\mathbf{h}}_{t-k}}{\partial \mathbf{b}_{h}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot \mathbf{u}_{t-k} \odot (\mathbf{1} - \tilde{\mathbf{h}}_{t-k}^2)
\end{aligned} \tag{5.21}$$

5.4.2.2 Gradients of J in terms of Parameters regarding Update Variable $\mathbf{u}_{t-k}$

The parameters of update gate variable $\mathbf{u}_t$, $\{\mathbf{W}_{xu}, \mathbf{W}_{hu}, \mathbf{b}_u\}$, experience the error from two paths as shown in Figure 5.13. Thus, the gradients in terms of the parameters are the sum of the gradients backpropagating through two paths. For the path $\textcircled{1}$, partial derivative of $\mathbf{h}_{t-k}$ in terms of $\mathbf{u}_{t-k}$ is simply acquired as $\tilde{\mathbf{h}}_{t-k}$ by the definition in Equation 5.18. In the same context, the partial derivative by the path $\textcircled{2}$ is $-\mathbf{h}_{t-k-1}$.



G_r
σ
W_{hr}
W_{xr}
$\frac{\partial h_{t-k}}{\partial u_{t-k}}$
σ
W_{hu}
W_{xu}
1-
1
2
tanh
$\frac{\partial J_t}{\partial h_{t-k}}$

FIGURE 5.13: Gradients in terms of update parameters $\{\mathbf{W}_{xu}, \mathbf{W}_{hu}, \mathbf{b}_u\}$ can be expressed by the chain rule that is derived from the partial derivative of an error J_t with respect to $\mathbf{h}_{t-k}$ and $\mathbf{h}_{t-k}$ with respect to $\mathbf{u}_{t-k}$.

Gradient in terms of the parameter $\mathbf{W}_{xu}$ backpropagating through each path is shown in Equation 5.22.

$$\begin{aligned}
\frac{\partial J_t}{\partial \mathbf{W}_{xu}} &= \frac{\partial J_t}{\partial \mathbf{W}_{xu}}^{\textcircled{1}} + \frac{\partial J_t}{\partial \mathbf{W}_{xu}}^{\textcircled{2}} \\
\frac{\partial J_t}{\partial \mathbf{W}_{xu}}^{\textcircled{1}} &= \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot \tilde{\mathbf{h}}_{t-k} \odot \mathbf{u}_{t-k} \odot (\mathbf{1} - \mathbf{u}_{t-k}) \otimes \mathbf{x}_{t-k} \\
\frac{\partial J_t}{\partial \mathbf{W}_{xu}}^{\textcircled{2}} &= -\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot \mathbf{h}_{t-k-1} \odot \mathbf{u}_{t-k} \odot (\mathbf{1} - \mathbf{u}_{t-k}) \otimes \mathbf{x}_{t-k}
\end{aligned} \tag{5.22}$$

For a sequence with length T, overall gradients in terms of update gate parameters $\{\mathbf{W}_{xu}, \mathbf{W}_{hu}, \mathbf{b}_u\}$ are denoted in Equation 5.23.

$$\begin{aligned}
\frac{\partial J}{\partial \mathbf{W}_{xu}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot (\tilde{\mathbf{h}}_{t-k} - \mathbf{h}_{t-k-1}) \odot \mathbf{u}_{t-k} \odot (\mathbf{1} - \mathbf{u}_{t-k}) \otimes \mathbf{x}_{t-k} \\
\frac{\partial J}{\partial \mathbf{W}_{hu}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot (\tilde{\mathbf{h}}_{t-k} - \mathbf{h}_{t-k-1}) \odot \mathbf{u}_{t-k} \odot (\mathbf{1} - \mathbf{u}_{t-k}) \otimes \mathbf{h}_{t-k-1} \\
\frac{\partial J}{\partial \mathbf{b}_{u}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t} \frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot (\tilde{\mathbf{h}}_{t-k} - \mathbf{h}_{t-k-1}) \odot \mathbf{u}_{t-k} \odot (\mathbf{1} - \mathbf{u}_{t-k})
\end{aligned} \tag{5.23}$$

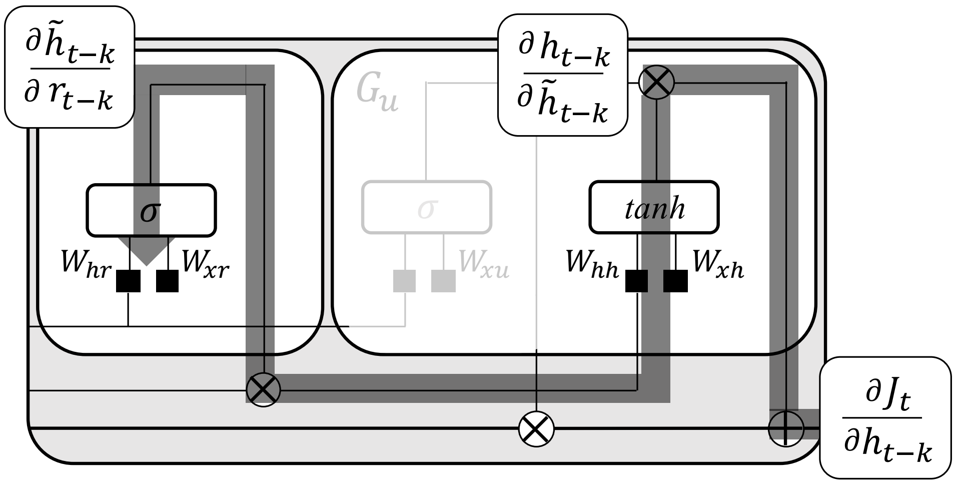

$\frac{\partial \tilde{h}_{t-k}}{\partial r_{t-k}}$
$\frac{\partial h_{t-k}}{\partial \tilde{h}_{t-k}}$
G_u
σ
W_{hr}
W_{xr}
σ
W_{xu}
$tanh$
W_{hh}
W_{xh}
$\frac{\partial J_t}{\partial h_{t-k}}$

FIGURE 5.14: Gradients in terms of reset gate parameters $\{\mathbf{W}_{xr}, \mathbf{W}_{hr}, \mathbf{b}_r\}$ can be expressed by the chain rule that is derived from the partial derivative of an state $\mathbf{h}_{t-k}$ with respect to candidate $\tilde{\mathbf{h}}_{t-k}$ and $\tilde{\mathbf{h}}_{t-k}$ with respect to $\mathbf{r}_{t-k}$.

5.4.2.3 Gradients of J in terms of Parameters regarding Reset Variable $\mathbf{r}_{t-k}$

The path that gradients backpropagate to reset gate parameters $\{\mathbf{W}_{xr}, \mathbf{W}_{hr}, \mathbf{b}_r\}$ is shown in Figure 5.14. The gradients introduced through the state $\mathbf{h}_{t-k}$, propagates back to reset gate variable $\mathbf{r}_{t-k}$ through candidate $\tilde{\mathbf{h}}_{t-k}$. Thus, the gradient in terms of reset gate parameters can be expressed by the chain rule. Equation 5.24 shows the gradients for a sequence with length T, where the partial derivative of $\tilde{\mathbf{h}}_{t-k}$ with respect to $\mathbf{r}_{t-k}$ is given in Equation 5.25 .

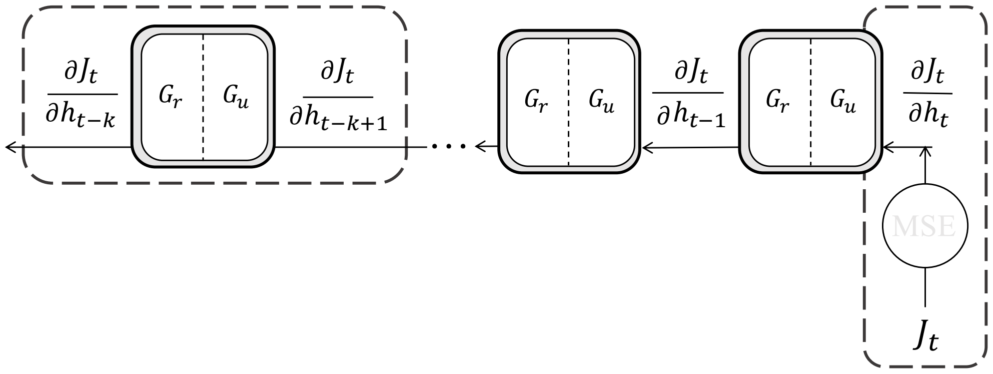
$$
\begin{aligned}
\frac{\partial J}{\partial \mathbf{W}_{xr}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}}\frac{\partial \mathbf{h}_{t-k}}{\partial \tilde{\mathbf{h}}_{t-k}}\frac{\partial \tilde{\mathbf{h}}_{t-k}}{\partial \mathbf{r}_{t-k}}\frac{\partial \mathbf{r}_{t-k}}{\partial \mathbf{W}_{xr}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot \mathbf{u}_{t-k} \odot \{(\mathbf{1}-\tilde{\mathbf{h}}^2_{t-k})\cdot \mathbf{W}_{hh}\} \odot \mathbf{h}_{t-k-1} \\
&\qquad \odot \mathbf{r}_{t-k} \odot (\mathbf{1}-\mathbf{r}_{t-k}) \otimes \mathbf{x}_{t-k} \\
\frac{\partial J}{\partial \mathbf{W}_{hr}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}}\frac{\partial \mathbf{h}_{t-k}}{\partial \tilde{\mathbf{h}}_{t-k}}\frac{\partial \tilde{\mathbf{h}}_{t-k}}{\partial \mathbf{r}_{t-k}}\frac{\partial \mathbf{r}_{t-k}}{\partial \mathbf{W}_{hr}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot \mathbf{u}_{t-k} \odot \{(\mathbf{1}-\tilde{\mathbf{h}}^2_{t-k})\cdot \mathbf{W}_{hh}\} \odot \mathbf{h}_{t-k-1} \\
&\qquad \odot \mathbf{r}_{t-k} \odot (\mathbf{1}-\mathbf{r}_{t-k}) \otimes \mathbf{h}_{t-k-1} \\
\frac{\partial J}{\partial \mathbf{b}_{r}} &= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}}\frac{\partial \mathbf{h}_{t-k}}{\partial \tilde{\mathbf{h}}_{t-k}}\frac{\partial \tilde{\mathbf{h}}_{t-k}}{\partial \mathbf{r}_{t-k}}\frac{\partial \mathbf{r}_{t-k}}{\partial \mathbf{b}_{r}} \\
&= \frac{1}{T}\sum_{t=0}^{T-1}\sum_{k=0}^{t}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} \odot \mathbf{u}_{t-k} \odot \{(\mathbf{1}-\tilde{\mathbf{h}}^2_{t-k})\cdot \mathbf{W}_{hh}\} \odot \mathbf{h}_{t-k-1} \\
&\qquad \odot \mathbf{r}_{t-k} \odot (\mathbf{1}-\mathbf{r}_{t-k})
\end{aligned}
\tag{5.24}
$$

$$
\begin{aligned}
&\frac{\partial \tilde{\mathbf{h}}_{t-k}}{\partial \mathbf{r}_{t-k}} = \{(\mathbf{1}-\tilde{\mathbf{h}}^2_{t-k})\cdot \mathbf{W}_{hh}\} \odot \mathbf{h}_{t-k-1} \\
&\Longleftrightarrow \quad \tilde{\mathbf{h}}_t = tanh\{\mathbf{W}_{hh}\cdot(\mathbf{r}_t \odot \mathbf{h}_{t-1}) + \mathbf{W}_{xh}\cdot \mathbf{x}_t + \mathbf{b}_h\}
\end{aligned}
\tag{5.25}
$$

5.4.2.4 Backpropagating Gradients between Neighboring Time Steps

In the following, the discussion concentrates on the paths within the GRU cell, where the gradients backpropagate through the inner architecture of the cell between two neighboring time steps, $t-k$ and $t-k+1$. Four different paths, path **A** in Figure 5.16, path **B** in Figure 5.17. path **C** in Figure 5.18 and path **D** in Figure 5.19, are suggested for the backpropagation of GRU. Among the paths, path **A** provide a bypass without non-linear activation so that the gradients can backpropagate without vanishing or exploding problem caused by the derivative of the activation function. Details will be discussed later.

The formal expressions of gradients for all gate parameters in Equation 5.21, 5.23 and 5.24, include the partial derivative of J_t with respect to $\mathbf{h}_{t-k}$, which represents how the error at a time step t influence to the state $\mathbf{h}_{t-k}$ at k time step back from t, shown in Figure 5.15. That can be specified by the sum of the backpropagated gradients from four paths, as shown in Equation 5.26.

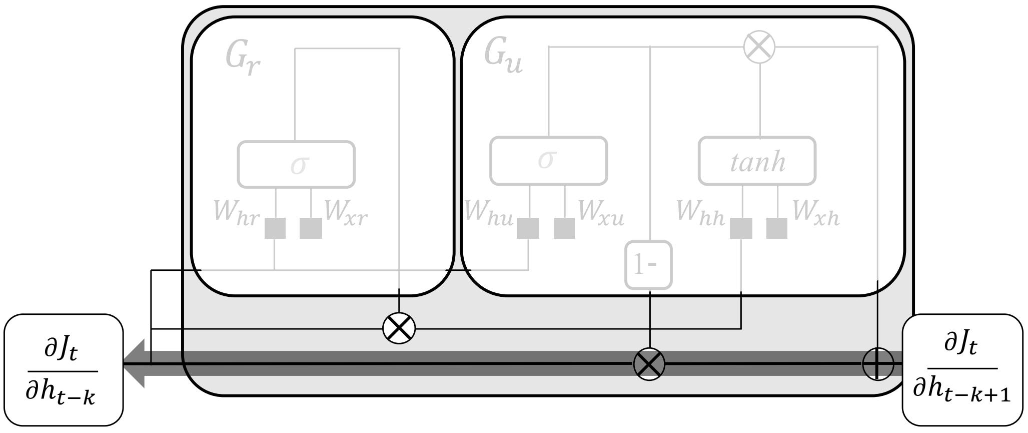


$\frac{\partial J_t}{\partial h_{t-k}}$
G_r
G_u
$\frac{\partial J_t}{\partial h_{t-k+1}}$
$\frac{\partial J_t}{\partial h_{t-1}}$
$\frac{\partial J_t}{\partial h_t}$
MSE
J_t

FIGURE 5.15: BPTT in terms of state over GRU cells. J_t, an error at time t, back-propagate through the inner architecture of GRU cell between the neighboring cells. The partial derivative of J_t with respect of $\mathbf{h}_{t-k}$ represents the effect of the error on the cell that k steps behind.

$$\begin{aligned}\frac{\partial J_t}{\partial \mathbf{h}_{t-k}} &= \frac{\partial J_t}{\partial \mathbf{h}_{t-k+1}}\frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{(\mathbf{A})} + \frac{\partial J_t}{\partial \mathbf{h}_{t-k+1}}\frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{(\mathbf{B})} \\ &+ \frac{\partial J_t}{\partial \mathbf{h}_{t-k+1}}\frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{(\mathbf{C})} + \frac{\partial J_t}{\partial \mathbf{h}_{t-k+1}}\frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{(\mathbf{D})}\end{aligned} \tag{5.26}$$

· **Path A : Bypass without Activation Function**



G_r
G_u
σ
W_{hr}
W_{xr}
W_{hu}
W_{xu}
tanh
W_{hh}
W_{xh}
1-
$\frac{\partial J_t}{\partial h_{t-k}}$
$\frac{\partial J_t}{\partial h_{t-k+1}}$

FIGURE 5.16: Path that gradients backpropagating to $\mathbf{h}_{t-k}$ through a bypass.

$$\begin{aligned} \frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{(\mathbf{A})} &= \mathbf{1} - \mathbf{u}_{t-k+1} \\ \Longleftrightarrow \quad \mathbf{h}_t &= (\mathbf{1} - \mathbf{u}_t) \odot \mathbf{h}_{t-1} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t \end{aligned} \tag{5.27}$$

The path **A**, shown in Figure 5.16, doesn't hold any non-linear activation function in the path. Thus, the partial derivative of $\mathbf{h}_{t-k+1}$ with respect to $\mathbf{h}_{t-k}$ in this case doesn't have multiplicative term, as denoted in Equation 5.27. The term $\mathbf{1} - \mathbf{u}_{t-k+1}$ allows that the partial derivative of J_t with respect to $\mathbf{h}_{t-k}$ can be same as partial derivative of J_t with respect to $\mathbf{h}_{t-k+1}$ if $\mathbf{u}_{t-k+1}$ is equal to zero. Therefore, the path plays an important role in avoiding the vanishing or exploding gradient problem.

· **Path B : Backpropagation through Candidate $\tilde{\mathbf{h}}_{t-k+1}$**

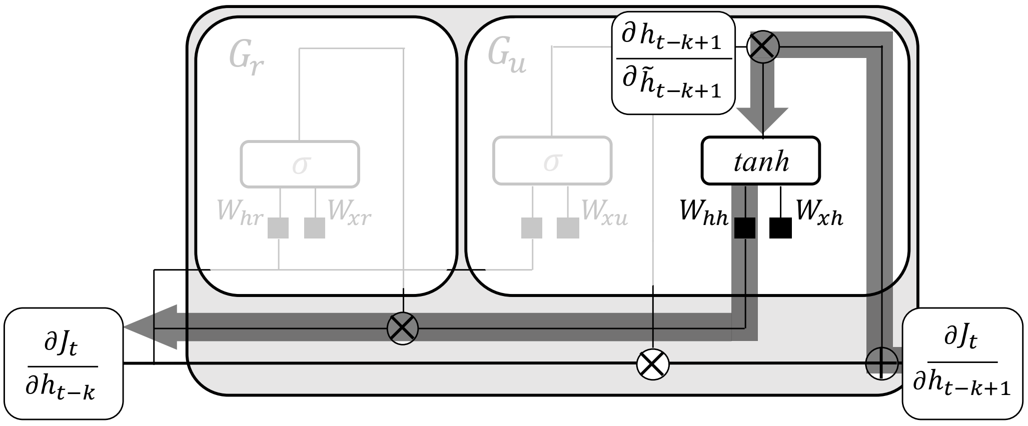
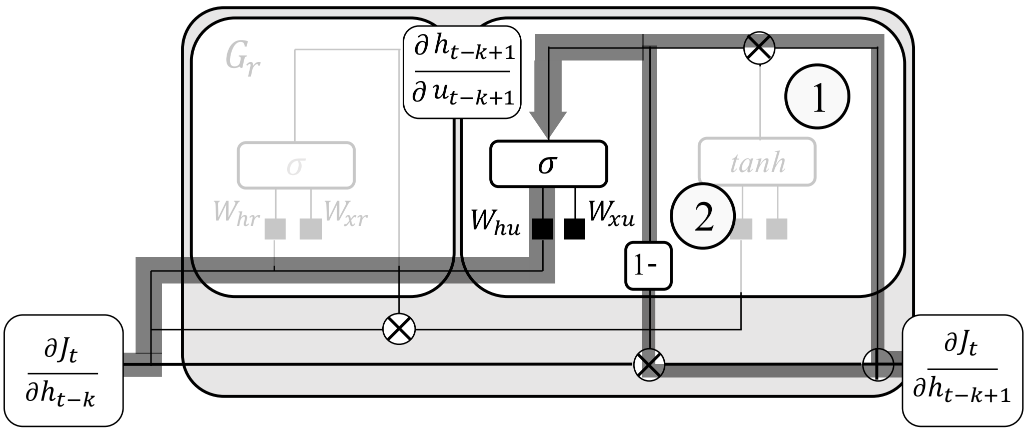

G_r
G_u
$\frac{\partial h_{t-k+1}}{\partial \tilde{h}_{t-k+1}}$
σ
σ
$tanh$
W_{hr}
W_{xr}
W_{xu}
W_{hh}
W_{xh}
$\frac{\partial J_t}{\partial h_{t-k}}$
$\frac{\partial J_t}{\partial h_{t-k+1}}$

FIGURE 5.17: Path that the error backpropagating through the candidate $\tilde{\mathbf{h}}_{t-k+1}$

$$\begin{aligned} \frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{(\mathbf{B})} &= \frac{\partial \mathbf{h}_{t-k+1}}{\partial \tilde{\mathbf{h}}_{t-k+1}} \frac{\partial \tilde{\mathbf{h}}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \\ &= \mathbf{u}_{t-k+1} \odot \{(\mathbf{1} - \tilde{\mathbf{h}}^2_{t-k+1}) \cdot \mathbf{W}_{hh}\} \odot \mathbf{r}_{t-k+1} \end{aligned} \tag{5.28}$$

Path **B**, shown in Figure 5.17, includes one non-linear activation function which is used for generating the candidate $\tilde{\mathbf{h}}_{t-k+1}$ in the forward propagation scheme. Note that the partial derivative of $\mathbf{h}_{t-k+1}$ with respect to the candidate $\tilde{\mathbf{h}}_{t-k+1}$ in Equation 5.28 returns the variable $\mathbf{u}_{t-k+1}$, which contributes to avoid the vanishing or exploding gradient problem in path **A**.

· **Path C : Backpropagation through Update Variable $\mathbf{u}_{t-k+1}$**



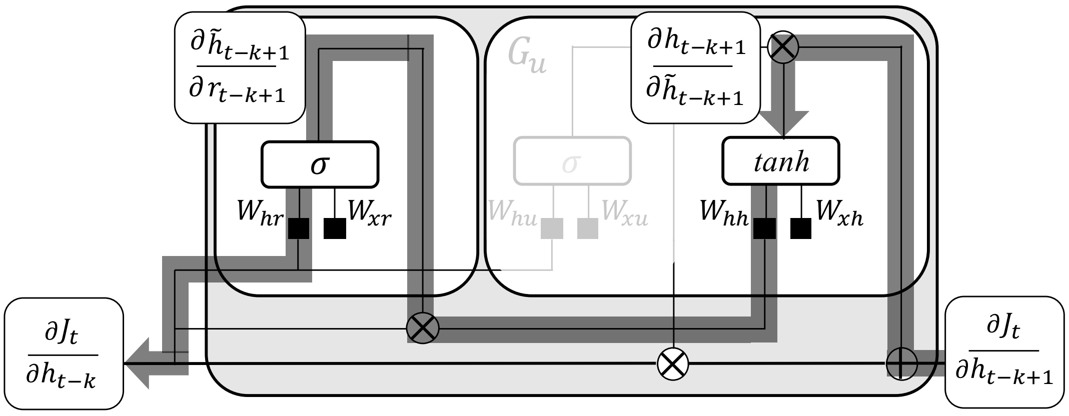
G_r
$\frac{\partial h_{t-k+1}}{\partial u_{t-k+1}}$
1
σ
σ
tanh
W_{hr}
W_{xr}
W_{hu}
W_{xu}
2
1-
$\frac{\partial J_t}{\partial h_{t-k}}$
$\frac{\partial J_t}{\partial h_{t-k+1}}$

FIGURE 5.18: Path that the error backpropagating through update variable $\mathbf{u}_{t-k+1}$.

$$
\begin{aligned}
\frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{(\mathbf{C})} &= \frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{u}_{t-k+1}} \frac{\partial \mathbf{u}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \\
&= (\tilde{\mathbf{h}}_{t-k+1} - \mathbf{h}_{t-k}) \odot \mathbf{u}_{t-k+1} \odot (\mathbf{1} - \mathbf{u}_{t-k+1}) \cdot \mathbf{W}_{hu}
\end{aligned} \tag{5.29}
$$

In path **C**, shown in Figure 5.18, the gradients backpropagate through the variable $\mathbf{u}_{t-k+1}$ before arriving the state $\mathbf{h}_{t-k}$. A non-linear sigmoid function is observed in the path. Due to the nature of sigmoid, its partial derivative also returns sigmoid itself. Equation 5.29 shows the partial derivative of $\mathbf{h}_{t-k+1}$ with respect to $\mathbf{h}_{t-k}$ for path **C**.

· **Path D : Backpropagation through Reset Variable $\mathbf{r}_{t-k+1}$**



$\frac{\partial \tilde{h}_{t-k+1}}{\partial r_{t-k+1}}$
G_u
$\frac{\partial h_{t-k+1}}{\partial \tilde{h}_{t-k+1}}$
σ
σ
tanh
W_{hr}
W_{xr}
W_{hu}
W_{xu}
W_{hh}
W_{xh}
$\frac{\partial J_t}{\partial h_{t-k}}$
$\frac{\partial J_t}{\partial h_{t-k+1}}$

FIGURE 5.19: Path that the error backpropagating through reset variable $\mathbf{r}_{t-k+1}$.

As reset gate variable $\mathbf{r}_{t-k+1}$ participates in computing the candidate $\tilde{\mathbf{h}}_{t-k+1}$, the error backpropagating to $\mathbf{r}_{t-K+1}$ must pass through $\tilde{\mathbf{h}}_{t-k+1}$. Thus, path $\mathbf{D}$, shown in Figure 5.19, includes two non-linear activation functions, a hyperbolic tangent in $\tilde{\mathbf{h}}_{t-k+1}$ and a sigmoid in $\mathbf{r}_{t-k+1}$. The derivative of $\tilde{\mathbf{h}}_{t-k+1}$ with respect to $\tilde{\mathbf{h}}_{t-k}$ for path $\mathbf{D}$ is specified in Equation 5.30.It includes the term $\mathbf{u}_{t-k+1}$ like other gradients above.

$$
\begin{aligned}
\frac{\partial \mathbf{h}_{t-k+1}}{\partial \mathbf{h}_{t-k}}^{(\mathbf{D})} &= \frac{\partial \mathbf{h}_{t-k+1}}{\partial \tilde{\mathbf{h}}_{t-k+1}} \frac{\partial \tilde{\mathbf{h}}_{t-k-+1}}{\partial \mathbf{r}_{t-k+1}} \frac{\partial \mathbf{r}_{t-k+1}}{\partial \mathbf{h}_{t-k}} \\
&= \mathbf{u}_{t-k+1} \odot \{(\mathbf{1} - \tilde{\mathbf{h}}^2_{t-k+1}) \cdot \mathbf{W}_{hh}\} \odot \mathbf{h}_{t-k} \odot \mathbf{r}_{t-k+1} \odot (\mathbf{1} - \mathbf{r}_{t-k+1}) \cdot \mathbf{W}_{hr}
\end{aligned}
\tag{5.30}
$$

Chapter 6

Deep RNN Architectures

6.1 Formal Description of Deep RNNs

Deep learning is built around a hypothesis that a deep, hierarchical model can be exponentially more efficient at representing some functions than a shallow one. In general, depth of a neural network's architecture refers to the number of levels of composition of non-linear operations in the function learned [96]. However, the depth of an RNN is more difficult to define unlike in the case of feedforward neural networks, because RNNs have an additional depth in time resulting from the composition of multiple nonlinear layers when unfolded in time [76].

Pascanu et al. [76] suggest a formal definition of deep RNN which can be obtained by adding extra layers in specific locations. Three options are proposed to obtaion a deep RNN : (a) deep transition RNN (b) deep output RNN (c) stacked RNN. Figure 6.1 shows each architecture deep RNN.

Deep transition RNN, shown in Figure 6.1(a), is characterized by the deep part located between the input and hidden states. It can extract more non-temporal structure from the input. The architecture is known to give a better disentanglement of the underlying factors of variation than the original input[97]. Equation 6.1 specifies the formal description of deep transition RNN , where f_l and $\{\mathbf{W}_{xl}, \mathbf{W}_{hl}, \mathbf{W}_{sl}, \mathbf{b}_l\}$ are element-wise nonlinear function and the weight/bias parameters for l^{th} layer of intermediate layers, $l = 1, 2, \cdots L$.

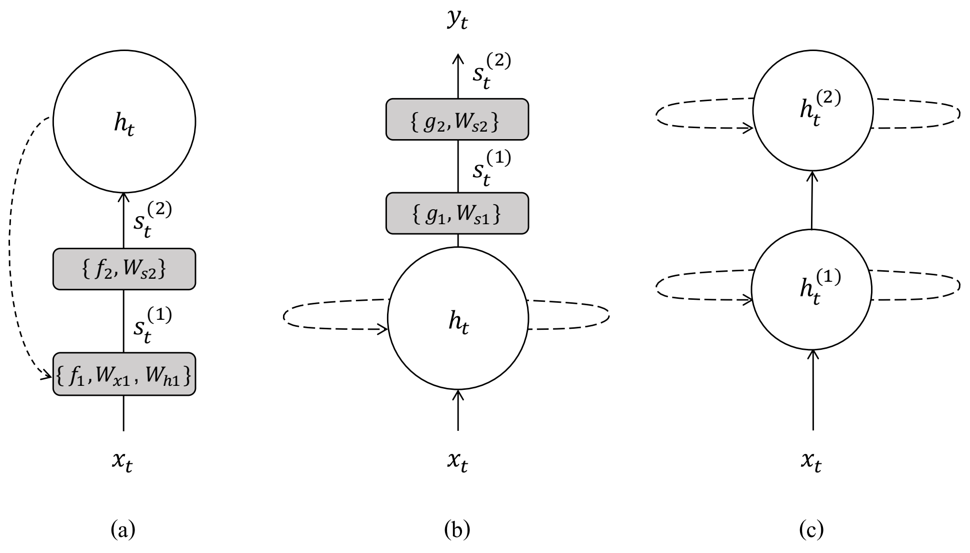


FIGURE 6.1: Different architectures of deep RNN : (a) deep transition RNN , (b) deep output RNN, (c) stacked RNN

$$\begin{aligned}
\mathbf{s}_t^{(l)} &= \cdot f_l(\mathbf{W}_{hl} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xl} \cdot \mathbf{x}_t + \mathbf{b}_l), \qquad && l = 1 \\
\mathbf{s}_t^{(l)} &= f_l(\mathbf{W}_{sl} \cdot \mathbf{s}_t^{(l-1)} + \mathbf{b}_l), && l > 1 \\
\mathbf{h}_t &= \mathbf{s}_t^{(L)}
\end{aligned} \tag{6.1}$$

In the same context, deep output RNN, shown in Figure 6.1(b) can be useful to disentangle the factors of variations in the hidden state, by implementing more non-linearity to compute the output. This allows the hidden state of the model to be more compact and may result in the model being able to summarize the history of previous inputs more efficiently [76]. Equation 6.2 specifies the formal description of deep output RNN, where g_l and $\{\mathbf{W}_{yl}, \mathbf{b}_l\}$ are element-wise nonlinear function and the weight/bias parameters for l^{th} layer of intermediate layers, $l = 1, 2, \cdots L$.

$$\begin{aligned}
\mathbf{h}_t &= g(\mathbf{W}_{hh} \cdot \mathbf{h}_{t-1} + \mathbf{W}_{xh} \cdot \mathbf{x}_t + \mathbf{b}_h), \quad && \textit{out of deep part} \\
\mathbf{s}_t^{(l)} &= g_l(\mathbf{W}_{sl} \cdot \mathbf{h}_t + \mathbf{b}_l), && l = 1 \\
\mathbf{s}_t^{(l)} &= g_l(\mathbf{W}_{sl} \cdot \mathbf{s}_t^{(l-1)} + \mathbf{b}_l), && l > 1 \\
\mathbf{y}_t &= \mathbf{s}_t^{(L)}
\end{aligned} \tag{6.2}$$

Stacked RNN, shown in Figure 6.1(c) is defined by stacking multiple recurrent hidden layers on top of each other [98–100]. New state at layer l $\mathbf{h}_t^{(l)}$, $l = 1, 2 \cdots L$ is defined in Equation 6.3 where f_l is the nonlinear function. Similar to Equation 3.2, $\mathbf{W}_{hl}$ represents weight parameter for the hidden state from the previous time step at the same layer and $\mathbf{W}_{xl}$ represents weight parameter for the input state from the same time step but from previous layer. When $l = 1$, state $\mathbf{h}_t^{(l)}$ is computed by the input $\mathbf{x}_t$, instead of $\mathbf{h}_t^{(l-1)}$. Each recurrent level of the stacked RNN captures the information in a different time scale of the input [101].

$$\mathbf{h}_t^{(l)} = f_l(\mathbf{W}_{hl} \cdot \mathbf{h}_{t-1}^{(l)} + \mathbf{W}_{xl} \cdot \mathbf{h}_t^{(l-1)} + \mathbf{b}_l) \tag{6.3}$$

Deep transition RNN and stacked RNN can be complementary because these two RNNs extend the standard RNN in orthogonal ways [59]. Using the orthogonal properties of the two RNNs, there is an attempt of the new architecture by adding non temporal depth in each level of stacked RNN for neural machine translation [102]. However, while being powerful in principle, these architectures are seldom used recently due to exacerbated gradient propagation issues resulting from extremely long back propagation paths. Therefore, recent researches regarding time series problem tend to apply only stacked RNNs to guarantee sufficient non-linearity in their model instead of adding deep transition layers in it [86, 87, 103].

6.2 Recurrent Skip Connections

The fact that a stacked RNN architecture provides a sufficient number of non-linearity enable to learn doesn't guarantee the RNNs readily to learn. Rather, an unnecessarily sufficient number of non-linearities, along with the axis of 'depth' or 'time', obstruct the learning process, as derivative of the non-linear activation functions can cause vanishing/exploding gradient in BPTT phase. This phenomenon is observed more prominently in learning long sequences because memorizing extremely long-term dependencies while maintaining mid- and short-term memory is difficult [29]. For intuitive understandings, skip connections in this section is discussed relying on single layer RNNs, instead of stacked RNNs. Thus, non-linearities exists only along the axis of 'time', not 'depth'.

There are two common ways in which RNNs can learn long sequences. One option is to utilize the gated structure of a recurrent cell, such as LSTM [85] and GRU [88]. The gated recurrent cells are based on the idea of creating paths through time that have derivatives neither vanish nor explode [104] and that enable information to propagate forward and backward over longer time steps than ordinary recurrent cell such as Elman cell.

Another option is to use skip connections through time in the RNN architecture. Skip connection is defined to add a direct connection from a cell in distant past to a present cell. The difference to an ordinary RNN, where a recurrent connection goes from a cell at one previous time step to a present cell, is that a recurrent cell with skip connection at time t have two recurrent connections from previous time steps, namely, $t-1$ and $t-s$.

$$\mathbf{h}_t = f(\mathbf{h}_{t-s}, \mathbf{h}_{t-1}, \mathbf{x}_t) \tag{6.4}$$

Equation 6.5 provides formal definition of recurrent neural network with skip connections, where s is referred as 'skip length'. Compared with Equation 3.2, the model with skip connections in Equation 6.4 includes hidden state term from s time step earlier.

Through the skip connections, the RNNs can skip non-linearities and shorten the number of time step when the network transport information over time. Gradients also back propagate quicker across time steps [99, 105]. Goodfellow et al. [104] refers that skip connections help the RNNs to obtain slower time scales. Figure 6.2 shows the recurrent neural network with skip connection $s = 2$.

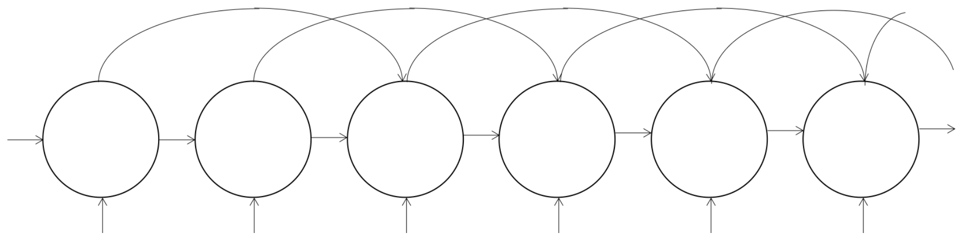


FIGURE 6.2: Recurrent Neural Network with skip connection $s = 2$. The RNN updates the state through two recurrent connections with length 1 and $s = 2$.

6.3 Stacked RNNs with Skip Connections

$$\mathbf{h}_t^{(l)} = f(\mathbf{h}_{t-1}^{(l)}, \mathbf{h}_{t-s}^{(l)}, \mathbf{h}_t^{(l-1)}) \tag{6.5}$$

El Hihi and Bengio [99] suggest an architecture combining stacked RNNs and skip connections. This architecture is a stack of multiple single layer RNNs where each single layer RNN has a different skip length. Skip length of each level is predefined by the purpose of the task and the structure of dataset. Depending on the skip length, each level of the RNNs learns at a different time scale. Figure 6.3 shows one simple architecture of stacked RNN with skip connections.

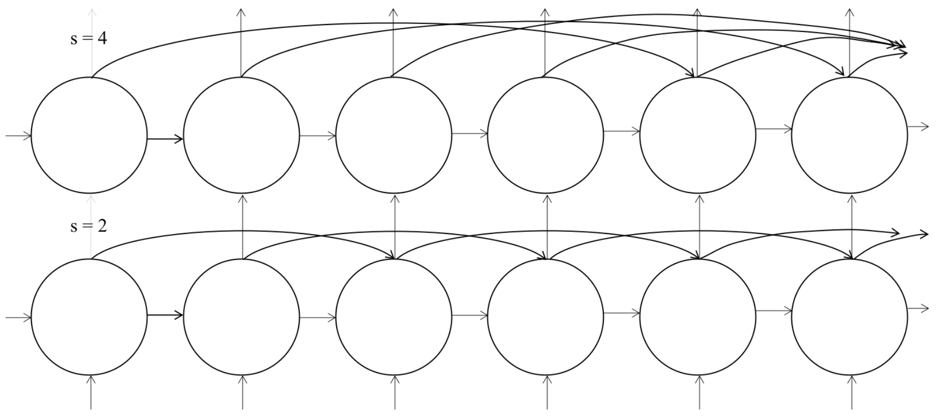


FIGURE 6.3: Hierarchical RNN with skip connection $s = \{2, 4\}$

This architecture is applicable on the tasks in which the dataset have both slower-varying components with faster-varying ones over time, for example, speech recognition and synthesis [106, 107] and financial and economic forecasting [108]. Recent studies in the field of NLP utilize the architecture in allocating different semantic analysis in each level, for example, the lowest level for word analysis and the upper level for phrase analysis, using more advanced architecture referred as clockwork RNN [109], hierarchical multi-scale RNN [103] and so on.

6.4 Dilated RNNs

Stacked RNNs with skip connections can capture multiple time scales by the different skip length in each layer. However, they still have limitations in stacking up many layers because both forward and back propagation are performed in a sequential manner.

Specifically, length 1 recurrent connections commonly existing over all layers result in the RNNs unnecessarily complex.

Hence, Chang et al. [29] suggest a multi-layer, and cell-independent architecture called dilated RNNs, characterized by dilated recurrent skip connections, referring 'dilations'. Recurrent cells with dilations are simplified by removing commonly existing length 1 recurrent connections while keeping longer term skip connections. Therefore, recurrent cells in each layer of dilated RNNs have only one recurrent connection with different skip length which provides more flexibility and capability of modeling time scales and longer dependencies. Figure 6.4 shows the architecture of dilated RNN with dilations $d = \{1, 2, 4\}$.

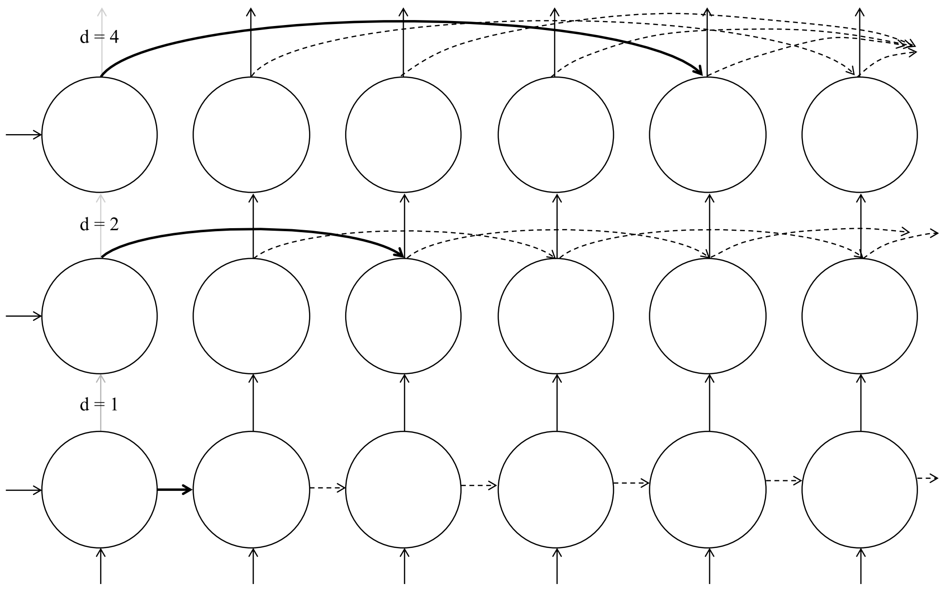


FIGURE 6.4: Dilated RNN with dilations $d = \{1, 2, 4\}$

6.4.1 Dilated Recurrent Skip Connections

$$\mathbf{h}_t^{(l)} = f(\mathbf{h}_{t-d}^{(l)}, \mathbf{h}_t^{(l-1)}) \tag{6.6}$$

Dilated Recurrent skip connections, referring dilations is defined by Equation 6.6, where $l = 1, 2, \cdots L$ representing the index of the levels of the architecture.

Compared with Equation 6.5, 6.6 shows the elimination of the term $\mathbf{h}_{t-1}^{(l)}$ which represents length 1 connection to the cell at time step t in the same layer l. The elimination brings significant enhancement of computational efficiency because the networks require fewer parameters [29]. The saved cost of computation can be invested into capturing other hidden time dependencies by stacking another recurrent units with dilations.

6.4.2 Exponentially Increasing Dilations

By so stacking multiple layers of a RNN with a different dilation, dilated RNNs can capture multiple time dependencies and aggregate multi-scale temporal context. The setting that literature suggests is to let dilations have exponentially increasing length, as introduced in WaveNet [107] and dilated CNN [110]. Let $d^{(l)}$ and d_0 be value of the dilation and initial value of the dilation, where M, $l = 1, 2, \cdots L$ are common ratio of dilation and the index of the layers.

$$d^{(l)} = d_0 M^{l-1} \tag{6.7}$$

There are two benefits to have exponentially increasing dilations. First, it makes different layers focus on different time scales. Second, it reduces the average length of paths between nodes at different time steps, called mean recurrent length [105], which improves the ability of RNN to extract long-term dependencies and prevents vanishing and exploding gradients [29]. In general, initial dilation d_0 is set to one to let the first layer capture time dependency through length 1 recurrent connection.

Chapter 7

Missing Data Analysis with RNNs

For scientific researchers missing data is a common problem because their possible effect on the results of the investigation is seldom quantified despite the fact that they are a likely source of bias [26]. Statisticians suggest sophisticated methods to estimate missing values such multiple imputation (MI) [111], full information maximum likelihood (FIML) [112]. However, despite good performance, they have limitation because these methods require statistical assumptions for the dataset, that somehow restrict the flexibility of methods.

On the other hand, machine learning approaches are more flexible with no or little prior assumptions. In addition, these are more capable of processing outliers, missing and noisy data [113]. Yi et al. [114] estimate the missing values in air quality data with geographical features by utilizing correlation in multiple points of view, called multi-view learning. Wang et al. [115] apply collaborative filtering to estimate missing values in recommendation system. Schnabel et al. [116] employ matrix completion method to estimate missing value in training and evaluation the recommender system.

Machine learning approach expands to missing data problems in time series. In the literature of Yu et al. [117] which suggests matrix factorization with temporal regularization, matrix completion method is applied for estimating missing values in multivariate time series.

RNNs significantly contributes to the analysis of missing data problem in time series. It is reasonable because RNNs' property, which consider not only the values (or substitution values if missing) at the current time step, but also all the previous information over time, enables the network somehow to learn the missing pattern for the purpose of minimizing the output error [21, 22]. Lipton et al. [118] argue that RNNs successfully attain classification task with missing data in multivariate clinical records.

Before initiating a recurrent dynamic, manual imputation techniques are introduced for missing region, such as mean substitution, zero imputation and last value carried forward to fill up the missing region by plausible values [27]. The techniques are simple to implement. However, strategy of choosing a proper technique should be carefully considered because it can deteriorate performance due to the bias occurred by the imputation [119], for example, underestimating the variance. In the aforementioned study from Lipton et al. [118] compares classification accuracy according to imputation techniques, zero imputation and last value carried forward. Detail and effect of each manual imputation techniques will be discussed in this chapter.

RNN analysis for missing data in time series differs into two groups. The first aims to estimate missing value explicitly, trying to find the most probable values to the corresponding missing value using recurrent nature [21, 120, 121]. As it consider each missing value as a variable to estimate, manual imputation for missing values is not necessary. While, the other group directly tackles to the task, such as classification, short term load forecasting and so on, to achieve the minimum loss rather than estimate the exact value in the missing region [122–124]. Manual imputation techniques are usually applied for the missing values.

As the missing values and patterns also provide rich information, called 'informative missingness' [125], recent studies put an effort to systemically model missing patterns into recurrent dynamic without manual imputation. Che et al. [90] suggest a modified RNN cell structure for time series classification with missing clinical data. Yoon et al. [126] suggest multi-directional RNN (M-RNN) to estimate missing values in multivariate time series under a recurrent dynamic. M-RNN utilize information in two ways, referring intra-stream and inter-stream. Intra-stream utilizes longitudinal information of each variate using bi-directional RNN (bi-RNN) [127] and inter-stream utilizes correlation among the variates in multiple time steps. Cao et al. [128] suggest Bi-directional

Recurrent Imputation for Time Series (BRITS) that enable to train the network for missing value estimation and classification simultaneously.

7.1 Types of Missing Data

It is important to distinguish between the types of missing data because the assumptions that why the data are missing can affect the underlying assumptions of the statistical modelling techniques that will be applied [129]. Statisticians found out that the approach should differ depending on the type of 'missingness', namely, missing completely at random (MCAR), missing at random (MAR), not missing at random (NMAR) [28].

7.1.1 Missing Completely at Random (MCAR)

MCAR is assumed if the events that lead to any particular item being missing are independent both of observable variables and of unobservable parameters of interest, and occur entirely at random [130]. When data are MCAR, the analysis performed on the data is unbiased. That is, the reduced dataset would represent a randomly drawn sub-sample of the original data. In practice, it is usually difficult to meet the MCAR assumption [28].

7.1.2 Missing at Random (MAR)

MAR is an assumption that the missingness is not MCAR, but where missingness can be fully accounted for by variables where there is complete information. That is, the pattern of missingness is traceable or predictable from other variates in the dataset, rather than appear the specific variate on which the data are missing [28]. Depending on the analysis method, these data can still induce parameter bias in analyses due to the contingent missingness [129].

7.1.3 Not Missing at Random (NMAR)

NMAR is the case except two other cases above. The pattern of missingness is non-random and it is not predictable from other variables in the dataset. In other words, the probability that an observation is missing is directly related to the means of measurement. This is sometimes referred to as non-ignorable because this cannot be ignored in the modelling process [129].

7.2 Manual Imputation Techniques

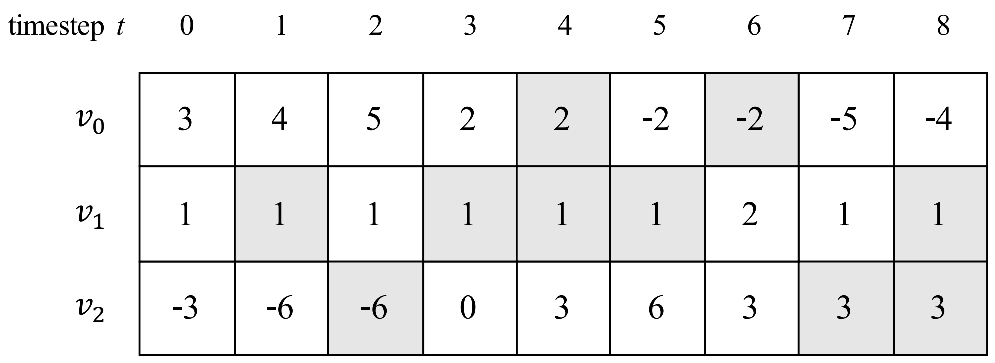
timestep t	0	1	2	3	4	5	6	7	8
v_0	3	4	5	2		-2		-5	-4
v_1	1		1				2	1	
v_2	-3	-6		0	3	6	3		

FIGURE 7.1: Missing data example in multivariate time series. Horizontal axis : time step, vertical axis : variable. Gray colored room illustrates one missing value.

In a broad sense, imputation techniques attempt to estimate the values of the missing data and fill-in or substitute new values [28]. Once this has been achieved then the researcher can proceed as the dataset were considered complete. For the task using RNNs such as classification, short term load forecasting and so on, traditional approach is based on the manual imputation. The most simple three techniques will now be described. Figure 7.1 shows the example of multivariate missing data in time series. In the figure, horizontal axis represents timestep while vertical axis represents each variable.

7.2.1 Last Value Carried Forward

Last value carried forward technique is to carry the last available value and impute this value for the missing values until new available value exists. Figure 7.2 shows this technique based on the multivariate missing data example. The technique is under the assumption that there will be no change within the missing period of time. If the assumption is not satisfied, dataset will be biased. This will also cause serious bias if the data is not MCAR [28]. Formal expression of last value carried forward technique is shown in the Equation 7.1. For x_t^v, a value of variable v at timestep t,



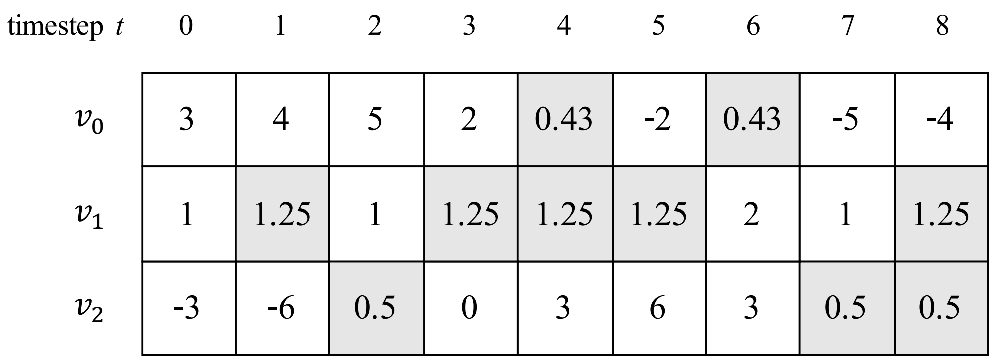
timestep t	0	1	2	3	4	5	6	7	8
v_0	3	4	5	2	2	-2	-2	-5	-4
v_1	1	1	1	1	1	1	2	1	1
v_2	-3	-6	-6	0	3	6	3	3	3

FIGURE 7.2: Last value carried forward for the multivariate missing data

$$x_t^v = \begin{cases} x_t^v, & \text{if } x_t^v \text{ is observed} \\ x_{t'}^v, & \text{otherwise} \end{cases} \tag{7.1}$$

where $t^{'} < t$ is the timestep of lastly observed value of the variable v.

7.2.2 Mean Substitution



timestep t	0	1	2	3	4	5	6	7	8
v_0	3	4	5	2	0.43	-2	0.43	-5	-4
v_1	1	1.25	1	1.25	1.25	1.25	2	1	1.25
v_2	-3	-6	0.5	0	3	6	3	0.5	0.5

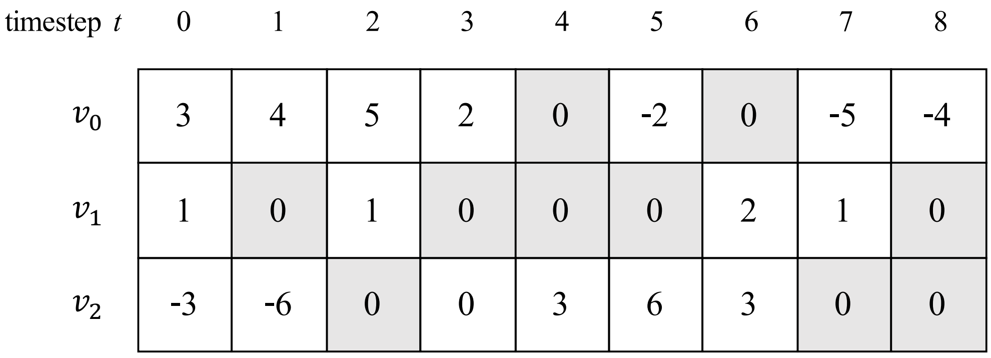
FIGURE 7.3: Mean substitution for the multivariate missing data

Mean substitution technique is to replace missing value with the mean value within the same variable. In Figure 7.3, gray colored room is filled with mean value of corresponding variable. The mean value is computed by 'observed values'. This approach assumes the data is MCAR but is not recommended as it can lead to under-estimate of the variance [28]. Formal expression of mean substitution technique is shown in the Equation 7.2. For x_t^v, a value of variable v at timestep t,

$$x_t^v = \begin{cases} x_t^v, & \text{if } x_t^v \text{ is observed} \\ \overline{x^v}, & \text{otherwise} \end{cases} \tag{7.2}$$

where $\overline{x^v}$ is the mean of observed values of the variable v.

7.2.3 Zero Imputation



timestep t	0	1	2	3	4	5	6	7	8
v_0	3	4	5	2	0	-2	0	-5	-4
v_1	1	0	1	0	0	0	2	1	0
v_2	-3	-6	0	0	3	6	3	0	0

FIGURE 7.4: Zero imputation for the multivariate missing data

Zero imputation technique is the simplest thus frequently used technique (see Figure 7.4). This results in similar effects with mean substitution technique when each variable has zero-mean. If not, it can cause more serious bias than mean substitution. Equation 7.3 shows the formal expression.

$$x_t^v = \begin{cases} x_t^v, & \text{if } x_t^v \text{ is observed} \\ 0, & \text{otherwise} \end{cases} \tag{7.3}$$

7.3 RNN Analysis with Manually Imputed Values

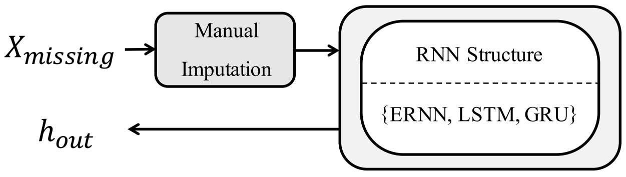


FIGURE 7.5: RNN analysis of missing data using only manual imputation

The simplest approach to deal with missing data in RNNs is to fill up the missing region by manual imputation techniques and run RNNs with the imputed time series(see Figure

7.5). However, this approach can give rise to severe bias because the RNNs don't know whether the input value is imputed or observed. This phenomenon is getting worse in the cases that the data is missing for a long time successively or that the ratio of missing to observed values is larger.

Lipton et al. [118] report that RNNs can realize the imputed values with only simple binary indicators for missingness, referring binary mask **M**. Binary mask describes observed/missing values of input **X** at each time step t and variate v. the binary mask m_t^v for input x_t^v is given by Equation 7.4. Figure 7.6 illustrates binary mask in accordance with the missing data example in Figure 7.3.

$$m_t^v = \begin{cases} 1, & \text{if } x_t^v \text{ is observed} \\ 0, & \text{otherwise} \end{cases} \tag{7.4}$$

timestep t	0	1	2	3	4	5	6	7	8
v_0	1	1	1	1	0	1	0	1	1
v_1	1	0	1	0	0	0	1	1	0
v_2	1	1	0	1	1	1	1	0	0

FIGURE 7.6: Binary mask for the multivariate missing data

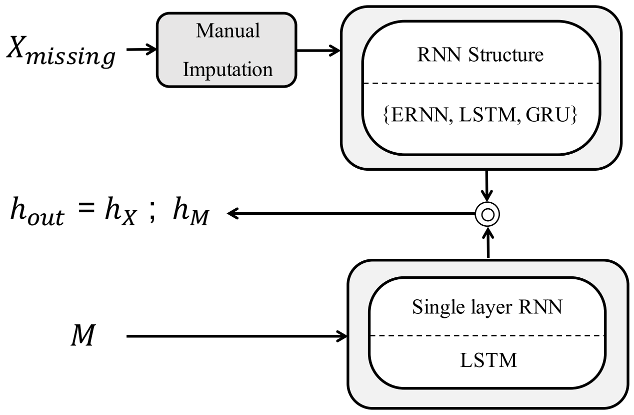


FIGURE 7.7: RNN analysis of missing data using manual imputation and binary mask. Binary mask sequence **M** is fed into external RNNs which are jointly trained with RNNs having $\mathbf{X}_{missing}$ as an input.

In this study, binary mask $\mathbf{M}$ is fed into an LSTM cell so that the RNN learn the history of missingness, shown in Figure 7.7. State output from $\mathbf{X}$ (upper cell in the figure) and another state output from $\mathbf{M}$ (lower cell in the figure) are concatenated to be an aggregated output for the RNN.

7.4 RNN Analysis for Systematical Modelling of Missing Patterns : GRU with Trainable Decay (GRU-D)

Recent studies [90, 126, 128] attempt to include the imputation procedure into a recurrent dynamic, that is, manual imputation is not required. To put it concretely, this approach modifies the structure of the recurrent cell to be able to impute the missing value with a plausible value using previous information processed by RNNs. Che et al. [90] suggest a modified cell structure from GRU for a missing pattern modelling, referring GRU with trainable decay (GRU-D). Strauman et al. [119] report that GRU-D provides an outstanding performance in classification task compared with other manual imputation techniques. In this section, details of GRU-D are discussed.

To deal with missing values in input data $\mathbf{X}$, binary mask $\mathbf{M}$ and missing time interval $\boldsymbol{\Delta}\}$ are newly introduced for GRU-D. For formal expression, a multivariate time series with m variates of length T is denoted as $\mathbf{X} = \{\mathbf{x}_0, \mathbf{x}_1, \cdots, \mathbf{x}_{T-1}\} \in \mathbf{R}^{m \times T}$, where $\mathbf{x}_t \in \mathbf{R}^m$ represents t^{th} observations of all variates and x_t^v denotes the observation of v^{th} variate of $\mathbf{x}_t$. In the same context, binary mask $\mathbf{M} \in \mathbf{R}^{m \times T}$ and missing value interval $\boldsymbol{\Delta} \in \mathbf{R}^{m \times T}$ are denoted where m_t^v and δ_t^v denote the observation of v^{th} variable at time t of $\mathbf{m}_t$ and $\boldsymbol{\delta}_t$ respectively.

7.4.1 Comparison between GRU-D and GRU

As shown in the Figure 7.8, GRU-D shares the gated structure of conventional GRU such as update gate and reset gate. Also, two noticeable changes are found at input node (gray shaded in the figure) and trainable decay γ_x and γ_h (red circle). Che et al. [90] claim that these two changes are independent from RNN cell structure thus applicable to the other RNN cells.

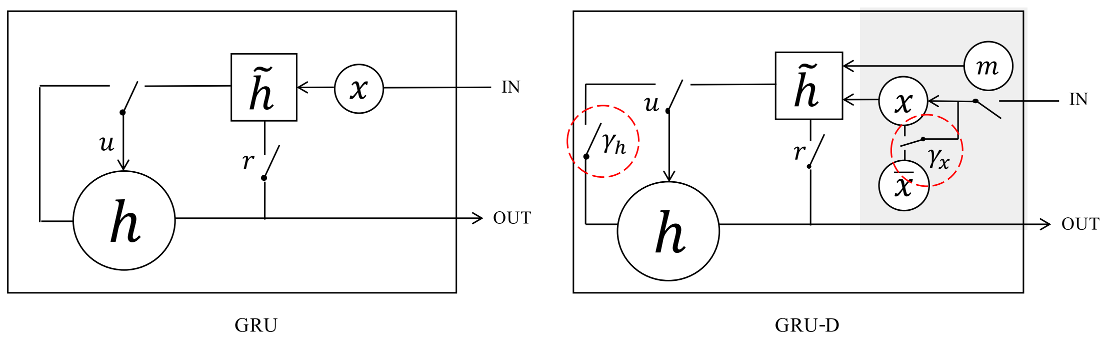


FIGURE 7.8: Schema of GRU and GRU-D. Two differences are found at input node gray-shaded region and extra switches in red dot-circled region.

7.4.2 Change in Input Node

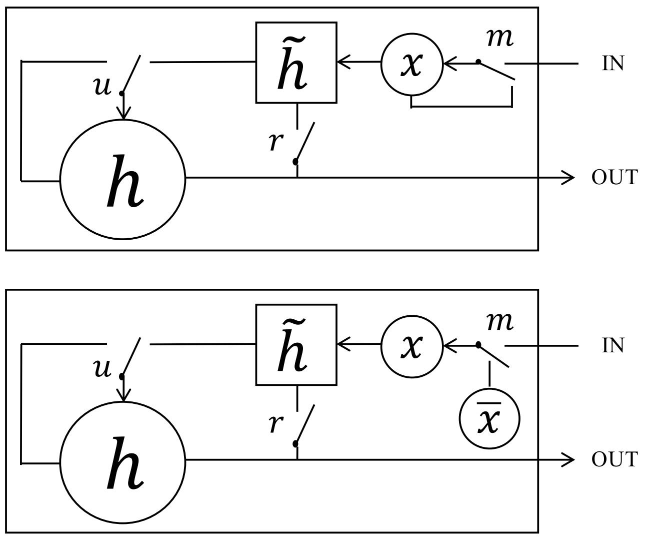


FIGURE 7.9: Schema of GRU for missing data. upper : last value carried forward. lower : mean substitution

In GRU-D, manual imputation is not operated thus missing values are remained by null in input **X**. The change of input node of RNN cell enables the imputation procedure using binary mask m_t^v within the cell. Equation 7.5 denotes input node modification in terms of two aforementioned imputation techniques, last value carried forward technique and mean substitution. Figure 7.9 describes them in the context of GRU.

$$\begin{aligned} x_t^v &\leftarrow m_t^v x_t^v + (1 - m_t^v) x_{t'}^v \\ &\quad : \text{ last value carried forward} \\ x_t^v &\leftarrow m_t^v x_t^v + (1 - m_t^v) \overline{x^v} \\ &\quad : \text{ mean substitution} \end{aligned} \tag{7.5}$$

GRU-D incorporate two imputation techniques, last value carried forward and mean substitution with input decay γ_x. Therefore, if the input x_t^v is missing, input node of GRU-D computes the substitute value $\widehat{x_t^v}$ between the value from last value carried forward $x_{t'}^v$ and mean substitution value $\overline{x^v}$ depending on $\gamma_{x,t}^v$. Equation 7.6 provides the formal expression.

$$\widehat{x_t^v} \leftarrow m_t^v x_t^v + (1 - m_t^v)\{\gamma_{x,t}^v \cdot x_{t'}^v + (1 - \gamma_{x,t}^v)\overline{x^v}\} \tag{7.6}$$

7.4.3 Two Trainable Decays : $\boldsymbol{\gamma_x}$ and $\boldsymbol{\gamma_h}$

The change in input node is known to have empirical limitations if its last observation happens a long time ago. First, the influence of the inputs is only significant in a certain temporal context and fade away over time [131]. Second, the missing value tend to be close to some default value (e.g mean or last value) [132].

GRU-D improves the limitations by applying decay mechanism into the RNN cells. Two decay rate vectors, input decay rate $\boldsymbol{\gamma_{x,t}} \in \mathbf{R}^m$ and state decay rate $\boldsymbol{\gamma_{h,t}} \in \mathbf{R}^s$, standing for decay mechanism, imply two important ideas. First, decay rate should be considered as a vector because each variable of the time series has its own meaning and importance. Secondly, decay rate should learn from the dataset instead of having a fix priori because missing patterns are usually unknown and complex.

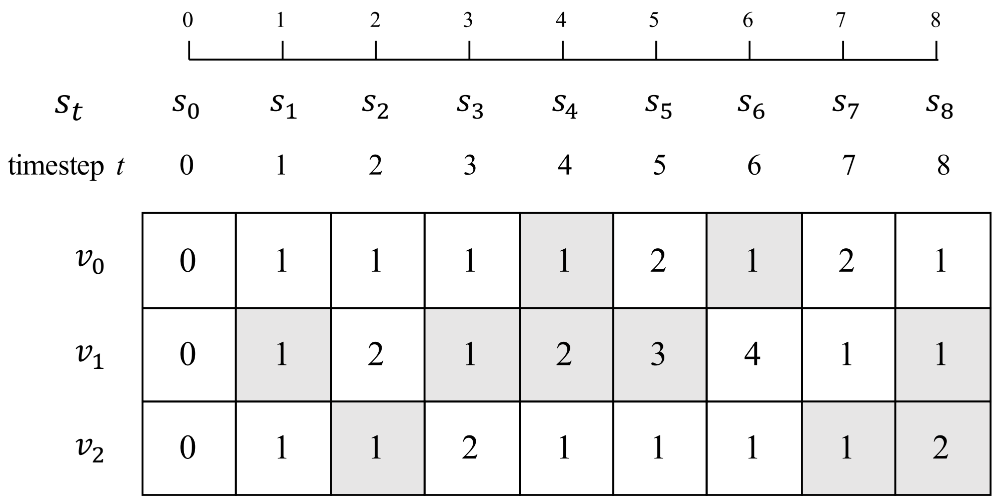
To prevent fluctuation of the decay rates in long missing time intervals, each decay rate monotonically decreases in a reasonable range between 0 and 1. $\gamma_{x,t}^v$, v^{th} variable of input decay rate vector $\boldsymbol{\gamma_{x,t}}$ and $\gamma_{h,t}^q$, q^{th} variable of state decay rate vector $\boldsymbol{\gamma_{h,t}}$ are denoted in Equation 7.7, where $\mathbf{W}_{x,\gamma} \in \mathbf{R}^{m \times m}$, $\mathbf{b}_{x,\gamma} \in \mathbf{R}^m$, $\mathbf{W}_{h,\gamma} \in \mathbf{R}^{s \times m}$, $\mathbf{b}_{h,\gamma} \in \mathbf{R}^s$.

Note that $\mathbf{W}_{x,\gamma}$ constrains to be diagonal to make input decay rate of each variates independent.

$$
\begin{aligned}
\gamma^{v}_{x,t} &= exp\big[-max\{0, (\mathbf{W}_{x,\gamma} \cdot \boldsymbol{\delta}_t + \mathbf{b}_{x,\gamma})^{v}\}\big] \\
\gamma^{q}_{h,t} &= exp\big[-max\{0, (\mathbf{W}_{h,\gamma} \cdot \boldsymbol{\delta}_t + \mathbf{b}_{h,\gamma})^{q}\}\big]
\end{aligned} \tag{7.7}
$$

, where $\boldsymbol{\delta}_t$ represents duration of missingness over time and helps RNNs to encapsulate hidden temporal missing pattern, referring missing time interval. Missing time interval is defined by actual observation time s_t which is introduced to incorporate time step t and actual observation time efficiently. For convenience sake, initial time for the first timestep s_0 is set to zero. Equation 7.8 reveals the definition of missing time interval $\boldsymbol{\delta}_t$.

$$
\delta^{v}_{t} = \begin{cases} s_t - s_{t-1} + \delta^{v}_{t-1}, & t > 1, m^{v}_{t-1} = 0 \\ s_t - s_{t-1}, & t > 1, m^{v}_{t-1} = 1 \\ 0, & t = 0 \end{cases} \tag{7.8}
$$



s_t	s_0	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8
timestep t	0	1	2	3	4	5	6	7	8
v_0	0	1	1	1	1	2	1	2	1
v_1	0	1	2	1	2	3	4	1	1
v_2	0	1	1	2	1	1	1	1	2

FIGURE 7.10: Regularly observed missing time interval. $s_t = \{0, 1, 2, 3, 4, 5, 6, 7, 8\}$

An example of missing time interval δ^{v}_{t} are shown in Figure 7.10 based on the actual observation time s_t in the upper side of each figure. Compared with binary mask m^{v}_{t}, the pattern of missing time interval δ^{v}_{t} shows one timestep lagged from binary mask.

Each decay rate commonly utilizes the history of missingness from missing time interval $\boldsymbol{\delta_t}$ but regulates the amount of information in a different way for each input $\mathbf{x}_t$ and state $\mathbf{h}_t$. Input decay rate $\boldsymbol{\gamma_{x,t}}$ facilitates the input candidate $\widehat{\mathbf{x}_t}$ to have plausible value between the last value and mean of the observed value if the value is missing as shown in Equation 7.6. State decay rate $\boldsymbol{\gamma_{h,t}}$ aids to capture missing patterns by protecting the previous state $\mathbf{h}_{t-1}$ from long and successive missing values. State candidate $\widehat{\mathbf{h}_{t-1}}$ is defined in Equation 7.9, element-wise product of (current) state decay rate $\boldsymbol{\gamma_{h,t}}$ and (previous) hidden state $\mathbf{h}_{t-1}$.

$$\widehat{\mathbf{h}_{t-1}} = \boldsymbol{\gamma_{h,t}} \odot \mathbf{h}_{t-1} \tag{7.9}$$

$$\begin{aligned}
\mathbf{u}_t &= \sigma(\mathbf{W}_{hu} \cdot \widehat{\mathbf{h}_{t-1}} + \mathbf{W}_{xu} \cdot \widehat{\mathbf{x}_t} + \mathbf{W}_{mu} \cdot \mathbf{m}_t + \mathbf{b}_u) \\
\mathbf{r}_t &= \sigma(\mathbf{W}_{hr} \cdot \widehat{\mathbf{h}_{t-1}} + \mathbf{W}_{xr} \cdot \widehat{\mathbf{x}_t} + \mathbf{W}_{mr} \cdot \mathbf{m}_t + \mathbf{b}_r) \\
\tilde{\mathbf{h}}_t &= tanh\{\mathbf{W}_{hh} \cdot (\mathbf{r}_t \odot \widehat{\mathbf{h}_{t-1}}) + \mathbf{W}_{xh} \cdot \widehat{\mathbf{x}_t} + \mathbf{W}_{mh} \cdot \mathbf{m}_t + \mathbf{b}_h\} \\
\mathbf{h}_t &= (\mathbf{1} - \mathbf{u}_t) \odot \widehat{\mathbf{h}_{t-1}} + \mathbf{u}_t \odot \tilde{\mathbf{h}}_t
\end{aligned} \tag{7.10}$$

Referring the formal expression of GRU, $\widehat{\mathbf{x}_t}$ from 7.6 and $\widehat{\mathbf{h}_{t-1}}$ from 7.9, GRU-D is defined in Equation 7.10. Note that unlike conventional GRU, binary mask $\mathbf{m}_t \in \mathbf{R}^m$ is involved in controlling the gates $\mathbf{u}_t$ and $\mathbf{r}_t$ also generating the candidate $\tilde{\mathbf{h}}_t$ with its weight parameters, $\{\mathbf{W}_{mu}, \mathbf{W}_{mr}, \mathbf{W}_{mh}\} \in \mathbf{R}^{s \times m}$.

Chapter 8

Proposed Approach : STLF with Missing Data Through Dilated RNNs and Attention

8.1 STLF with Missing Data Through standard RNNs

Standard RNN architectures implementing self connections with time lag 1 can be inefficient for STLF tasks on missing data. Very often, missing values tend to be observed consecutively over a certain period of time, i.e. there are large time windows (for example 1 week) where data are missing. As standard architecture transfers information through recurrent connection with time delay 1, the internal state of the network is updated many times with 'fake' input values (e.g. zeros if missing values are replaced with zero imputation). This gradually destroys the content of the RNN state in presence of long sequences of missing values.

Figure 8.1 depicts an unfolded graph of standard architecture for a STLF task, where 3 consecutive values $\{x_{t-1}, x_t, x_{t+1}\}$ are missing (thus manually imputed). It is analogous that the last prediction of missing region p_{t+1} is less reliable than other predictions because three unreliable inputs $\{x_{t-1}, x_t, x_{t+1}\}$ are fed into the RNN sequentially and involved in the computation procedure of p_{t+1}. Analogously, the forecasting accuracy are influenced by the choice of manual imputation techniques.

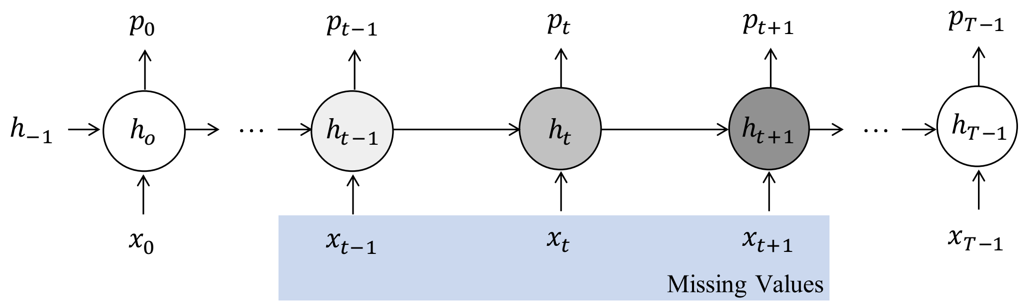


FIGURE 8.1: Unfolded graph of standard architecture for a time series with 3 missing values. Last prediction of missing region p_{t+1} is less reliable than other predictions because three unreliable inputs $\{x_{t-1}, x_t, x_{t+1}\}$ are fed into the RNN sequentially.

8.2 Proposed Approach : Dilated RNNs with Attention

In this thesis we propose a framework to deal with missing data problems using RNNs. The proposed framework consists of dilated RNNs [29] combining with a modified attention mechanism [30] to handle missing data.

Intuitively, updating the state less often will protect the RNNs from the bias caused by a long window of consecutively imputed values. In unfolded graph, it is interpreted by a utilizing a long skip connection or dilation. On the other hand, the RNNs don't need to exploit from the long dilation if the values are correctly observed. In the proposed approach, dilated RNNs provide multiple layers with different length of dilations and a modified attention mechanism enables the RNNs to learn how to exploit the layers with different dilations in order to achieve minimum error for STLF tasks.

8.2.1 Dilated RNNs for Missing Data

Dilated RNNs are a stack of multiple single layer RNN with different dilations. Recurrent cells in each layer of dilated RNNs have a dilation with different length which provides more flexibility and capability of modeling time scales and longer dependencies. Flexibility of dilated RNNs, given by the different lengths of dilation, can improve the accuracy of the forecast in the missing region. Each layer of the network can capture different temporal dependencies by utilizing its own dilation. As a result, each layer participates in the forecast with its own information from different time steps in the past.

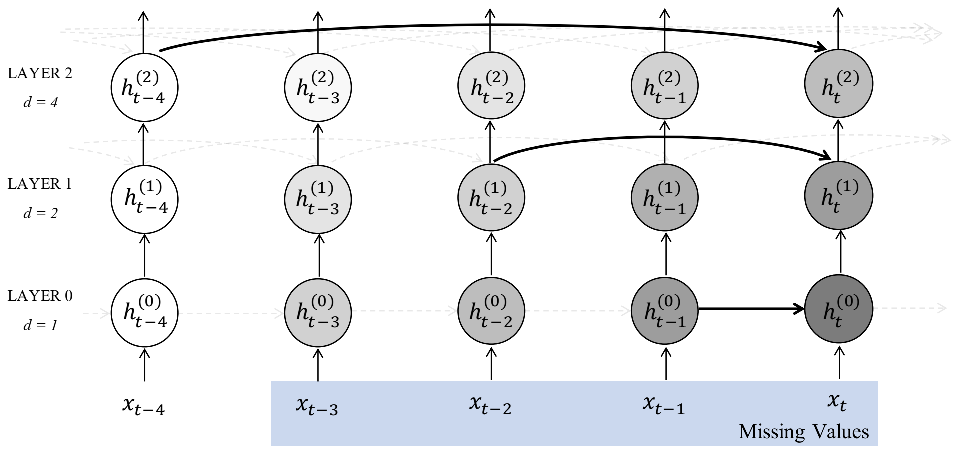


FIGURE 8.2: Dilated RNNs with dilation $d = \{1, 2, 4\}$ for missing data analysis. Bias will decrease when forecasting p_t by allocating more importance to the state in layer 2 than layer 0. In this manner, the forecast accuracy of missing data can be improved by exploiting multiple time scales.

Figure 8.2 depicts the inflow of information at time step t in dilated RNN with $d = \{1, 2, 4\}$ and missing values $\{x_{t-3}, x_{t-2}, x_{t-1}, x_t\}$. Due to unreliable inputs in the missing region, states in the blue box of the figure is somehow biased. But the effect of bias is not identical for each layer. For example, let's take the components belonging to the time step t. The state at layer 0, $h_t^{(0)}$, has been updated with imputed values four time steps before while the state at layer 2, $h_t^{(2)}$, is never updated with imputed values. Thus, it can assumed that bias will decrease when forecasting p_t by allocating more importance to the state in layer 2 than layer 0. In this manner, the forecast accuracy of missing data can be improved by exploiting multiple time scales. $h_t^{(l)}$, state at time step t and layer l expressed in Equation 8.1 where $l = 0, 1, 2$.

$$\begin{aligned} h_t^{(0)} &= f(h_{t-1}^{(0)}, x_t) \\ h_t^{(1)} &= f(h_{t-2}^{(1)}, h_t^{(0)}) \\ h_t^{(2)} &= f(h_{t-4}^{(2)}, h_t^{(1)}) \end{aligned} \tag{8.1}$$

8.2.2 Attention Mechanism

Attention mechanism is introduced by Bahdanau et al. [30] in the field of neural machine translation. Neural machine translation is based on the framework of RNN Encoder Decoder, proposed by Cho et al. [31]. With an encoder and a decoder for each language,

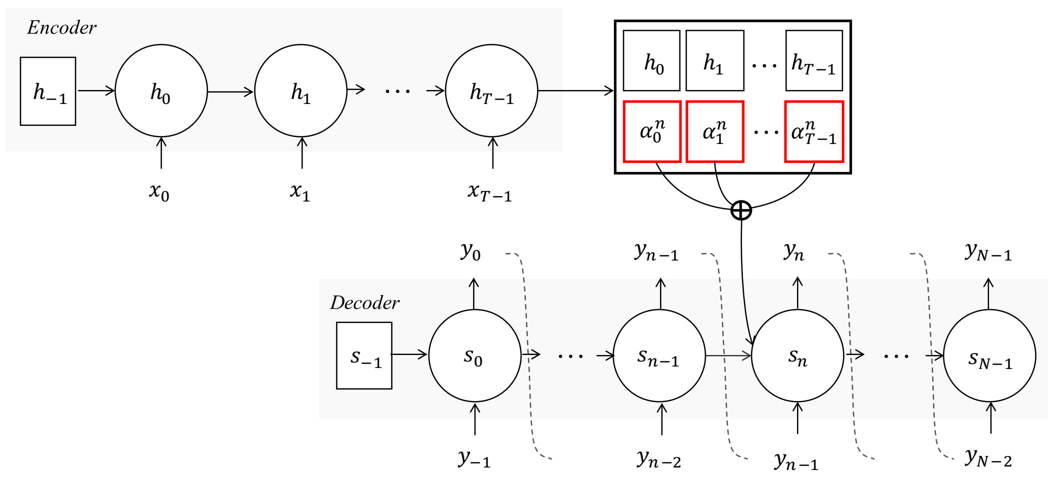


FIGURE 8.3: Attention mechanism in a encoder decoder framework. The attention mechanism is to focus on specific internal states generated by encoder at each decoding step. The contribution of each state is governed by means of a weighted average.

this framework applies to each sentence whose outputs are compared [133]. An encoder neural network reads and encodes a source sentence into a fixed length vector. A decoder then outputs a sequence with variable length from the encoded vector. The whole encoder decoder system consists of the encoder and the decoder for a language pair which are jointly trained to maximize the probability of a correct translation given a source sentence [30].

The attention mechanism is to focus on specific internal states generated by encoder at each decoding step. The contribution of each state is governed by means of a weighted average. Figure 8.3 illustrates the attention mechanism in a encoder decoder framework. An aggregated information of input formed by taking a weighted average of encoder state h_t, $t = 0, 1, \cdots T-1$ with weights α_t^n, is transferred as input to the decoder that modifies its states s_n $n = 0, 1, \cdots N-1$. The weights α_t^n usually take values in the interval $\left[0, 1\right]$ so that the weighted average decides specic time steps to allocate importance.

$$\begin{aligned} s_n &= f(s_{n-1}, y_{n-1}, c_n) && \text{Decoder state} \\ c_n &= \sum_{t=0}^{T-1} \alpha_t^n h_t && \text{Weighted average} \\ \alpha_t^n &= \frac{exp(e_t^n)}{\sum_{k=0}^{T-1} exp(e_k^n)} && \text{Softmax} \\ e_t^n &= g(s_{n-1}, h_t) && \text{Feedforward NN} \end{aligned} \tag{8.2}$$

The weights α_t^n are produced by the scores e_t^n, which represents how well the inputs around position t and the output at position n match. The scores are computed by a feedforward network $g(\cdot)$ with the RNN decoder state s_{n-1} and the encoder sentence h_t. Equation 8.2 denotes the formal expression of the attention mechanism [104].

8.2.3 STLF with Missing Data Through Attention and Dilated RNNs

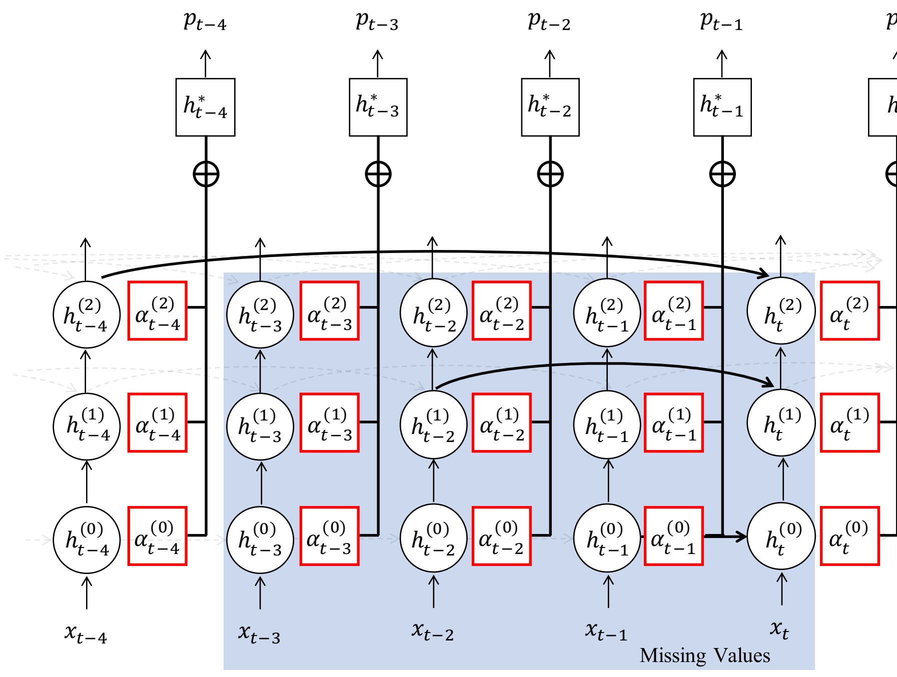

p_{t-4}
p_{t-3}
p_{t-2}
p_{t-1}
p_t
h^*_{t-4}
h^*_{t-3}
h^*_{t-2}
h^*_{t-1}
h^*_t
$h^{(2)}_{t-4}$ $\alpha^{(2)}_{t-4}$ $h^{(2)}_{t-3}$ $\alpha^{(2)}_{t-3}$ $h^{(2)}_{t-2}$ $\alpha^{(2)}_{t-2}$ $h^{(2)}_{t-1}$ $\alpha^{(2)}_{t-1}$ $h^{(2)}_t$ $\alpha^{(2)}_t$
$h^{(1)}_{t-4}$ $\alpha^{(1)}_{t-4}$ $h^{(1)}_{t-3}$ $\alpha^{(1)}_{t-3}$ $h^{(1)}_{t-2}$ $\alpha^{(1)}_{t-2}$ $h^{(1)}_{t-1}$ $\alpha^{(1)}_{t-1}$ $h^{(1)}_t$ $\alpha^{(1)}_t$
$h^{(0)}_{t-4}$ $\alpha^{(0)}_{t-4}$ $h^{(0)}_{t-3}$ $\alpha^{(0)}_{t-3}$ $h^{(0)}_{t-2}$ $\alpha^{(0)}_{t-2}$ $h^{(0)}_{t-1}$ $\alpha^{(0)}_{t-1}$ $h^{(0)}_t$ $\alpha^{(0)}_t$
x_{t-4}
x_{t-3}
x_{t-2}
x_{t-1}
x_t
Missing Values

FIGURE 8.4: Modified attention mechanism in dRNN(3) with dilation $d = \{1, 2, 4\}$. The networks learn how to utilize different time scales for the optimal forecast using weights $\alpha_t^{(l)}$ which decide the contribution of each state from one layer $h_t^{(l)}$. For a long window of consecutive missing values, the network is expected to concentrate on the higher layer which has longer time scales.

The proposed approach applies the attention mechanism on the states $h_t^{(l)}$ from dilated RNN to learn which layer provides the most reliable state each time, in presence of missing values in the input sequence. In other words, the networks learn how to utilize different time scales for the optimal forecast depending on whether the values are missing or not. Figure 8.4 illustrates the attention mechanism in dilated RNNs. The weights $\alpha_t^{(l)}$ are intended to decide the contribution of each state from one layer $h_t^{(l)}$ so that at each time step each layer has a different importance for computing the output. As a result, for a long window of consecutive missing values, the network is expected to concentrate on the higher layer which has longer time scales. This is done by learning higher $\alpha_t^{(l)}$, to suppress bias induced by manual imputation. The aggregated state h_t^*

formed by taking a weighted average of state of each layer $h_t^{(l)}$, $l = 0, 1, 2$ with weights $\alpha_t^{(l)}$, is used for computing the forecast p_t in a general manner. Equation 8.3 denotes the formal expression for L layer dilated RNN, or dRNN(L).

$$\begin{aligned} p_t &= g(h_t^*) && \text{Forecast value} \\ h_t^* &= \sum_{l=0}^{L-1} \alpha_t^{(l)} h_t^{(l)} && \text{Weighted average} \end{aligned} \tag{8.3}$$

8.2.4 Composition of weights $\alpha_t^{(l)}$

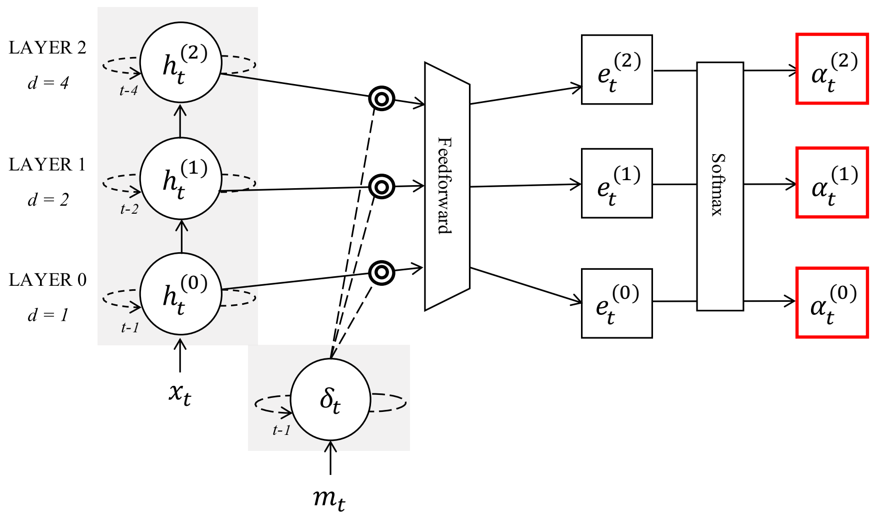


FIGURE 8.5: Schema of weight composition for dRNN(3) in Figure 8.4. The weight $\alpha_t^{(l)}$ is obtained by the score $e_t^{(l)}$ applied by a softmax function. The score is derived by the concatenation of missing history δ_t and the state of the dRNN(3) $h_t^{(l)}$ applied by feedforward NN.

$$\begin{aligned} \alpha_t^{(l)} &= \frac{exp(e_t^{(l)})}{\sum_{k=0}^{L-1} exp(e_t^{(k)})} && \text{Softmax} \\ e_t^{(l)} &= g(h_t^{(l)}; \delta_t) && \text{Feedforward NN} \\ \delta_t &= f(\delta_{t-1}, m_t) && \text{External RNN} \end{aligned} \tag{8.4}$$

The weights $\{\alpha_t^{(l)}\}$, $l = 0, 1, \cdots L-1$ are derived from the scores $\{e_t^{(l)}\}$, $l = 0, 1, \cdots L-1$ processed by the softmax function so that they have values within the interval $\left[0, 1\right]$ and the sum over the layers is one.

As the scores $e_t^{(l)}$ play a role in incorporating current state at each layer and the missing history from binary mask m_t in Equation 8.5, they are derived by the concatenation of two vectors $h_t^{(l)}$ and δ_t, referring state at layer l and history of missingness respectively, processed by a feedforward network.

δ_t is the state of an external (small) RNNs, which is fed with the binary mask sequence m_t. It represents missing history of input data and plays a similar role in missing time interval of GRU-D [90]. It is derived from binary mask time series m_t processed by another RNNs which train jointly, such as LSTM. Formal expression is denoted in Equation 8.4.

$$m_t = \begin{cases} 1, & \text{if } x_t \text{ is observed} \\ 0, & \text{otherwise} \end{cases} \tag{8.5}$$

Chapter 9

Results

In this chapter, we compare the performance achieved by each RNN model presented in Chapter 6 and Chapter 7 in terms of the forecast accuracy of two different time series with missing values. The forecast accuracy is represented by the Mean Squared Error (MSE) obtained on the unseen values of the test set.

The MSE is defined as:

$$\mathrm{MSE}(\mathcal{Y}_k, \mathcal{Y}_k^*) = \frac{1}{|\mathcal{X}_k|} \sum_{\mathbf{x} \in \mathcal{X}_k} (\mathbf{y_x} - \mathbf{y_x^*})^2 \,, \tag{9.1}$$

where $\mathbf{y_x} \in \mathcal{Y}_k$ is the output when the input $\mathbf{x} \in \mathcal{X}_k$ is processed and $\mathbf{y_x^*} \in \mathcal{Y}_k^*$ is the ground-truth value that the network must learn to reproduce. The lower MSE implies the higher forecast accuracy. All models including RNNs used for the experiments have been implemented in Python, using Tensorflow Library [134].

9.1 Datasets

For experiments, we analyze both a synthetically generated time series and a time series from real-world load data from a public dataset, in order to provide controlled and easily replicable results for the architectures under analysis. The synthetic time series, is the generated from the Mackey-Glass system [135], while the real-world time series one comes from the GEFCom 2012 competition [2] and comes from real measurements of electricity load.

In order to obtain a forecasting problem that is not too trivial, it is reasonable to select as forecast horizon a time interval t_f that guarantees the measurements in the time series to become linearly decorrelated. Hence, we consider the first zero of the autocorrelation function of the time series. [4].

9.1.1 Synthetic Time Series : Mackey-Glass (MG) system

The Mackey-Glass (MG) system is commonly used as benchmark for forecasting chaotic time series. The input signal is generated from the MG time-delay differential system, described in Equation 9.2.

$$\frac{dx}{dt} = \alpha \cdot \frac{x_{t-\tau_{\mathrm{MG}}}}{1 + x_{t-\tau_{\mathrm{MG}}}^{10}} - \beta \cdot x_t. \tag{9.2}$$

For the forecasting task, we set $\tau_{\mathrm{MG}} = 17, \alpha = 0.2, \beta = 0.1$, initial condition $x_0 = 1.2$. Integration step for Equation 9.2 is set to 0.1. The forecast horizon t_f is set to 12, which is the first time step where the autocorrelation goes to zero.

All the time series considered in the following consist of $15,000$ time steps. We use the first $9,000$ samples of the time series as training set to learn the parameters of the RNN models. The next $3,000$ samples of the data are used as validation set and the forecast accuracy achieved by the RNNs on this second dataset is used to tune the hyperparameters of the models. The final model performance is evaluated on a test set, corresponding to the last $3,000$ samples of the values in the time series.

As the MG time series doesn't have missing values originally, we manually inject artificial windows of missing data, according to the following three policies. First, each missing window covers 50 consecutive time steps. This reflects a common phenomenon observed in practice, that missing values very often tend to be observed consecutively over a certain period of time. Secondly, the windows with length 50 are randomly introduced, so that there is no specific missing pattern observed. Lastly, the total number of time samples in all missing windows does not exceed 30 % of the whole time series. For example, training set which have $9,000$ time steps in total, includes $2,700$ missing values in the sequence.

9.1.2 Real World Load Time Series : GEFCom2012 Electricity Load

The real world time series that we analyze is the electricity consumption from the Global Energy Forecasting Competition 2012 (GEFCom 2012) [2]. The GEFCom 2012 dataset consists of 4.5 years (1, January, 2004 - 30, June, 2008) of hourly electricity load collected from a US energy supplier. The dataset comprises time series of consumption measurements, from 20 different feeders in the same geographical area. The values in each time series represent the average hourly load, which varies from $10,000$kWh to $200,000$kWh. For the forecasting task, we extract a aggregated time series from the electricity consumption, which is the sum of time series from 20 different feeders.

To study the seasonality in the aggregated time series, we compute the autocorrelation function, which is depicted as the gray line in Figure 9.1 ⟨Right⟩. From the small subplot in top-right part of the figure, relative to a small segment of the time series, it emerges a strong seasonal pattern every 24 hours. By applying a seasonal differentiation with lag 24 the main seasonal pattern is removed, as we can see from the autocorrelation function of the differentiated time series, depicted as a black line in the figure. After differentiation, the autocorrelation becomes close to zero after the first lags and, therefore, we can exclude the presence of a second, strong seasonal pattern (e.g. a weekly pattern) [4].

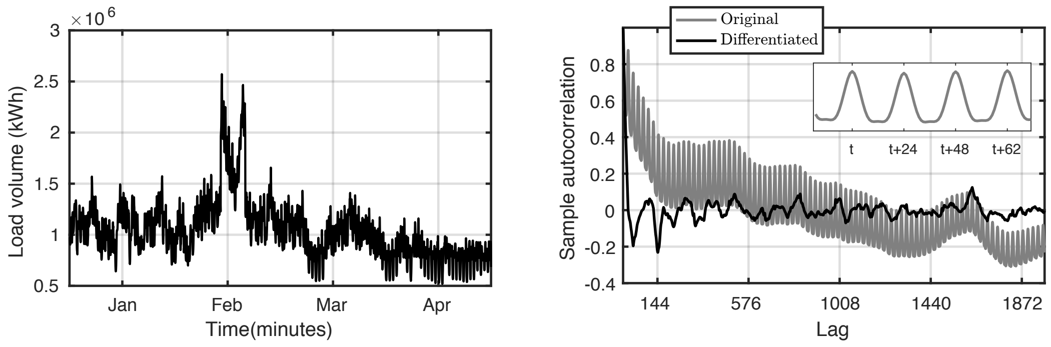


FIGURE 9.1: ⟨Left⟩ load profile in kilowatt-hour (kWh) of the aggregated electricity consumption registered in the first 4 months of activity in 2006, from the GEFCom 2012 dataset. The sampling time in the time series is 1 hour. ⟨Right⟩ the autocorrelation functions of the GEFCom time series before (gray line) and after (black line) a seasonal differentiation at lag 24. The small subplot on the top-right part of the figure reports a magnified version of the autocorrelation function before differentiation at lag $t = 200$. Source : Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis, Bianchi et al. [4]

There are eight time windows with length of one week where data are missing in GEFCom 2012 sequence, located the year of 2005 and 2006. Those are depicted in Figure

9.2. To be sure that missing data appear, at the same time, in the training, validation and test set, those are constructed as follows.

- Training set with 4, 416 time steps (184 days), starts 1 o'clock of 1, June, 2015 to 24 o'clock of 1, December, 2005.
- Validation set with 888 time steps (37 days), starts 1 o'clock of 4, December, 2015 to 24 o'clock of 9, January, 2006.
- Test set with 888 time steps (37 days), starts 1 o'clock of 22, January, 2006 to 24 o'clock of 27, February, 2006.

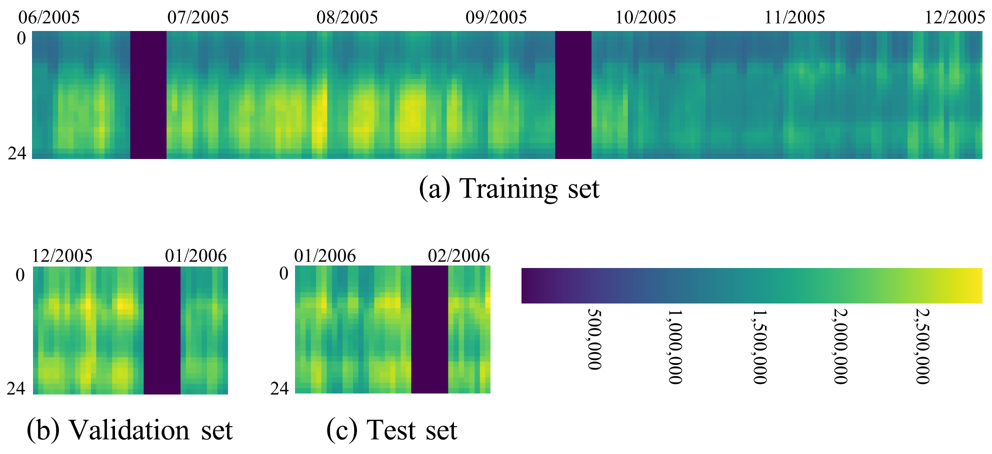


FIGURE 9.2: GEFCom 2012 dataset. Each dark strip represents the missing windows with duration of one week.

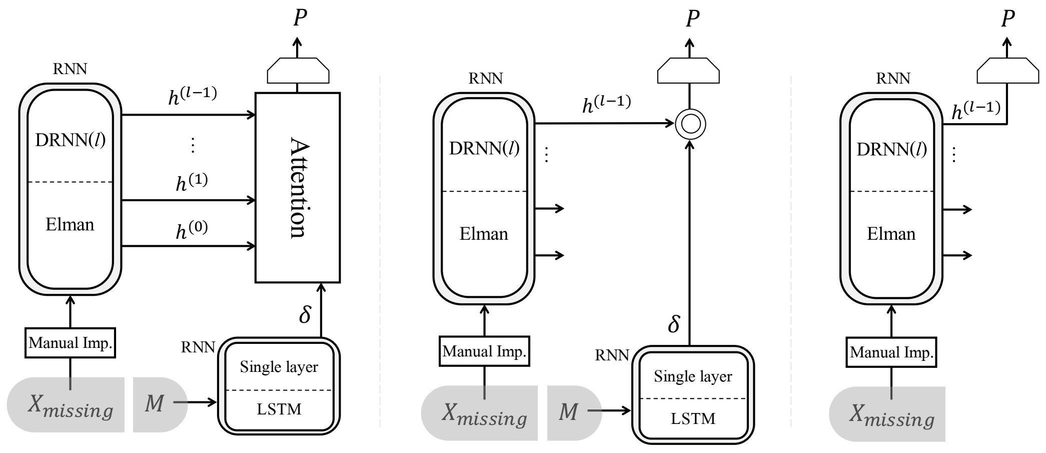
Two preprocessing operations that are applied to the GEFCom dataset are a seasonal differencing at lag 24 and a z-score standardization. The forecast time interval t_f is set to 24 with GEFCom 2012 dataset because the time series, after differencing at lag 24, have the first zero autocorrelation after 24 time steps.

9.2 Experimental Setup

The optimization of the parameters of the network is performed by means of gradient descent, using as training objective to be minimized the MSE of the entire time series. As optimization algorithm, Nesterov momentum [136] with momentum $\theta = 0.9$ is used for the synthetic time series and ADAM [137] optimizer is used for GEFCom 2012 dataset.

The particular optimizer is chosen based on the fact that the loss curve on training samples decreases monotonically as the number of iterations increases.

9.2.1 Model Comparison



(a) DRNN(l) with attention (b) DRNN(l) with time (c) DRNN(l)

FIGURE 9.3: Schema of DRNN(l) models. (a) DRNN(l) with modified attention ; (b) DRNN(l) with time ; (c) DRNN(l). Every model has input with missing values $\mathbf{X}_{missing}$. The effect of modified attention mechanism is compared by the model (a) and (b), where model (b) concatenates the state output of two RNNs. Model (c) are also suggested to see the effect of binary mask towards DRNN(l) by comparing with model (b). $\mathbf{M}$ and $\mathbf{P}$ represent binary mask and forecast respectively.

The primary purpose of the experiments is to investigate if the modified attention mechanism improves the forecasting accuracy under the DRNN structure. Five different models, including two baseline models, are compared in terms of the forecast accuracy achieved on the test set.

The principal models of interest, based on the DRNN architecture, are depicted in Figure 9.3. Model (a) and (b) are taken into account to compare the performance of modified attention mechanism. Model (c) is also tested to investigate the effect of the external RNN by comparing with model (b). All models based on DRNN share the same configuration of hyperparameters, including the number of layers and the number of neurons per layer.

Two base line models, GRU and ARIMA, are introduced to compare the forecast accuracy with the proposed DRNN-based models. As baseline among the different machine learning approaches, the conventional GRU RNN was selected because it can learn time dependencies in a comparable way to LSTM, but it uses a simpler architecture than LSTM. As baseline for the statistical approach, ARIMA is chosen. The order of ARIMA(p, d, q) is carefully selected by following commonly used practices for the design of the coefficients[1].

9.2.2 Hyperparameters

To identify an optimal configuration for the forecasting task, we evaluate each RNN by the forecast accuracy according to the specific values of the hyperparameters. In the following, details of configuration searching method, proposed by Bianchi et al. [4], are discussed.

We opted for a random search as it can find more accurate results than a grid search, when the same number of configurations are evaluated [138]. Specifically, different configuration of the hyperparameters are randomly selected from admissible intervals. Table 9.1 reports the optimal configurations of each model for the forecasting tasks.

For DRNN models, the dilation at layer $l = 1, 2, \cdots, L$, $d^{(l)}$ is defined as a power of 2, denoted in Equation 9.3.

$$d^{(l)} = 2^{l-1} \tag{9.3}$$

Dataset	Network	L	t_f	k_2	k_1	N_h	N_δ	OPT	θ	μ	β
MG	DRNN+att	5	12	1024	512	20	10	Nesterov	0.9	$3.34e^{-4}$	$3.70e^{-7}$
	DRNN+time	5	12	1024	512	20	10	Nesterov	0.9	$3.34e^{-4}$	$3.70e^{-7}$
	DRNN	5	12	1024	512	20		Nesterov	0.9	$3.34e^{-4}$	$3.70e^{-7}$
	GRU	1	12	64	32	20		Nesterov	0.9	$3.34e^{-4}$	$3.70e^{-7}$
	ARIMA(3,0,0)		12								
GEFCom	DRNN+att	8	24	256	128	10	5	Adam		$2.00e^{-4}$	$5.00e^{-3}$
	DRNN+time	8	24	256	128	10	5	Adam		$2.00e^{-4}$	$5.00e^{-3}$
	DRNN	8	24	256	128	10		Adam		$2.00e^{-4}$	$5.00e^{-3}$
	GRU	1	24	64	32	20		Adam		$3.34e^{-4}$	$2.00e^{-3}$
	ARIMA$(2,0,1)$		24								

TABLE 9.1: Optimal RNNs configurations for the synthetic time series. The acronyms in the table are: L – number of hidden layers; t_f – the forecast time interval; k_2 – number of time step the gradient is propagated back in BPTT (length of backward pass); k_1 – number of new time steps processed forward before computing the BPTT (length of forward pass); N_h – number of nodes in the hidden layer; N_δ – number of nodes in the missing history $\boldsymbol{\delta}_t$; OPT – gradient descent strategy; θ – Momentum; μ – learning rate; β – L$_2$ regularization parameter;

[1]https://people.duke.edu/~rnau/arimrule.htm

The number of layer of DRNN, L is selected according to criteria that the longest dilation should be shorter than the shortest width of missing windows. For example, as the shortest width of a missing window used for MG dataset is 50, so that the longest dilation $d^{(L)}$ should be an element within the subset $d^{(L)} \in \{1, 2, 4, 8, 16, 32\}$, which corresponds to having a total number of layers $L \in \{1, 2, 3, 4, 5, 6\}$ respectively. For GEFCom 2012 dataset, the width of a missing window is 168 so that the longest dilation is upperbounded to $d^{(L)} \in \{1, 2, 4, 8, 16, 32, 64, 128\}$ and the corresponding layer $L \in \{1, 2, 3, 4, 5, 6, 7, 8\}$. For the experiment with MG set, we set $L = 5$ so that $d^{(5)} = 16$ and with GEFCom 2012 set, we set $L = 8$ and $d^{(8)} = 128$.

According to what discussed in Chapter 4 about the selection of the extent of the backward and forward pass, the length of backward pass is defined as $k_2 = 2^b$, where b is randomly chosen according to criteria, $b \in \{L, L+1, L+2, L+3, L+4, L+5\}$ to make k_2 longer than the longest dilation $d^{(L)} = 2^{(L-1)}$. The length of forward pass k_1 is set to a half of k_2. Regarding the number of hidden units N_h in the recurrent hidden layer, we randomly choose one value from the set $\{5, 10, 15, 20\}$ to avoid overfitting. The number of hidden units for binary mask N_δ in the external recurrent hidden layer, we manually set to have quotient of N_h divided by 2. We define learning rate, $\mu = 10^c$, where c is sampled from the uniform interval $c \in [-4, -2]$. For the L_2 regularization coefficient β, we sample from $[0, 0.1]$, an interval containing values commonly assigned to this hyperparameter in RNNs [139].

Once the candidates of the possible hyperparameter configurations are selected, performances are evaluated on the validation set, after having trained the network for $1,000$ epochs. Based on the performance, hyperparameters are fine tuned to figure out the optimal configuration. After the optimal configuration of the hyperparameters has been identified, we train each model for $5,000$ epochs three times using random and independent initializations of the network parameters and compute the forecast accuracy on the test set. We report the highest forecast accuracy on the test set among the three time training session.

9.3 Results

9.3.1 Synthetic Dataset

9.3.1.1 Forecast Accuracy

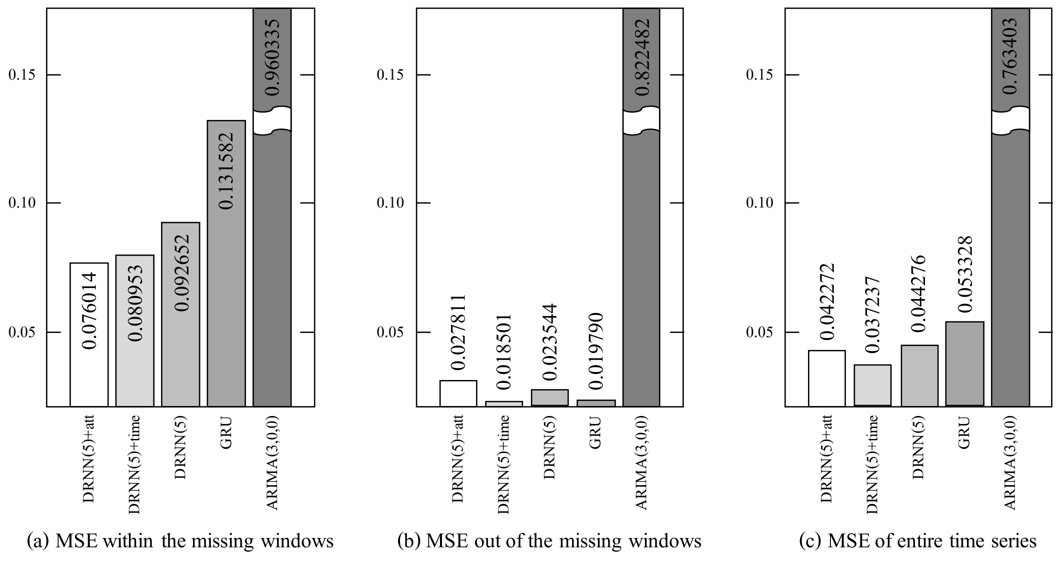


FIGURE 9.4: MSE comparison for RNN models with MG set. (a) MSE within the missing windows ; (b) MSE out of the missing windows ; (c) MSE of entire time series. All subplots show that the RNN based models outperform the ARIMA model.

In Figure 9.4 we report the forecast accuracy of MG test set with respect to MSE obtained from each model. To show the difference between the prediction performance of the different models with or without missing values in the input, the MSE presented in each subplot is computed on (a) imputed inputs (within the missing windows); (b) observed inputs (out of the missing windows); and (c) entire time series. In the figure (a), ⟨DRNN(5) with attention⟩ model obtains the lowest MSE(0.076) and the model ⟨DRNN(5) with time⟩ follows by (MSE:0.081), where the models are illustrated in Figure 9.3 (a) and (b) respectively. Meanwhile, in Figure 9.4 (b), ⟨DRNN(5) with time⟩ model achieves the lowest MSE(0.018) while ⟨DRNN(5) with attention⟩ model gives the greatest MSE among RNN based models. For the MSE of the entire time series shown in Figure 9.4 (c), ⟨DRNN(5) with time⟩ have the lowest MSE(0.037) and ⟨DRNN(5) with attention⟩ model follows by (0.042).

The results in Figure 9.4 reveal that,

- The RNN based models outperform the ARIMA model in terms of the forecasting accuracy. All the plots in Figure 9.4 show that the RNNs achieve outstanding forecast accuracy than ARIMA in the forecasting task. The result supports the hypothesis that the RNNs are more suitable models than the conventional ARIMA for the forecasting task. On the other hand, it is still doubtful whether the ARIMA model is suitable for comparison to RNN models when the input data have missing values since the forecasting accuracy of ARIMA is too low. Figure 9.5 ⟨e⟩ depicts that ARIMA(3,0,0) model fails to forecast with imputed inputs and always produces as output just a constant value. However, we include the ARIMA model for a comparison because it is still widely used as a baseline of the forecasting task.

- The dilated structure contributes to the accurate forecasting within the missing windows. In Figure 9.4 (a), DRNN based models achieve lower MSE than GRU. Let's compare the two models, ⟨DRNN(5)⟩ and GRU, which don't have external RNNs utilizing binary mask m_t as a secondary input. Obviously, ⟨DRNN(5)⟩ (MSE:0.093) brings higher accuracy than GRU($\mathrm{MSE}: 0.132$). The result supports the hypothesis that utilizing longer dilations can improve the forecasting accuracy within the missing windows, rather than utilizing only recurrent connections with length 1. Figure 9.5 shows each forecast of the models for the same period of the test set. Comparing the forecast in the missing windows with ⟨DRNN(5)⟩ (the third from the top) and the baseline GRU (the fourth from the top), the forecast (red line) of GRU gets misaligned from the target (green line) in the earlier time steps than ⟨DRNN(5)⟩. It implies that DRNN structure can capture longer term dependencies than GRU.

- External RNNs utilizing binary mask m_t as a secondary input contribute to the accurate forecasting. Comparing ⟨DRNN(5) with time⟩ model to ⟨DRNN(5)⟩, we figure out that the external RNN utilizing binary mask m_t as an auxiliary input improves the forecasting accuracy. By looking at all the subplots in Figure 9.4, we can observe that the MSEs of ⟨DRNN(5) with time⟩ are lower than ⟨DRNN(5)⟩.

- The modified attention mechanism contributes to the accurate forecasting within the missing windows. Figure 9.4 (a) shows that ⟨DRNN(5) with attention⟩ model achieves the highest forecast accuracy (MSE:0.076) within the missing windows of the sequence but ⟨DRNN(5) with time⟩ also gives comparable accuracy (MSE:0.080).

As the difference between two models, ⟨DRNN(5) with attention⟩ and ⟨DRNN(5) with time⟩, is the presence of attention mechanism, we anticipate that the improvement of the forecasting accuracy within the missing windows is an effect of the attention mechanism. However, since the gap in MSE between the two models is not large enough, there would be an objection that it is uncertain to claim that the attention mechanism contributes to the improvement. Further experiments can clarify the conflicting opinions. For example, the performance of the modified attention mechanism can be examined by increasing the ratio of the missingness of MG set, that is set to 30 % in this experiment. Meanwhile, Figure 9.4 (b) illustrates that ⟨DRNN(5) with time⟩ model brings a higher accuracy(MSE:0.019) than ⟨DRNN(5) with attention⟩ (MSE:0.028) for the observed inputs. We infer that, depending on the nature of the time series, it might be more accurate for the forecast to utilize the highest layer of DRNN consistently rather than exploit multiple layers using attention weights when the inputs of the network are observed values. As shown in Figure 9.5, out of the missing windows, MG set exhibits a smooth and quasi-periodic pattern which is not a challenging task for the network to learn even with single layer. In Figure 9.4 (b), the fact that MSE of GRU(0.020) is comparably low to MSE of ⟨DRNN(5) with time⟩(0.019) supports the inference. Accordingly, we can improve the attention mechanism such that the attention mechanism works only when the inputs are missing values while the RNNs utilize the longest dilation when the inputs are observed values.

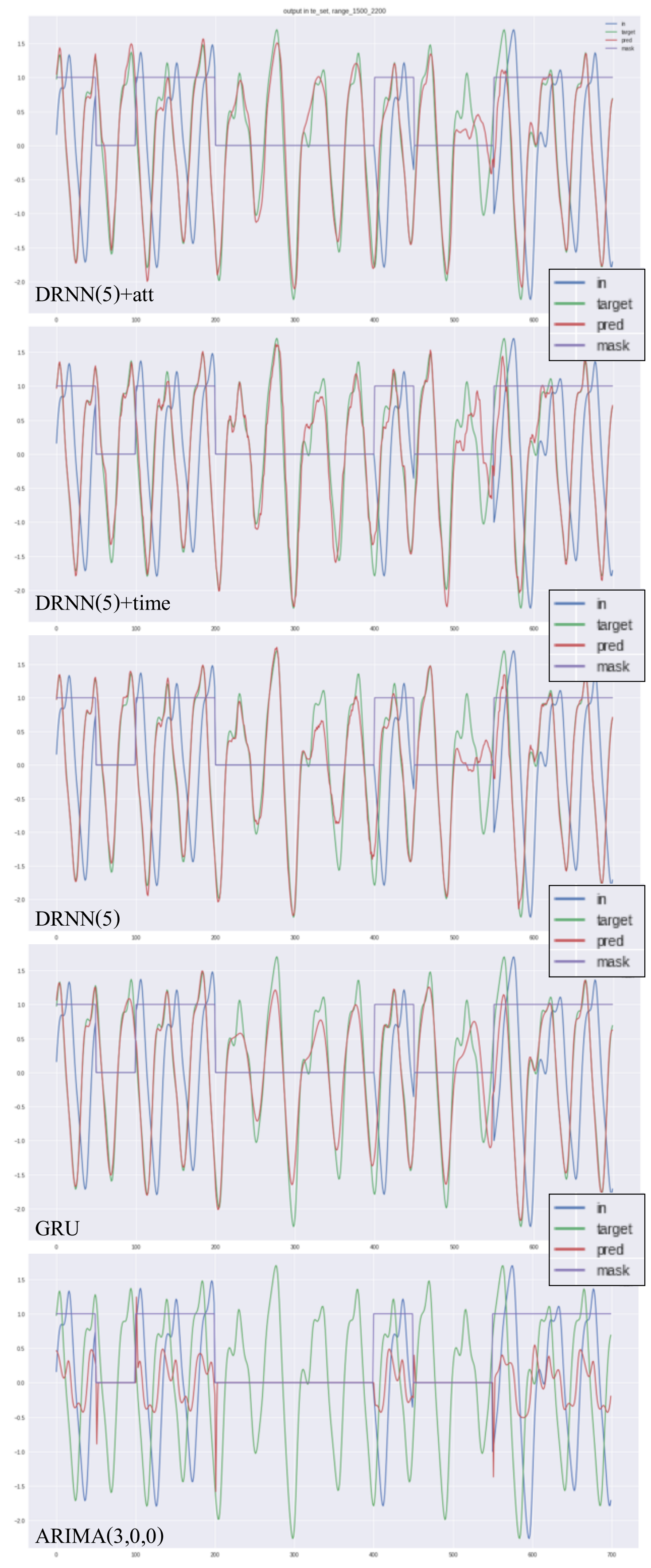


FIGURE 9.5: Forecast plots over a period of time step in test set of MG, depending on the 5 different models. Red line depicts the forecast values while green line depicts the target, that the forecast aims to reach. Blue and violet lines show input and binary mask, which indicates if the input data are missing. From the top, ⟨a⟩ DRNN(5) with modified attention; ⟨b⟩ DRNN(5) with external RNN with LSTM cell having binary mask as input; ⟨c⟩ DRNN(5). ⟨d⟩ GRU ; ⟨e⟩ ARIMA(3,0,0).

9.3.1.2 Change of Attention Weight $\alpha_t^{(l)}$ around Missing Windows

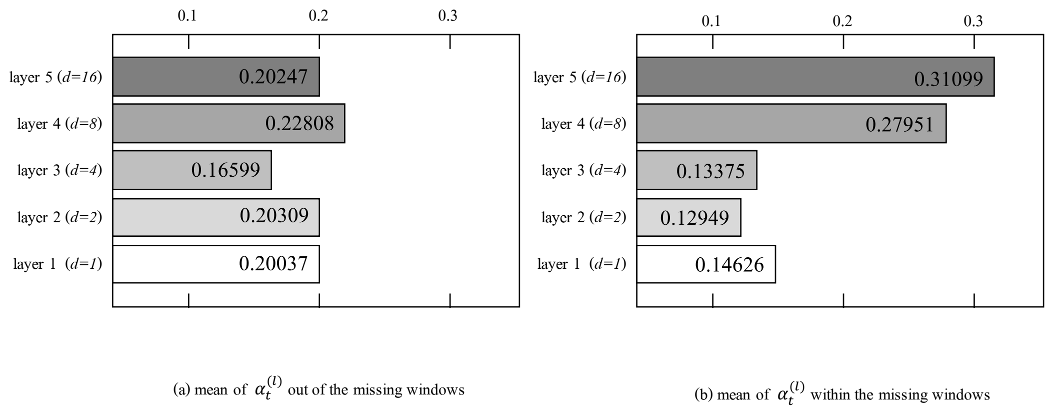


FIGURE 9.6: Change of attention weights $\alpha_t^{(l)}$ depending on the input. The bar plot (a) depicts the weights which are derived by the observed values. The second bar plot (b) depicts the learned weights which are returned in presence of missing values. The weight for layer 4 (dilation 8) and 5 (dilation 16) increases while the weights for layer 1, 2, and 3 decrease when imputed values are fed into the network. Attention weights induce the RNN to exploit information from the higher layers which are expected to be less biased by the imputed values.

An important sanity check for the proposed model ⟨DRNN(5) with attention⟩ (model (a) in Figure 9.3), consists in verifying the change of attention weights $\alpha_t^{(l)}$ in the different layers, when the input data are missing. To investigate the results, we divide the attention weights $\alpha_t^{(l)}$ of each layer according to whether the input is imputed(See Figure 9.6 (a)) or observed(See Figure 9.6 (b)), and compare the change of mean values under each condition. As the attention weights $\alpha_t^{(l)}$ are indicators of which layer the RNNs exploit for the forecast, the changes between the two cases can be an evidence of the hypothesis that the networks will consider the layer with longer dilation as more reliable towards the successive imputed values within the missing windows.

Comparing subplots in Figure 9.6, the average of attention weights of layer 4 (dilation $d = 8$) and 5 (dilation $d = 16$) significantly increase while the weights of layer 1, 2 and 3 decrease within the missing windows, where imputed values are fed into the network. It means that the RNNs focus the attention more on the longer dilation for a better forecast towards the imputed inputs because the attention is finite resource, which sums to one. Figure 9.7 shows the changes of attention weights within and out of the missing windows in the same period of the test set of Figure 9.5. In the missing windows where light blue line indicates zero, the the weights with respect to lower layers decreases while weights of higher layers increase.

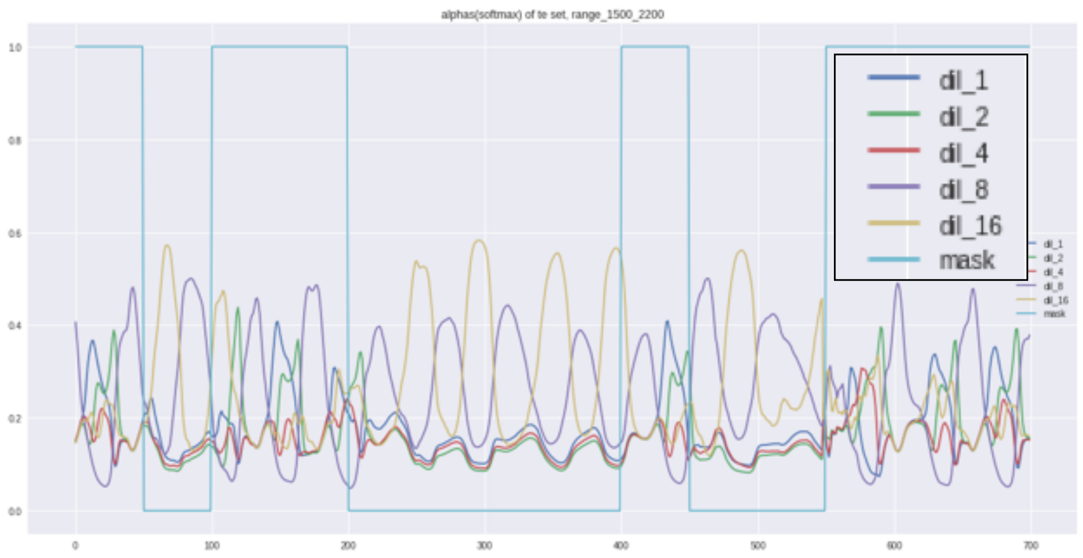


FIGURE 9.7: Plot of attention weights $\alpha_t^{(l)}$ over a period of time step. Rectangular line depicts binary mask m_t, displaying 0 if an input value is imputed and 1 otherwise. Yellow and purple lines denoting the attention weight $\alpha_t^{(4)}$ and $\alpha_t^{(5)}$ interact with rectangular line, showing an increase when rectangular line is 0 and a decrease on the other case.

9.3.2 Results on Real World Dataset

9.3.2.1 Forecast Accuracy

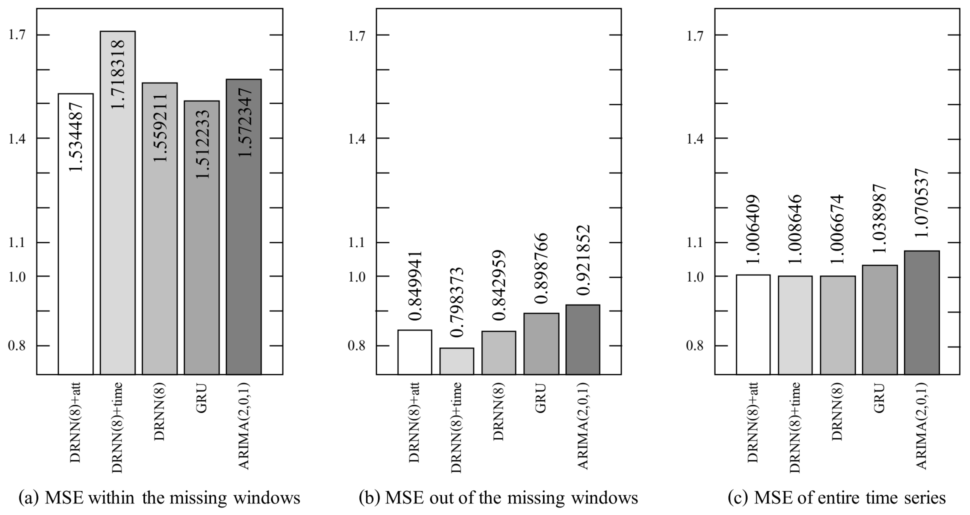


FIGURE 9.8: MSE comparison for RNN models with GEFCom 2012 set. (a) MSE within the missing windows ; (b) MSE out of the missing windows ; (c) MSE of entire time series.

In Figure 9.8 we report the forecast accuracy of GEFCom 2012 test set with respect to MSE obtained from each model. Each subplot of 9.8 represents the MSE according to conditions as given in Figure 9.8. Overall, MSEs of GEFCom 2012 set indicate higher values than the MSEs of MG dataset in Figure 9.4.

In the figure (a), all the models except ⟨DRNN(8) with time⟩ show comparable performance, and GRU(MSE:1.512) is slightly better than the others. Among DRNN models, ⟨DRNN(8) with attention⟩ brings the lowest MSE(MSE:1.534). Figure 9.8 (b), ⟨DRNN(8) with time⟩ model achieves the lowest MSE(0.798) and other DRNN based models, ⟨DRNN(8)⟩(0.843) and ⟨DRNN(8) with attention⟩(0.850), follow by. For the MSE of the entire time series shown in Figure 9.8 (c), DRNN based models indicate similar MSEs, achieving a higher accuracy than two baselines.

The results in Figure 9.8 reveal that,

- Compared with the experiment with MG set, all the RNN based models result in lower forecasting accuracies with GEFCom 2012 set. It is expected by the reasons. First, the task is more difficult. For MG set, the task is designed to forecast

$t_{f,MG} = 12$ time steps ahead, while the task with GEFCom 2012 set is to forecast $t_{f,GEFCom} = 24$ time steps ahead. Secondly, time series has an unpredictable pattern as shown in Figure 9.9, unlike MG set which has a smooth, quasi-periodic pattern. Prior to the experiment, we perform preprocessing to remove seasonality by applying seasonal differencing at lag 24 on the raw version of GEFCom 2012 time series. The restored pattern from the differencing is given in Figure 9.10. Another reason is overfitting. The training with MG set doesn't show the overfitting until 5000 epochs, but the GEFCom 2012 set exhibits overfitting around 500 epochs. Two remedies, L2-regularization and hidden size N_h reduction, are applied in order to prevent the overfit but no major improvement was achieved.

- By comparing ⟨DRNN(8)⟩ and GRU, dilated structure achieves lower MSE than GRU for entire time series shown in Figure 9.8 (b) and (c). We argue that the dilations can contribute to the accurate forecasting with GEFCom 2012 set as they do with MG set. As in the previous experiment in Figure 9.4 (c), ⟨DRNN(8)⟩ shows higher accuracy than GRU in the entire series shown in Figure 9.8 (c). However, while MG set gives higher accuracy within the missing windows, GEFCom 2012 set shows higher accuracy with imputed inputs. This inconsistency should be verified through additional experiments.

- In a real-world scenario, due to the irregular patterns in the time series, using exogenous variables could greatly improve the forecast accuracy and obtain a more accurate training of the model. In the case of electricity load forecasting, we can improve the accuracy in future research by performing a comprehensive analysis including external variables such as weather effects, calendar effects, and so on.

- Even in this case, the intuition behind the usage of attention mechanism for missing data is confirmed. Indeed, we observe a consistent result regarding modified attention mechanism. Comparing ⟨DRNN(8) with attention⟩ to ⟨DRNN(8) with time⟩, ⟨DRNN(8) with attention⟩ shows lower MSE within the missing windows, while ⟨DRNN(8) with time⟩, gives lower MSE out of the missing windows as observed in the experiment with MG set.

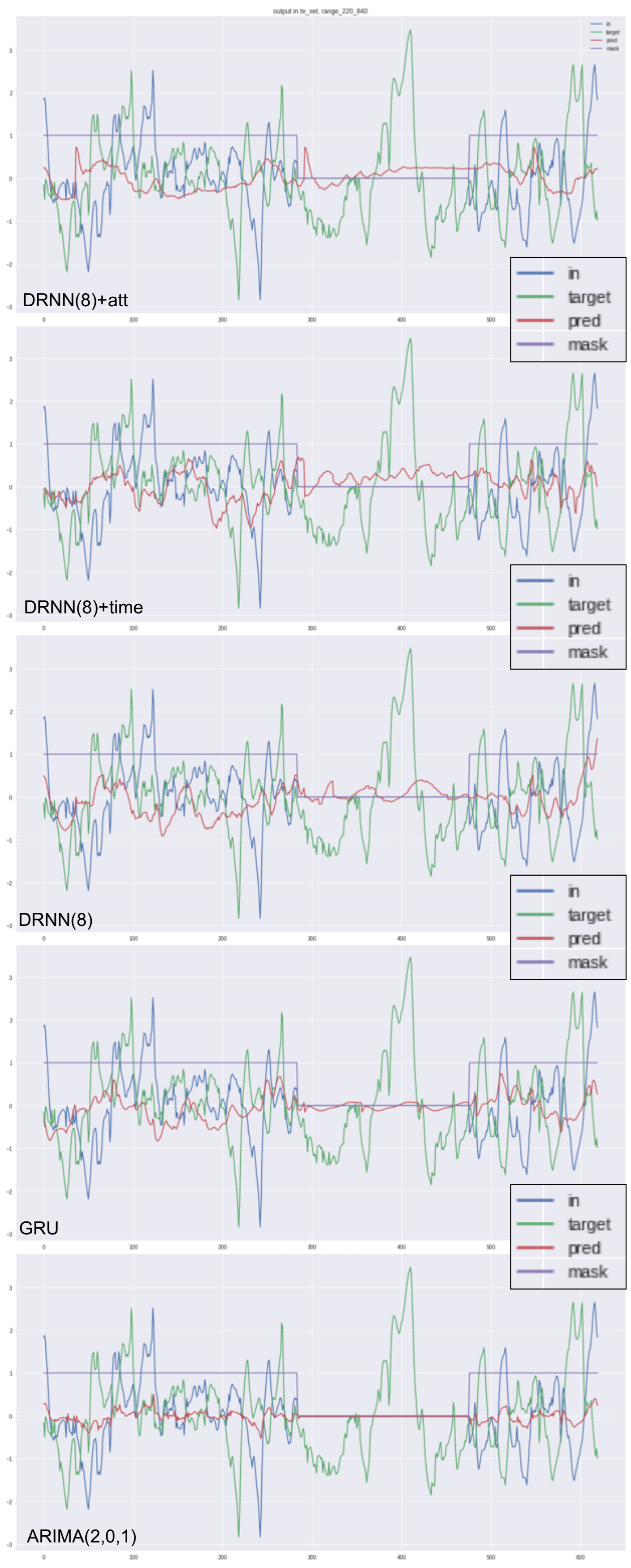


FIGURE 9.9: Forecast plots over a period of time step in test set of GEFCom 2012 after the seasonal differencing at lag 24, depending on the 5 different models. Red line depicts the forecast values while green line depicts the target, that the forecast aims to reach. Blue and violet lines show input and binary mask. From the top, ⟨a⟩ DRNN(8) with modified attention ; ⟨b⟩ DRNN(8) with external RNN with LSTM cell having binary mask as input; ⟨c⟩ DRNN(8). ⟨d⟩ GRU ; ⟨e⟩ ARIMA(2,0,1). Overall, all the models don't make an accurate forecast with GEFCom 2012 set.

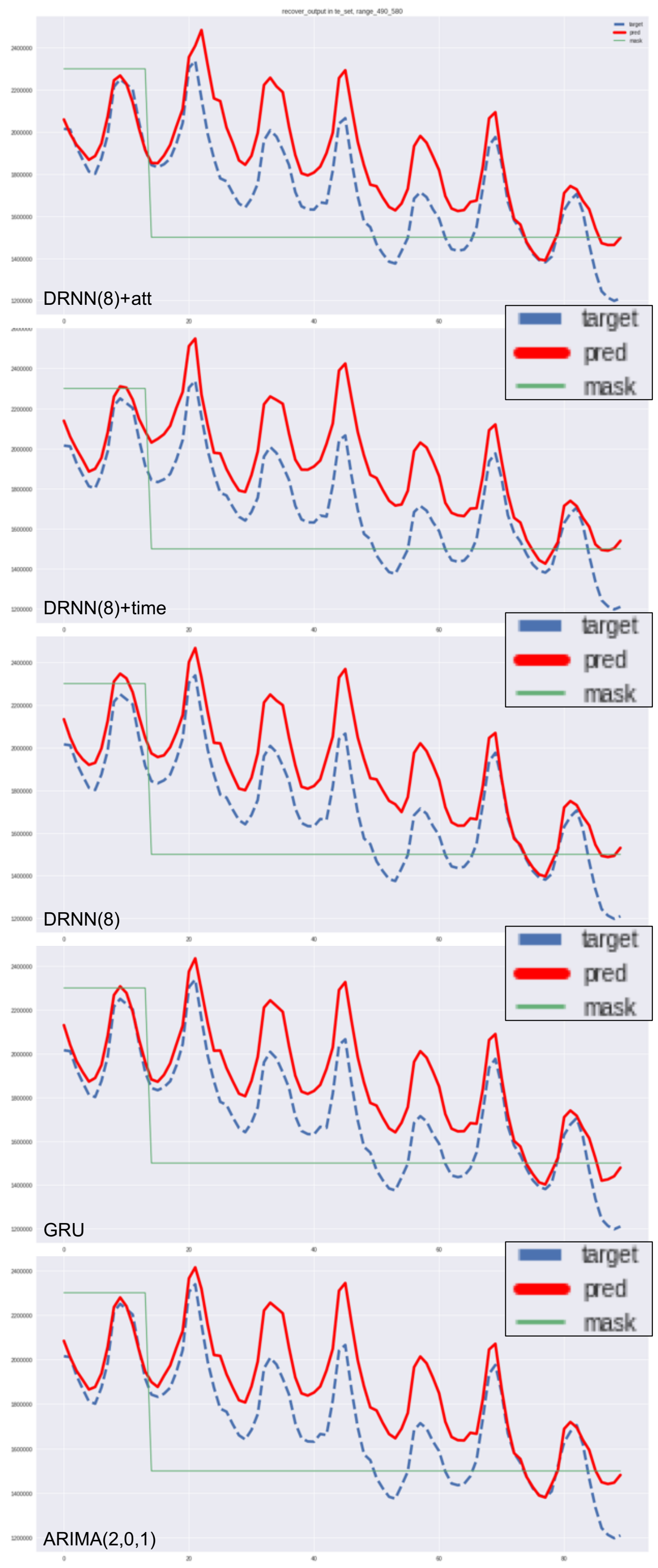


FIGURE 9.10: Forecast plots over a period of time step in test set of GEFCom 2012 after restoring from the seasonal differencing at lag 24, depending on the 5 different models. Red line depicts the forecast values while blue line depicts the target, that the forecast aims to reach. Green line shows the binary mask. From the top, ⟨a⟩ DRNN(8) with modified attention ; ⟨b⟩ DRNN(8) with external RNN with LSTM cell having binary mask as input; ⟨c⟩ DRNN(8). ⟨d⟩ GRU ; ⟨e⟩ ARIMA(2,0,1).

9.3.2.2 Change of Attention Weight $\alpha_t^{(l)}$ around Missing Windows

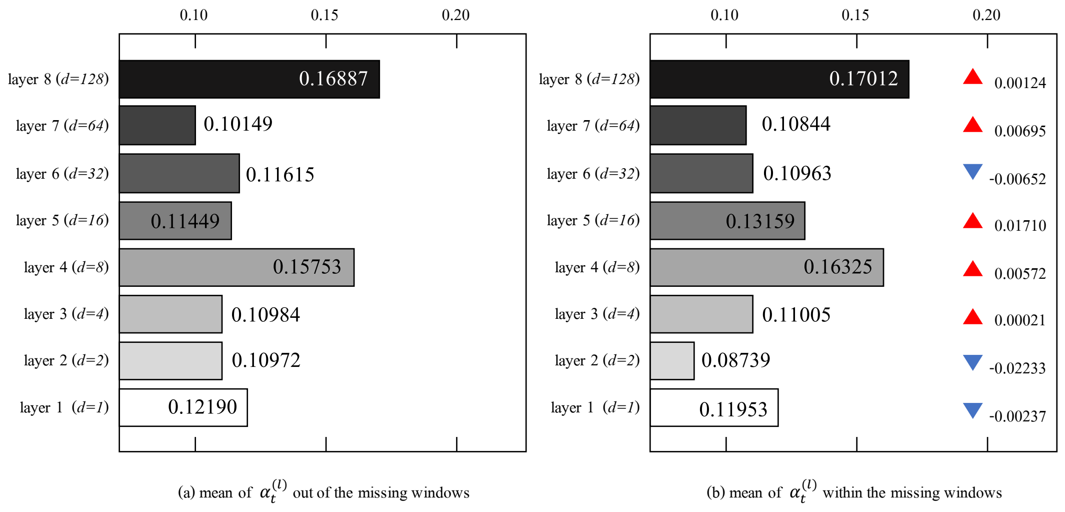


FIGURE 9.11: Change of attention weights $\alpha_t^{(l)}$ depending on the input. Graph (a) depicts the weights which are derived by the observed values. Graph (b) depicts the weights which are derived by the imputed values. The change is less striking than in the experiment with MG set but we still can observe an increase at higher layers and an decrease at lower layers.

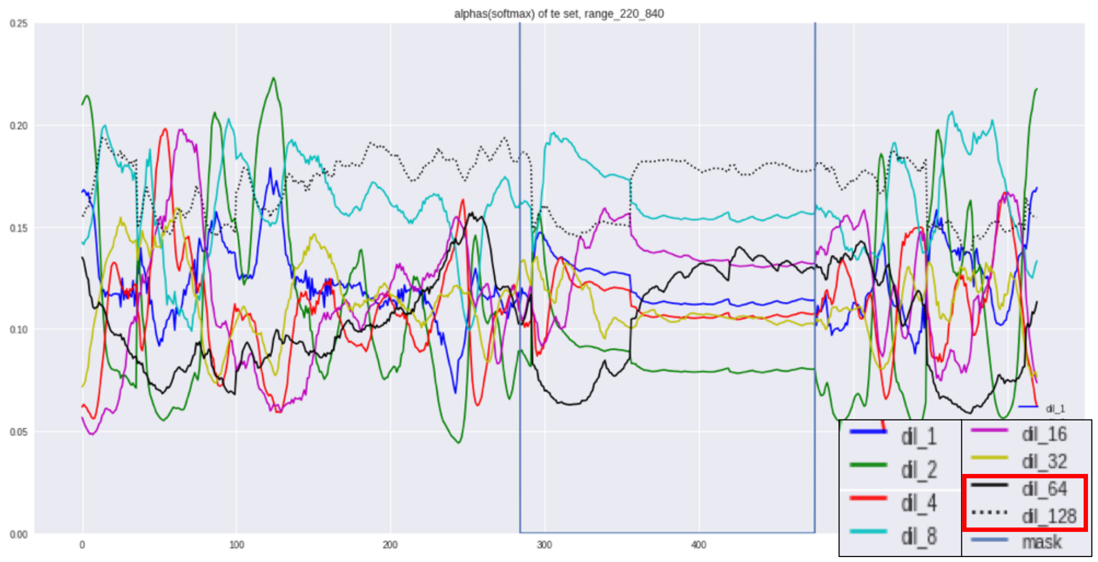


FIGURE 9.12: In the beginning of the missing window between the blue vertical lines, the weights with dilation $d = \{64, 128\}$ increase, while others turn to decrease. Around 400 time steps, all the weights turn to be flat.

The difference in the mean values of the attention weights $\alpha_t^{(l)}$ of the two groups in Figure 9.11 (a) and (b), is less striking than in the previous experiment with MG set shown in Figure 9.6 but we still can observe an increase at higher layers and an decrease at lower layers. Furthermore, we detect an interesting change within the missing window from Figure 9.12. Within the missing window, the network appears to trust the information more of the longer dilations as the time passes, because the weights with dilation $d = \{64, 128\}$ increase, while others turn to decrease. The observation indirectly indicates that the network uses attention to find more reliable information on its own, although the attention mechanism has not shown a definite improvement in the forecasting performance.

9.4 Discussions

Through the experiments, we compare the forecast accuracy of the proposed model $\langle$DRNN(l) with attention$\rangle$ and its variants, $\langle$DRNN(l) with time$\rangle$ and $\langle$DRNN(l)$\rangle$, to two baseline models (ARIMA and GRU) with synthetic and real world time series respectively. We are able to observe the improvement of the forecast accuracy from the presence of dilated structure by comparing to GRU containing recurrent connection with length 1. Also, we confirm that the modified attention mechanism improves the forecast accuracy within the missing windows. From the experiments with different dataset, consistency in the change of attention weights is observed, depending on whether the input is missing or observed. This indicates empirical evidence that the proposed mechanism is behaving as expected. Therefore, we argue that with further research effort and a more detailed search of the hyperparameters, it would be able to achieve much higher performance by means of the proposed procedure. In addition, real world data, such as electricity load time series, should be analyzed with exogenous variables in order to achieve higher forecasting accuracy since they are expected to have considerable correlation.

Chapter 10

Conclusions and Future Directions

10.1 Conclusions

In this thesis, we first reviewed various types of RNN for the time series forecasting task, explaining their internal mechanisms, discussing their properties and the procedures for the training. Concretely, one of the contributions of the thesis is that we provide a formal description of truncated BPTT(k_2, k_1), and explain how to construct mini-batches of the training dataset for the forecasting tasks with RNNs. As RNNs have been mostly used for classification, there is a lack of knowledge of how to prepare the data and train them for the forecasting tasks. Also, we proposed a novel model as a main contribution that utilize dilated RNN(DRNN) and a modified attention mechanism, focusing on the problem of STLF with missing data. The proposed model showed significant improvement with synthetic dataset in the forecasting accuracy with respect to the baselines, such as ARIMA and GRU. From the experiment with real world dataset, we observed interesting phenomena which is worth further research although we couldn't reach the same level of improvement in terms of the forecast accuracy over the baseline models, as in the synthetic data. The proposed models and findings presented in the thesis would contribute to a more accurate time series forecast and eventually help to improve efficiency of the resource management. In industrial fields, an accurate time series forecasting plays an irreplaceable role in managing and distributing their resources.

However, missing values, usually missing windows, obstruct the accurate forecasting. As conventional forecasting models such as ARIMA are not able to forecast with missing inputs while RNNs have shown robust performance on the time series forecast, it is natural that RNNs emerge as an alternative for the forecast tasks with missing inputs.

However, in comparison with the importance to the industries, in academia, there is a limited study of time series forecasting using RNNs with missing inputs. The significant performance of the proposed model identified by the synthetic dataset, suggests that the thesis is a preliminary study of time series forecasting with missing data, which is expected to become a promising research topic due to the possibility of various application.

10.2 Future Directions

In order to improve the forecast accuracy, we suggest five different directions for the future work.

First, the optimal configurations of the network, such as the number of layers l, length of backward pass k_2, the number of nodes in the hidden layer N_h, the number of nodes in the missing history N_δ, and so forth, should be investigated more in detail and in relation to the characteristics of the time series and missing patterns. During the training, we found out that the results of the forecast vary depending on the configurations of the network.

Secondly, we can also change the combination of dilations. For example, DRNNs can learn the long-term dependencies of the raw data (by the longest dilation) first to outline the data structure and then capture shorter-term dependencies by shorter dilations which are exponentially decreasing over layers, unlike the configuration in this thesis. we can compare the performance between exponentially increasing and exponentially decreasing dilations in terms of the speed of training and forecast accuracy.

Third, manual imputation techniques also need further study. In this thesis, we restrict the imputation techniques by mean substitution. But we need to make sure if similar performance is achieved with last value carried forward (LVCF) technique. In this case, GRU-D [90] model can be one of the baselines because the trainable decay of GRU-D γ_x consider the missing input as a value between mean substitution and LVCF.

Fourthly, to foster trust on the results, especially with real world dataset, the forecast accuracy should be improved. In the thesis we restricted to use only univariate time series for the forecast. But exogenous variables can be tested by measuring the improvement of the performance in the future work.

Last but not least, the result that $\langle$DRNN(l) with time$\rangle$ model brings higher accuracy out of the missing windows inspires the attention mechanism to be developed such that the attention mechanism works only when the inputs are missing values while the RNNs utilize the longest dilation when the inputs are observed values.

Appendix A

Support Vector Machine

A support vector machines (SVM)[1] can be explained by a linear classifier in a kernel space, induced by a usually non-linear kernel $\phi(w) \cdot \phi(x) = K(w, x)$.

$$\ell_i = g(x_i) = \text{sign}(\phi(w) \cdot \phi(x_i) + b), \tag{A.1}$$

In order to train a SVM, the cost function $\phi^*(w)$ is minimized under the constraints $y_i(\phi(w) \cdot \phi(x_i) + b) \geq 1$.

$$\phi^*(w) = \arg\min_{\phi(w)} \frac{1}{2} \|\phi(w)\|^2$$

The constraints can be included in the previous cost function $\phi^*(w)$ by using the Lagrangian multipliers,

$$L(\phi(w), b, \alpha) = \frac{1}{2}||\phi(w)||^2 - \sum_i \alpha_i(y_i(\phi(w) \cdot \phi(x_i) + b) - 1). \tag{A.2}$$

It follows that the weight vectors become a linear combination of the data points

$$\phi(w) = \sum_i y_i \alpha_i \phi(x_i), \tag{A.3}$$

[1]The equations are referred from the master's thesis of Wickstrøm [56]. Title: Uncertainty Modeling and Interpretability in Convolutional Neural Networks for Polyp Segmentation

and the classifier can be expressed as

$$g(x) = \operatorname{sign}\left(\sum_i y_i \alpha_i \phi(x_i) \cdot \phi(x) + b\right) = \operatorname{sign}\left(\sum_i y_i \alpha_i K(x_i, x) + b\right). \qquad \text{(A.4)}$$

If we substitute (A.4) in (A.2), we obtain the following dual cost function

$$W(\alpha) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j \phi(x_i) \cdot \phi(x_j) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} y_i y_j \alpha_i \alpha_j K(x_i, x_j), \qquad \text{(A.5)}$$

and the optimization now reads

$$\begin{aligned} \hat{\alpha} &= \arg\max_{\alpha} W(\alpha), \\ \text{such that} &\quad \alpha_i \geq 0. \end{aligned} \qquad \text{(A.6)}$$

Once the training is complete, new points are classified directly by applying (A.4).

Appendix B

Optimization techniques : Gradient Descent Algorithms

This appendix[1] describes the gradient descent algorithm and its many variants.

Gradient descent is the most widely used for optimizing neural networks because gradient descent only requires computing the gradients of a network which can be efficient compared to methods that require higher order derivatives to be computed [56].

A common issue for gradient descent algorithms are regions where the cost plateaus before descending further, which lead to gradients close to zero and thus no parameter updates. A typical solution is adding momentum [140] which accelerates the algorithm in the relevant direction. Momentum is included by adding a fraction θ of the gradients of the previous time step, expressed as:

$$v_t = \theta v_{t-1} + \mu \frac{\partial J}{\partial \mathbf{W}_{old}} \tag{B.1}$$

$$\mathbf{W}_{new} = \mathbf{W}_{old} - v_t \tag{B.2}$$

Momentum is often illustrated as a ball rolling down a hill which can traverse flat region as a result of the momentum it gathers while rolling down the hill. However, a ball

[1]The discussion is given in the master's thesis of Wickstrøm [56]. Title: Uncertainty Modeling and Interpretability in Convolutional Neural Networks for Polyp Segmentation

rolling blindly down a hill might overshoot a desired minimum, so to give the ball a sense of direction one could employ a variation of momentum known as Nesterov Momentum [136]. Nesterov Momentum considers $\mathbf{W}^*_{old} = \mathbf{W}_{old} - \theta v_{t-1}$, thus approximating the next position of the parameters. We can implement this procedure by

$$v_t = \theta v_{t-1} + \mu \frac{\partial J}{\partial \mathbf{W}^*_{old}} \tag{B.3}$$

$$\mathbf{W}_{new} = \mathbf{W}_{old} - v_t \tag{B.4}$$

There are a number of recent variations of gradient descent which seek to improve the optimization procedure, such as ADAM [137], Adagrad [141], AdaDelta [142], and RMSprop [143]. In the following we provide the details of ADAM which is a recently proposed algorithm known as the Adaptive Moment Estimation.

ADAM computes an adaptive learning rate for each parameter by storing an exponentially decaying average of past gradients and past squared gradients, defined as:

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \frac{\partial J}{\partial \mathbf{W}^*_{old}} \tag{B.5}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2) \Big(\frac{\partial J}{\partial \mathbf{W}^*_{old}} \Big)^2 \tag{B.6}$$

where m_t is an estimate of the mean of the gradients, v_t is an estimate of the variance of the gradients, β_1 is the decay rate of the estimated mean of the gradients, and β_2 is the decay rate of the estimated variance of the gradients. The authors of ADAM noticed that since m_t and v_t are initialized as vectors of zeros they are biased towards zero. Therefore, they computed bias corrected estimates

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{B.7}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{B.8}$$

which they used update the parameters, in the following way:

$$\mathbf{W}_{new} = \mathbf{W}_{old} - \frac{\mu}{\sqrt{\hat{v}_t + \epsilon}} \hat{m}_t. \tag{B.9}$$

Because ADAM adjusts $\hat{m}_t$ and $\hat{v}_t$ automatically during the training we do not need to tune these hyperparameters manually, which can be a time-consuming a difficult process.

Bibliography

[1] Tao Hong and Shu Fan. Probabilistic electric load forecasting: A tutorial review. *International Journal of Forecasting*, 32(3):914–938, 2016.

[2] Kaggle. GEFCom 2012 global energy forecasting competition 2012. https://www.kaggle.com/c/global-energy-forecasting-competition-2012-load-forecasting, 2012. Accessed: 2017-04-26.

[3] The-Hien Dang-Ha, Filippo Maria Bianchi, and Roland Olsson. Local short term electricity load forecasting: Automatic approaches. In *Neural Networks (IJCNN), 2017 International Joint Conference on*, pages 4267–4274. IEEE, 2017.

[4] Filippo Maria Bianchi, Enrico Maiorino, Michael C Kampffmeyer, Antonello Rizzi, and Robert Jenssen. *Recurrent Neural Networks for Short-Term Load Forecasting: An Overview and Comparative Analysis*. Springer, 2017.

[5] Eisa Almeshaiei and Hassan Soltan. A methodology for electric power load forecasting. *Alexandria Engineering Journal*, 50(2):137–144, 2011.

[6] Yi Yang, Jie Wu, Yanhua Chen, and Caihong Li. A new strategy for short-term load forecasting. In *Abstract and Applied Analysis*, volume 2013. Hindawi, 2013.

[7] Bo Wang, Neng-ling Tai, Hai-qing Zhai, Jian Ye, Jia-dong Zhu, and Liang-bo Qi. A new armax model based on evolutionary algorithm and particle swarm optimization for short-term load forecasting. *Electric Power Systems Research*, 78 (10):1679–1685, 2008.

[8] A. Deihimi and H. Showkati. Application of echo state networks in short-term electric load forecasting. *Energy*, 39(1):327–340, 2012.

[9] Y. Peng, M. Lei, J.-B. Li, and X.-Y. Peng. A novel hybridization of echo state networks and multiplicative seasonal ARIMA model for mobile communication traffic series forecasting. *Neural Computing and Applications*, 24(3-4):883–890, 2014.

[10] H. Shen and J. Z. Huang. Interday forecasting and intraday updating of call center arrivals. *Manufacturing & Service Operations Management*, 10(3):391–410, 2008.

[11] F M Bianchi, Simone Scardapane, Aurelio Uncini, Antonello Rizzi, and Alireza Sadeghian. Prediction of telephone calls load using Echo State Network with exogenous variables. *Neural Networks*, 71:204–213, 2015. URL 10.1016/j.neunet.2015.08.010.

[12] F. M. Bianchi, E. De Santis, A. Rizzi, and A. Sadeghian. Short-term electric load forecasting using echo state networks and PCA decomposition. *IEEE Access*, 3: 1931–1943, Oct. 2015. ISSN 2169-3536. URL 10.1109/ACCESS.2015.2485943.

[13] Luiz Felipe Amaral, Reinaldo Castro Souza, and Maxwell Stevenson. A smooth transition periodic autoregressive (stpar) model for short-term load forecasting. *International Journal of Forecasting*, 24(4):603–615, 2008.

[14] S Sp Pappas, L Ekonomou, P Karampelas, DC Karamousantas, SK Katsikas, GE Chatzarakis, and PD Skafidas. Electricity demand load forecasting of the hellenic power system using an arma model. *Electric Power Systems Research*, 80 (3):256–264, 2010.

[15] John V Ringwood, D Bofelli, and Fiona T Murray. Forecasting electricity demand on short, medium and long time scales using neural networks. *Journal of Intelligent and Robotic Systems*, 31(1-3):129–147, 2001.

[16] Dong-Xiao Niu, Qiang Wang, and Jin-Chao Li. Short term load forecasting model based on support vector machine. In *Advances in Machine Learning and Cybernetics*, pages 880–888. Springer, 2006.

[17] Tao Hong and Mohammad Shahidehpour. Load forecasting case study. *EISPC, US Department of Energy*, 2015.

[18] J. W. Taylor. A comparison of univariate time series methods for forecasting intraday arrivals at a call center. *Management Science*, 54(2):253–265, 2008.

[19] Prajakta S Kalekar. Time series forecasting using holt-winters exponential smoothing. *Kanwal Rekhi School of Information Technology*, 4329008:1–13, 2004.

[20] Anton Maximilian Schäfer and Hans-Georg Zimmermann. Recurrent neural networks are universal approximators. *International Journal of Neural Systems*, 17 (04):253–263, 2007. URL 10.1142/S0129065707001111.

[21] Yoshua Bengio and Francois Gingras. Recurrent neural networks for missing or asynchronous data. In *Advances in neural information processing systems*, pages 395–401, 1996.

[22] Volker Tresp and Thomas Briegel. A solution for missing data in recurrent neural networks with an application to blood glucose prediction. In *Advances in Neural Information Processing Systems*, pages 971–977, 1998.

[23] J. Zhang, J. Han, R. Wang, and G. Hou. Day-ahead electricity price forecasting based on rolling time series and least square-support vector machine model. In *2011 Chinese Control and Decision Conference (CCDC)*, pages 1065–1070, May 2011. doi: 10.1109/CCDC.2011.5968342.

[24] Xiaolei Ma, Zhimin Tao, Yinhai Wang, Haiyang Yu, and Yunpeng Wang. Long short-term memory neural network for traffic speed prediction using remote microwave sensor data. *Transportation Research Part C: Emerging Technologies*, 54: 187–197, 2015.

[25] Nikolay Laptev, Jason Yosinski, Li Erran Li, and Slawek Smyl. Time-series extreme event forecasting with neural networks at uber. In *International Conference on Machine Learning*, number 34, pages 1–5, 2017.

[26] Mark Woodward, WCS Smith, and Hugh Tunstall-pedoe. Bias from missing values: sex differences in implication of failed venepuncture for the scottish heart health study. *International journal of epidemiology*, 20(2):379–383, 1991.

[27] Zhiyong Cui, Ruimin Ke, and Yinhai Wang. Deep bidirectional and unidirectional LSTM recurrent neural network for network-wide traffic speed prediction. *CoRR*, abs/1801.02143, 2018. URL http://arxiv.org/abs/1801.02143.

[28] Derrick A Bennett. How can i deal with missing data in my study? *Australian and New Zealand journal of public health*, 25(5):464–469, 2001.

[29] Shiyu Chang, Yang Zhang, Wei Han, Mo Yu, Xiaoxiao Guo, Wei Tan, Xiaodong Cui, Michael Witbrock, Mark A Hasegawa-Johnson, and Thomas S Huang. Dilated recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 77–87, 2017.

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

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

[32] HM Al-Hamadi and SA Soliman. Short-term electric load forecasting based on kalman filtering algorithm with moving window weather and load model. *Electric power systems research*, 68(1):47–59, 2004.

[33] K. Uezato T. Senjyu, P. Mandal and T. Funabashi. Next day load curve forecasting using recurrent neural network structure. *IEE Proceedings-Generation, Transmission and . . .*, 151(3):201–212, 2004. ISSN 1350-2360. URL 10.1049/ip-gtd.

[34] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443 – 473, 2006. ISSN 0169-2070. URL http://doi.org/10.1016/j.ijforecast.2006.01.001. Twenty five years of forecasting.

[35] David Simchi-Levi, Edith Simchi-Levi, and Philip Kaminsky. *Designing and managing the supply chain: Concepts, strategies, and cases.* McGraw-Hill New York, 1999.

[36] D. W. Bunn. Forecasting loads and prices in competitive power markets. *Proceedings of the IEEE*, 88(2), 2000.

[37] P. A. Ruiz and G. Gross. Short-term resource adequacy in electricity market design. *IEEE Transactions on Power Systems*, 23(3):916–926, 2008.

[38] Shu Fan and Rob J Hyndman. Short-term load forecasting based on a semi-parametric additive model. *IEEE Transactions on Power Systems*, 27(1):134–141, 2012.

[39] Rob Hyndman, Anne B Koehler, J Keith Ord, and Ralph D Snyder. *Forecasting with exponential smoothing: the state space approach.* Springer Science & Business Media, 2008. ISBN 9783540719182.

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

[41] George EP Box and David R Cox. An analysis of transformations. *Journal of the Royal Statistical Society. Series B (Methodological)*, pages 211–252, 1964.

[42] James W Taylor, Patrick E McSharry, et al. Short-term load forecasting methods: An evaluation based on european data. *IEEE Transactions on Power Systems*, 22 (4):2213–2219, 2007.

[43] James W. Taylor. Triple seasonal methods for short-term electricity demand forecasting. *European Journal of Operational Research*, 204(1):139–152, 2010. ISSN 03772217. doi: 10.1016/j.ejor.2009.10.003.

[44] Peter J Brockwell, Richard A Davis, and Matthew V Calder. *Introduction to time series and forecasting*, volume 2. Springer, 2002.

[45] Rob J Hyndman and George Athanasopoulos. *Forecasting: principles and practice.* OTexts, 2018.

[46] William WS Wei. Time series analysis. In *The Oxford Handbook of Quantitative Methods in Psychology: Vol. 2.* 2006.

[47] Martin T Hagan and Suzanne M Behr. The time series approach to short term load forecasting. *IEEE Transactions on Power Systems*, 2(3):785–791, 1987.

[48] Hong-Tzer Yang, Chao-Ming Huang, and Ching-Lien Huang. Identification of armax model for short term load forecasting: An evolutionary programming approach. In *Power Industry Computer Application Conference, 1995. Conference Proceedings., 1995 IEEE*, pages 325–330. IEEE, 1995.

[49] Rafał Weron and Adam Misiorek. Forecasting spot electricity prices: A comparison of parametric and semiparametric time series models. *International journal of forecasting*, 24(4):744–763, 2008.

[50] Billy M Williams and Lester A Hoel. Modeling and forecasting vehicular traffic flow as a seasonal arima process: Theoretical basis and empirical results. *Journal of transportation engineering*, 129(6):664–672, 2003.

[51] Theresa Hoang Diem Ngo Ngo. The Box-Jenkins Methodology for Time Series Models. *Proceedings of the SAS Global Forum 2013 conference*, 6:1–11, 2013. URL http://support.sas.com/resources/papers/proceedings13/454-2013.pdf.

[52] S Aman, M Frincu, C Charalampos, U Noor, Y Simmhan, and V Prasanna. Empirical comparison of prediction methods for electricity consumption forecasting. *University of Southern California, Tech. Rep*, pages 14–942, 2014.

[53] David Posada and Thomas R Buckley. Model selection and model averaging in phylogenetics: advantages of akaike information criterion and bayesian approaches over likelihood ratio tests. *Systematic biology*, 53(5):793–808, 2004.

[54] Carroll Croarkin, Paul Tobias, JJ Filliben, Barry Hembree, Will Guthrie, et al. Nist/sematech e-handbook of statistical methods. *NIST/SEMATECH, July. Available online: http://www. itl. nist. gov/div898/handbook*, 2006.

[55] Nicholas I Sapankevych and Ravi Sankar. Time series prediction using support vector machines: a survey. *Computational Intelligence Magazine, IEEE*, 4(2): 24–38, 2009.

[56] Kristoffer Knutsen Wickstrøm. Uncertainty modeling and interpretability in convolutional neural networks for polyp segmentation. Master's thesis, UiT The Arctic University of Norway, 2018.

[57] Lean Yu, Shouyang Wang, and Kin Keung Lai. Forecasting crude oil price with an emd-based neural network ensemble learning paradigm. *Energy Economics*, 30 (5):2623–2635, 2008.

[58] Mohamed A Mohandes, Shafiqur Rehman, and Talal O Halawani. A neural networks approach for wind speed prediction. *Renewable Energy*, 13(3):345–354, 1998.

[59] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural networks*, 2(5):359–366, 1989.

[60] 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. ISSN 0169-2070. URL http://doi.org/10.1016/S0169-2070(97)00044-7.

[61] H.S. Hippert, C.E. Pedreira, and R.C. Souza. Neural networks for short-term load forecasting: a review and evaluation. *IEEE Transactions on Power Systems*, 16 (1):44–55, 2001. ISSN 08858950. URL 10.1109/59.910780.

[62] Rob Law. Back-propagation learning in improving the accuracy of neural network-based tourism demand forecasting. *Tourism Management*, 21(4):331 – 340, 2000. ISSN 0261-5177. URL http://doi.org/10.1016/S0261-5177(99)00067-9.

[63] Sheng-Hshiung Tsaur, Yi-Chang Chiu, and Chung-Huei Huang. Determinants of guest loyalty to international tourist hotelsa neural network approach. *Tourism Management*, 23(4):397 – 405, 2002. ISSN 0261-5177. URL http://doi.org/10.1016/S0261-5177(01)00097-8.

[64] Sen Cheong Kon and Lindsay W Turner. Neural network forecasting of tourism demand. *Tourism Economics*, 11(3):301–328, 2005. URL http://dx.doi.org/10.5367/000000005774353006.

[65] Alfonso Palmer, Juan Jos Montao, and Albert Ses. Designing an artificial neural network for forecasting tourism time series. *Tourism Management*, 27(5):781 – 790, 2006. ISSN 0261-5177. URL http://doi.org/10.1016/j.tourman.2005.05.006.

[66] Oscar Claveria and Salvador Torra. Forecasting tourism demand to catalonia: Neural networks vs. time series models. *Economic Modelling*, 36:220 – 228, 2014. ISSN 0264-9993. URL http://doi.org/10.1016/j.econmod.2013.09.024.

[67] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[68] Nikolaos Kourentzes. Intermittent demand forecasts with neural networks. *International Journal of Production Economics*, 143(1):198 – 206, 2013. ISSN 0925-5273. URL http://doi.org/10.1016/j.ijpe.2013.01.009.

[69] Luis A. Daz-Robles, Juan C. Ortega, Joshua S. Fu, Gregory D. Reed, Judith C. Chow, John G. Watson, and Juan A. Moncada-Herrera. A hybrid {ARIMA} and artificial neural networks model to forecast particulate matter in urban areas: The case of temuco, chile. *Atmospheric Environment*, 42(35):8331 – 8340, 2008. ISSN 1352-2310. URL http://doi.org/10.1016/j.atmosenv.2008.07.020.

[70] E Plummer. Time series forecasting with feed-forward neural networks: guidelines and limitations. *Neural Networks*, 1:1, 2000.

[71] Joo Paulo Teixeira and Paula Odete Fernandes. Tourism time series forecast - different ann architectures with time index input. *Procedia Technology*, 5:445 – 454, 2012. ISSN 2212-0173. URL http://dx.doi.org/10.1016/j.protcy.2012.09.049.

[72] Oscar Claveria, Enric Monte, and Salvador Torra. Tourism demand forecasting with neural network models: Different ways of treating information. *International Journal of Tourism Research*, 17(5):492–500, 2015. ISSN 1522-1970. URL 10.1002/jtr.2016. JTR-13-0416.R2.

[73] Filippo Maria Bianchi, Michael Kampffmeyer, Enrico Maiorino, and Robert Jenssen. Temporal overdrive recurrent neural network. *arXiv preprint arXiv:1701.05159*, 2017.

[74] Alex Graves. Sequence transduction with recurrent neural networks. *arXiv preprint arXiv:1211.3711*, 2012.

[75] Alex Graves. Generating sequences with recurrent neural networks. *arXiv preprint arXiv:1308.0850*, pages 1–43, 2013.

[76] Razvan Pascanu, Tomas Mikolov, and Yoshua Bengio. On the difficulty of training recurrent neural networks. In *International Conference on Machine Learning*, pages 1310–1318, 2013.

[77] T. Mikolov. *Statistical language models based on neural networks.* PhD thesis, PhD thesis, Brno University of Technology. 2012.[PDF], 2012.

[78] Ilya Sutskever, James Martens, and Geoffrey E Hinton. Generating text with recurrent neural networks. In *Proceedings of the 28th International Conference on Machine Learning (ICML-11)*, pages 1017–1024, 2011.

[79] Alex Graves. Practical variational inference for neural networks. In *Advances in Neural Information Processing Systems*, pages 2348–2356, 2011.

[80] 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.

[81] Aaron 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. *arXiv preprint arXiv:1609.03499*, 2016.

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

[83] Alex Graves, Marcus Liwicki, Horst Bunke, Jürgen Schmidhuber, and Santiago Fernández. Unconstrained on-line handwriting recognition with recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 577–584, 2008.

[84] 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.

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

[86] Junyoung Chung, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. Gated feedback recurrent neural networks. In *International Conference on Machine Learning*, pages 2067–2075, 2015.

[87] Zeping Yu and Gongshen Liu. Sliced recurrent neural networks. *arXiv preprint arXiv:1807.02291*, 2018.

[88] 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.

[89] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning internal representations by error propagation. Technical report, California Univ San Diego La Jolla Inst for Cognitive Science, 1985.

[90] Zhengping Che, Sanjay Purushotham, Kyunghyun Cho, David Sontag, and Yan Liu. Recurrent neural networks for multivariate time series with missing values. *Scientific reports*, 8(1):6085, 2018.

[91] Filippo Maria Bianchi, Lorenzo Livi, and Cesare Alippi. Investigating echo-state networks dynamics by means of recurrence analysis. *IEEE transactions on neural networks and learning systems*, 2016.

[92] Bianchi Filippo Maria, Livi Lorenzo, Alippi Cesare, and Jenssen Robert. Multiplex visibility graphs to investigate recurrent neural network dynamics. *Scientific Reports*, 7:44037, mar 2017. URL http://dx.doi.org/10.1038/srep4403710.1038/srep44037.

[93] Sepp Hochreiter. The vanishing gradient problem during learning recurrent neural nets and problem solutions. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 6(02):107–116, 1998.

[94] Lorenzo Livi, Filippo Maria Bianchi, and Cesare Alippi. Determination of the edge of criticality in echo state networks through fisher information maximization. *IEEE transactions on neural networks and learning systems*, 2017.

[95] Ronald J Williams and Jing Peng. An efficient gradient-based algorithm for online training of recurrent network trajectories. *Neural computation*, 2(4):490–501, 1990.

[96] Yoshua Bengio et al. Learning deep architectures for ai. *Foundations and trends® in Machine Learning*, 2(1):1–127, 2009.

[97] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. Domain adaptation for large-scale sentiment classification: A deep learning approach. In *Proceedings of the 28th international conference on machine learning (ICML-11)*, pages 513–520, 2011.

[98] Jürgen Schmidhuber. Learning complex, extended sequences using the principle of history compression. *Neural Computation*, 4(2):234–242, 1992.

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

[100] Alex Graves. Generating sequences with recurrent neural networks. *CoRR*, abs/1308.0850, 2013. URL http://arxiv.org/abs/1308.0850.

[101] Razvan Pascanu, Caglar Gulcehre, Kyunghyun Cho, and Yoshua Bengio. How to construct deep recurrent neural networks. *arXiv preprint arXiv:1312.6026*, 2013.

[102] Antonio Valerio Miceli Barone, Jindrich Helcl, Rico Sennrich, Barry Haddow, and Alexandra Birch. Deep architectures for neural machine translation. *CoRR*, abs/1707.07631, 2017. URL http://arxiv.org/abs/1707.07631.

[103] Junyoung Chung, Sungjin Ahn, and Yoshua Bengio. Hierarchical multiscale recurrent neural networks. *arXiv preprint arXiv:1609.01704*, 2016.

[104] Ian Goodfellow, Yoshua Bengio, Aaron Courville, and Yoshua Bengio. *Deep learning*, volume 1. MIT press Cambridge, 2016.

[105] Saizheng Zhang, Yuhuai Wu, Tong Che, Zhouhan Lin, Roland Memisevic, Ruslan R Salakhutdinov, and Yoshua Bengio. Architectural complexity measures of recurrent neural networks. In *Advances in Neural Information Processing Systems*, pages 1822–1830, 2016.

[106] Dzmitry Bahdanau, Jan Chorowski, Dmitriy Serdyuk, Philemon Brakel, and Yoshua Bengio. End-to-end attention-based large vocabulary speech recognition. In *Acoustics, Speech and Signal Processing (ICASSP), 2016 IEEE International Conference on*, pages 4945–4949. IEEE, 2016.

[107] Aäron van den Oord, Sander Dieleman, Heiga Zen, Karen Simonyan, Oriol Vinyals, Alex Graves, Nal Kalchbrenner, Andrew W. Senior, and Koray Kavukcuoglu. Wavenet: A generative model for raw audio. *CoRR*, abs/1609.03499, 2016. URL http://arxiv.org/abs/1609.03499.

[108] Jie Wang, Jun Wang, Wen Fang, and Hongli Niu. Financial time series prediction using elman recurrent random neural networks. *Computational intelligence and neuroscience*, 2016, 2016.

[109] Jan Koutník, Klaus Greff, Faustino J. Gomez, and Jürgen Schmidhuber. A clockwork RNN. *CoRR*, abs/1402.3511, 2014. URL http://arxiv.org/abs/1402.3511.

[110] Fisher Yu and Vladlen Koltun. Multi-scale context aggregation by dilated convolutions. *CoRR*, abs/1511.07122, 2015. URL http://arxiv.org/abs/1511.07122.

[111] Donald B Rubin and Nathaniel Schenker. Multiple imputation for interval estimation from simple random samples with ignorable nonresponse. *Journal of the American statistical Association*, 81(394):366–374, 1986.

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

[113] Matthew G Karlaftis and Eleni I Vlahogianni. Statistical methods versus neural networks in transportation research: Differences, similarities and some insights. *Transportation Research Part C: Emerging Technologies*, 19(3):387–399, 2011.

[114] Xiuwen Yi, Yu Zheng, Junbo Zhang, and Tianrui Li. St-mvl: Filling missing values in geo-sensory time series data. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, IJCAI'16, pages 2704–2710. AAAI Press, 2016. ISBN 978-1-57735-770-4. URL http://dl.acm.org/citation.cfm?id=3060832.3060999.

[115] Jun Wang, Arjen P De Vries, and Marcel JT Reinders. Unifying user-based and item-based collaborative filtering approaches by similarity fusion. In *Proceedings of the 29th annual international ACM SIGIR conference on Research and development in information retrieval*, pages 501–508. ACM, 2006.

[116] Tobias Schnabel, Adith Swaminathan, Ashudeep Singh, Navin Chandak, and Thorsten Joachims. Recommendations as treatments: Debiasing learning and evaluation. *CoRR*, abs/1602.05352, 2016. URL http://arxiv.org/abs/1602.05352.

[117] 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.

[118] Zachary C Lipton, David C Kale, and Randall Wetzel. Modeling missing data in clinical time series with rnns. *Machine Learning for Healthcare*, 2016.

[119] Andreas Storvik Strauman, Filippo Maria Bianchi, Karl Øyvind Mikalsen, Michael Kampffmeyer, Cristina Soguero-Ruiz, and Robert Jenssen. Classification of post-operative surgical site infections from blood measurements with missing data using recurrent neural networks. In *Biomedical & Health Informatics (BHI), 2018 IEEE EMBS International Conference on*, pages 307–310. IEEE, 2018.

[120] Shahla Parveen. *Connectionist approaches to the deployment of prior knowledge for improving robustness in automatic speech recognition.* PhD thesis, University of Sheffield, 2003.

[121] Han-Gyu Kim, Gil-Jin Jang, Ho-Jin Choi, Minho Kim, Young-Won Kim, and Jaehun Choi. Recurrent neural networks with missing information imputation for medical examination data prediction. In *Big Data and Smart Computing (BigComp), 2017 IEEE International Conference on*, pages 317–323. IEEE, 2017.

[122] Zachary Chase Lipton, David C. Kale, Charles Elkan, and Randall C. Wetzel. Learning to diagnose with LSTM recurrent neural networks. *CoRR*, abs/1511.03677, 2015. URL http://arxiv.org/abs/1511.03677.

[123] JWC Van Lint, SP Hoogendoorn, and Henk J van Zuylen. Accurate freeway travel time prediction with state-space neural networks under missing data. *Transportation Research Part C: Emerging Technologies*, 13(5-6):347–369, 2005.

[124] Pedro J. García-Laencina, José-Luis Sancho-Gómez, and Aníbal R. Figueiras-Vidal. Pattern classification with missing data: a review. *Neural Computing and Applications*, 19(2):263–282, Mar 2010. ISSN 1433-3058. doi: 10.1007/s00521-009-0295-6. URL https://doi.org/10.1007/s00521-009-0295-6.

[125] Donald B Rubin. Inference and missing data. *Biometrika*, 63(3):581–592, 1976.

[126] Jinsung Yoon, William R. Zame, and Mihaela van der Schaar. Estimating missing data in temporal data streams using multi-directional recurrent neural networks. *CoRR*, abs/1711.08742, 2017. URL http://arxiv.org/abs/1711.08742.

[127] Alex Graves and Jürgen Schmidhuber. Framewise phoneme classification with bidirectional lstm and other neural network architectures. *Neural Networks*, 18 (5-6):602–610, 2005.

[128] Wei Cao, Dong Wang, Jian Li, Hao Zhou, Lei Li, and Yitan Li. BRITS: bidirectional recurrent imputation for time series. *CoRR*, abs/1805.10572, 2018. URL http://arxiv.org/abs/1805.10572.

[129] Roderick JA Little and Donald B Rubin. *Statistical analysis with missing data*, volume 333. John Wiley & Sons, 2014.

[130] Denise F Polit and Cheryl Tatano Beck. *Nursing research: Generating and assessing evidence for nursing practice*. Lippincott Williams & Wilkins, 2008.

[131] Li Zhou and George Hripcsak. Temporal reasoning with medical dataa review with emphasis on medical natural language processing. *Journal of biomedical informatics*, 40(2):183–202, 2007.

[132] Yoram Vodovotz, Gary An, and Ioannis P Androulakis. A systems engineering perspective on homeostasis and disease. *Frontiers in bioengineering and biotechnology*, 1:6, 2013.

[133] Karl Moritz Hermann and Phil Blunsom. Multilingual distributed representations without word alignment. *arXiv preprint arXiv:1312.6173*, 2013.

[134] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Dec Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL http://tensorflow.org/. Software available from tensorflow.org.

[135] J Doyne Farmer and John J Sidorowich. Predicting chaotic time series. *Physical review letters*, 59(8):845, 1987.

[136] Yurii Nesterov. A method of solving a convex programming problem with convergence rate O(1/sqrt(k)). *Soviet Mathematics Doklady*, 27:372–376, 1983.

[137] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014. URL `http://arxiv.org/abs/1412.6980`.

[138] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *The Journal of Machine Learning Research*, 13(1):281–305, 2012.

[139] Albert Zeyer, Patrick Doetsch, Paul Voigtlaender, Ralf Schlüter, and Hermann Ney. A comprehensive study of deep bidirectional LSTM rnns for acoustic modeling in speech recognition. *CoRR*, abs/1606.06871, 2016.

[140] Ning Qian. On the momentum term in gradient descent learning algorithms. *Neural Networks*, 12(1):145 – 151, 1999. ISSN 0893-6080. doi: https://doi.org/10.1016/S0893-6080(98)00116-6. URL `http://www.sciencedirect.com/science/article/pii/S0893608098001166`.

[141] John Duchi, Elad Hazan, and Yoram Singer. Adaptive subgradient methods for online learning and stochastic optimization. *J. Mach. Learn. Res.*, 12:2121–2159, jul 2011. ISSN 1532-4435. URL `http://dl.acm.org/citation.cfm?id=1953048.2021068`.

[142] Matthew D. Zeiler. ADADELTA: an adaptive learning rate method. *CoRR*, abs/1212.5701, 2012. URL `http://arxiv.org/abs/1212.5701`.

[143] T Tieleman and G Hinton. Rmsprop adaptive learning. in: Coursera: Neural networks for machine learning, 2012.