



저작자표시-비영리-변경금지 2.0 대한민국

이용자는 아래의 조건을 따르는 경우에 한하여 자유롭게

- 이 저작물을 복제, 배포, 전송, 전시, 공연 및 방송할 수 있습니다.

다음과 같은 조건을 따라야 합니다:



저작자표시. 귀하는 원저작자를 표시하여야 합니다.



비영리. 귀하는 이 저작물을 영리 목적으로 이용할 수 없습니다.



변경금지. 귀하는 이 저작물을 개작, 변형 또는 가공할 수 없습니다.

- 귀하는, 이 저작물의 재이용이나 배포의 경우, 이 저작물에 적용된 이용허락조건을 명확하게 나타내어야 합니다.
- 저작권자로부터 별도의 허가를 받으면 이러한 조건들은 적용되지 않습니다.

저작권법에 따른 이용자의 권리는 위의 내용에 의하여 영향을 받지 않습니다.

이것은 [이용허락규약\(Legal Code\)](#)을 이해하기 쉽게 요약한 것입니다.

[Disclaimer](#)

Master's Thesis

# Deep Neural Networks to Learn Basis Functions with a Temporal Covariance Loss

Janghoon Ju

Department of Computer Science and Engineering

Graduate School of UNIST

2019

# Deep Neural Networks to Learn Basis Functions with a Temporal Covariance Loss

Janghoon Ju

Department of Computer Science and Engineering

Graduate School of UNIST

# Deep Neural Networks to Learn Basis Functions with a Temporal Covariance Loss

A thesis  
submitted to the Graduate School of UNIST  
in partial fulfillment of the  
requirements for the degree of  
Master of Science

Janghoon Ju

7/5/2019

Approved by

Jaesik Choi

Advisor


Jaesik Choi

# Deep Neural Networks to Learn Basis Functions with a Temporal Covariance Loss

Janghoon Ju

This certifies that the thesis of Janghoon Ju is approved.


7/5/2019



Advisor: Jaesik Choi



Committee Member: Kwang In Kim



Committee Member: Sungahn Ko

## **Abstract**

Deep Neural Networks (DNNs) and Gaussian Processes (GPs) are commonly used prediction models to solve regression problems on time series data. A GP can approximate a smooth function arbitrarily well. When the function satisfies some conditions. We adopt the principles of GP learning to DNN learning on time series data. While previous approaches need to change the architecture of DNNs or be explicitly derived from the GPs algorithm, we concentrate on the learning scheme of DNNs to leverage the important principles of GPs by proposing the Temporal Covariance loss function. Whereas the conventional loss function of DNNs only captures the mean of the target values, Temporal Covariance loss function further captures the covariance of the target values where covariance function is the other factor to define GPs along with the mean function. We show that learning DNNs and Convolutional Neural Networks (CNNs) with the Temporal Covariance loss function can obtain more accurate models for sets of regression problems with US groundwater data and NASDAQ 100 stock data.



# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Background</b>	<b>2</b>
2.1	Regression . . . . .	2
2.2	Linear Regression . . . . .	2
2.3	Gaussian Processes . . . . .	3
2.3.1	Smooth Functions . . . . .	4
2.3.2	Positive Semidefinite Kernels . . . . .	4
2.3.3	Reproducing Kernel Hilbert Space . . . . .	5
2.3.4	Eigenfunctions of Kernels . . . . .	6
2.4	Deep Neural Networks . . . . .	7
2.4.1	Convolutional Neural Networks . . . . .	8
2.4.2	Long Short-Term Memory Networks . . . . .	8
<b>3</b>	<b>Related Work</b>	<b>10</b>
<b>4</b>	<b>Temporal Covariance Loss</b>	<b>11</b>
4.1	Learning the Basis Functions in Deep Neural Networks . . . . .	11



4.2	Temporal Covariance Loss . . . . .	13
4.3	Principles of a Temporal Covariance Loss Based Neural Networks . . . . .	14
<b>5</b>	<b>Experimental Results</b>	<b>16</b>
5.1	Experimental Setup . . . . .	16
5.2	Groundwater Dataset . . . . .	17
5.2.1	Quantitative Analysis . . . . .	19
5.2.2	Quantitative Analysis with Fully Connected Neural Networks . . . . .	20
5.2.3	Quantitative Analysis with Convolutional Neural Networks . . . . .	22
5.3	Stock Market Dataset . . . . .	24
5.3.1	Quantitative Analysis . . . . .	25
5.3.2	Quantitative Analysis with Fully Connected Neural Networks . . . . .	26
5.3.3	Quantitative Analysis with Convolutional Neural Networks . . . . .	28
<b>6</b>	<b>Conclusion</b>	<b>30</b>

# List of Figures

4.1	An architecture of a DNN model for regression tasks. There are various types of layers such as convolutional or fully-connected layers in the DNNs (grayed line in the figure). When we consider the blue colored layer as the last layer of DNN, the relation between the last hidden and output layer can be represented by the linear regression. . . . .	12
4.2	The proposed structure of DNN with the Temporal Covariance loss function. The DNN is trained to reduce the mean squared error (MSE) loss between the Temporal Covariance and the outer product of the basis function matrix. . . . .	13
5.1	The details of experiments setting on the groundwater dataset with DNNs. Where target $\mathbf{y}$ is chosen in given dataset $\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^{88}\}$ . The purpose of the experiments is finding $F(\cdot)$ by DNNs. The function $F(\cdot)$ represents the relationship between $\mathbf{y}$ and $\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^{88}\}$ with given 10 days delayed. . . . .	17
5.2	Comparisons the root mean square error (RMSE) of proposed models with four baseline approach models. The $y$ -axis is RMSE of each model and the linear regression as “Linear” and the horizontal red line, the baseline approach models are denoted as yellow color, the proposed models with hyper-parameter 0.1 are labeled by “DNN Cov” and “CNN Cov” with green color. . . . .	19
5.3	Comparisons the RMSE between the linear regression, the baseline approach DNN model and the proposed model on the number of 10 randomly selected groundwater dataset. The $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the horizontal red line, the baseline approach DNN model is denoted “Baseline” and the color is yellow, the proposed model with hyper-parameter 0.1 is labeled by “Lambda 0.1” with green color. . . . .	20

5.4	Comparisons the RMSE of the proposed DNN model with three different hyper-parameters on the number of 10 randomly selected groundwater dataset. The $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the red horizontal line, the baseline approach model as “Baseline” and the color is yellow, and the proposed model with different hyper-parameters $\{0.1, 1, 10\}$ as “Lambda $\{0.1, 1, 10\}$ ” with green color. . . .	21
5.5	Comparisons the RMSE between the linear regression, the baseline approach CNN model and the proposed model on the number of 10 randomly selected groundwater dataset. The $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the horizontal red line, the baseline approach CNN model is denoted “Baseline” and the color is yellow, the proposed model with hyper-parameter 0.1 is labeled by “Lambda 0.1” with green color. . . . .	22
5.6	Comparisons the RMSE of the proposed CNN model with three different hyper-parameters on the number of 10 randomly selected groundwater dataset. The $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the red horizontal line, the baseline approach model as “Baseline” and the color is yellow, and the proposed model with different hyper-parameters $\{0.1, 1, 10\}$ as “Lambda $\{0.1, 1, 10\}$ ” with green color. . . .	23
5.7	Comparisons the RMSE of proposed models with four baseline approach models. The $y$ -axis is RMSE of each model and the linear regression as “Linear” and the horizontal red line, the baseline approach models are denoted as yellow color, the proposed models with hyper-parameter 0.1 are labeled by “DNN Cov” and “CNN Cov” with green color. .	25
5.8	Comparisons the RMSE between the linear regression, the baseline approach DNN model and the proposed model on the number of 10 randomly selected stock market dataset. The $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the horizontal red line, the baseline approach DNN model is denoted “Baseline” and the color is yellow, the proposed model with hyper-parameter 0.1 is labeled by “Lambda 0.1” with green color. . . . .	26
5.9	Comparisons the RMSE of the proposed DNN model with three different hyper-parameters on the number of 10 randomly selected stock market dataset. The $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the red horizontal line, the baseline approach model as “Baseline” and the color is yellow, and the proposed model with different hyper-parameters $\{0.1, 1, 10\}$ as “Lambda $\{0.1, 1, 10\}$ ” with green color. . . .	27

- 5.10 Comparisons the RMSE between the linear regression, the baseline approach CNN model and the proposed model on the number of 10 randomly selected stock market dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the horizontal red line, the baseline approach CNN model is denoted “Baseline” and the color is yellow, the proposed model with hyper-parameter 0.1 is labeled by “Lambda 0.1” with green color. . . . . 28
- 5.11 Comparisons the RMSE for the proposed CNN model with three different hyper-parameters on the number of 10 randomly selected stock market dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the red horizontal line, the baseline approach model as “Baseline” and the color is yellow, and the proposed model with different hyper-parameters  $\{0.1, 1, 10\}$  as “Lambda  $\{0.1, 1, 10\}$ ” with green color. . . . 29

# List of Tables

5.1	Comparisons the mean of RMSE between the linear regression, the Long short-term memory (LSTM) model, the baseline approach DNN, CNN model and the proposed models on the number of 10 randomly selected groundwater dataset. The baseline approach DNN, CNN model does not contain the Temporal Covariance loss and the proposed models with hyper-parameter 0.1 are denoted as “Proposed DNN model” and “Proposed CNN model”. A <b>bold</b> font indicates the best result obtained. . . . .	19
5.2	Comparisons the mean of RMSE between the linear regression, the baseline approach DNN model and the proposed model on the number of 10 randomly selected groundwater dataset. The baseline approach DNN model does not contain the Temporal Covariance loss and the proposed model with hyper-parameter 0.1 is denoted as “Proposed with $\lambda$ 0.1”. A <b>bold</b> font indicates the best result obtained. . . . .	20
5.3	Comparisons the mean of RMSE for the proposed DNN model with three different hyper-parameters $\lambda$ on the number of 10 randomly selected groundwater dataset. The proposed DNN model with different hyper-parameters $\{0.1, 1, 10\}$ is denoted as “Proposed with $\lambda \{0.1, 1, 10\}$ ”. A <b>bold</b> font indicates the best result obtained. . . . .	21
5.4	Comparisons the mean of RMSE between the linear regression, the baseline approach CNN model and the proposed model on the number of 10 randomly selected groundwater dataset. The baseline approach CNN model does not contain the Temporal Covariance loss and the proposed model with hyper-parameter 0.1 is denoted as “Proposed with $\lambda$ 0.1”. A <b>bold</b> font indicates the best result obtained. . . . .	22

- 5.5 Comparisons the mean of RMSE for the proposed CNN model with three different hyper-parameters  $\lambda$  on the number of 10 randomly selected groundwater dataset. The proposed CNN model with different hyper-parameters  $\{0.1, 1, 10\}$  is denoted as “Proposed with  $\lambda \{0.1, 1, 10\}$ ”. A **bold** font indicates the best result obtained. . . . . 23
- 5.6 Comparisons the mean of RMSE between the linear regression, the LSTM model, the baseline approach DNN, CNN model and the proposed models on the number of 10 randomly selected stock market dataset. The baseline approach DNN, CNN model does not contain the Temporal Covariance loss and the proposed models with hyper-parameter 0.1 are denoted as “Proposed DNN model” and “Proposed CNN model”. A **bold** font indicates the best result obtained or the best result among Neural Networks (NNs). . . . 25
- 5.7 Comparisons the mean of RMSE between the linear regression, the baseline approach DNN model and the proposed model on the number of 10 randomly selected stock market dataset. The baseline approach DNN model does not contain the Temporal Covariance loss and the proposed model with hyper-parameter 0.1 is denoted as “Proposed with  $\lambda 0.1$ ”. A **bold** font indicates the best result obtained. . . . . 26
- 5.8 Comparisons the mean of RMSE for the proposed DNN model with three different hyper-parameters  $\lambda$  on the number of 10 randomly selected stock market dataset. The proposed DNN model with different hyper-parameters  $\{0.1, 1, 10\}$  is denoted as “Proposed with  $\lambda \{0.1, 1, 10\}$ ”. A **bold** font indicates the best result obtained. . . . . 27
- 5.9 Comparisons the mean of RMSE between the linear regression, the baseline approach CNN model and the proposed model on the number of 10 randomly selected stock market dataset. The baseline approach CNN model does not contain the Temporal Covariance loss and the proposed model with hyper-parameter 0.1 is denoted as “Proposed with  $\lambda 0.1$ ”. A **bold** font indicates the best result obtained. . . . . 28
- 5.10 Comparisons the mean of RMSE for the proposed CNN model with three different hyper-parameters  $\lambda$  on the number of 10 randomly selected stock market dataset. The proposed CNN model with different hyper-parameters  $\{0.1, 1, 10\}$  is denoted as “Proposed with  $\lambda \{0.1, 1, 10\}$ ”. A **bold** font indicates the best result obtained. . . . . 29

# List of Abbreviations

**CNN** Convolutional Neural Network. 1, 8, 15, 16, 18, 19, 22–25, 28, 29

**DNN** Deep Neural Network. 1, 7, 8, 10–21, 24–27, 30

**ELU** exponential linear unit. 18, 24

**GP** Gaussian Process. 1, 3, 6, 10–12, 14, 15

**LSTM** Long short-term memory. 9, 18, 19, 24, 25, 30

**MSE** mean squared error. 13–15

**NN** Neural Network. 1, 8, 10, 15, 16, 18, 20, 24–26

**ReLU** rectified linear unit. 7

**RKHS** reproducing kernel Hilbert space. 5, 6, 10, 14

**RMSE** root mean square error. 17, 19–29

**RNN** Recurrent Neural Network. 8, 30

**SE** squared exponential. 3, 4

# Chapter 1

## Introduction

The regression problems are a set of problems finding relationships or functions among given input data and the continuous outcome, such as stock price prediction or weather forecasting. There are various machine learning models to solve regression problems. GPs [Rasmussen, 2003] or DNNs are commonly adopted to solve regression problems. Theoretically, the GPs can represent smooth functions which satisfy mean square continuity and the differentiability [Williams and Rasmussen, 2006]. Also, [Neal, 1995] explains that GPs can have the NNs as a kernel and the NNs can represent GPs with infinite numbers of hidden units.

The purpose of this work is to train a DNN with the principles of a GP learning. To guide the training procedure in our objective, we propose the novel loss function, Temporal Covariance loss, to train covariance information into the DNN for regression problems. More precisely, we aim to train the output of the last hidden layer of the DNN, which is the basis function of the DNN, to have its inner product approximate the covariance function via Temporal Covariance loss. Accordingly, the DNN learn the basis function which is more likely to be explained by GPs.

This paper is composed of the following chapters. Chapter 2 revisits basic concepts of the Linear regression, GPs, DNNs and CNNs. In Chapter 3 we introduce related work and the theoretical connection between GPs and DNNs. In Chapter 4, we explain the proposed model and its implementation in detail. Chapter 5 explains the settings of experiments, two real-world datasets, and the results. Finally, Chapter 6 wraps up with the conclusion of the thesis.



## Chapter 2

# Background

### 2.1 Regression

A regression task is one of the supervised machine learning tasks for continuous data pairs  $\{y|\mathbf{x}\}$ . The goal of the regression problem is to approximate function  $F(\cdot)$  which represents a relationship between a dependent variable  $y$  and one or multiple independent variables  $\mathbf{x}$  along a residual  $\epsilon$ ,

$$y = F(\mathbf{x}) + \epsilon. \quad (2.1)$$

### 2.2 Linear Regression

Linear regression models relation between a single dependent variable and one (or multiple) independent variable(s). The linear regression is one of the widely studied regression methods. With a dependent variable  $y$ , independent variables  $\mathbf{x}$  and a weight vector  $\mathbf{w}$ , the following Equation represents the general vector form of the linear regression with  $\epsilon$ , Gaussian noise,

$$f(\mathbf{x}) = \mathbf{x}^T \mathbf{w}, \quad y = f(\mathbf{x}) + \epsilon. \quad (2.2)$$

If we replace  $\mathbf{x}$  with  $\phi(\mathbf{x})$  which denotes a non-linear function of inputs, Equation (2.2) can be rephrased as

$$y = \phi(\mathbf{x})^T \mathbf{w} + \epsilon. \quad (2.3)$$

Acknowledged as basis function expansion [Murphy, 2012]. With  $\epsilon \sim \mathcal{N}(0, \sigma^2)$ , the linear regression equation becomes as follows,

$$p(y|\mathbf{x}, \mathbf{w}) = \mathcal{N}(y|\phi(\mathbf{x})^T \mathbf{w}, \sigma^2). \quad (2.4)$$

We call it as the Bayesian linear regression model [Williams and Rasmussen, 2006].

## 2.3 Gaussian Processes

Here, we will introduce the definition of GP.

**Definition 2.3.1.** GP is a collection of random variables, any finite number of which have a joint Gaussian distribution [Williams and Rasmussen, 2006].

A GP is effectively expressed by its mean function and covariance function.

$$\begin{aligned} m(x) &= \mathbb{E}[f(x)], \\ k(x, x') &= \mathbb{E}[(f(x) - m(x))(f(x') - m(x'))), \end{aligned} \quad (2.5)$$

where we define mean function  $m(x)$  and the covariance function  $k(x, x')$  of a real process  $f(x)$ .

$$f(x) \sim \mathcal{GP}(m(x), k(x, x')). \quad (2.6)$$

Whenever we select a subset from the collection of random variables from GP, the subset distribution is again Gaussian. When a GP is specified, we can calculate the marginal likelihood of given data or can derive predictive distribution for new points given existing data.

We can obtain an example of GP from our Bayesian linear regression model  $f(\mathbf{x}) = \phi^T \mathbf{w}$  with prior  $\mathbf{w} \sim \mathcal{N}(0, \Sigma_p)$ . Mean and covariance are as below.

$$\begin{aligned} \mathbb{E}[f(\mathbf{x})] &= \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}] = 0, \\ \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] &= \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \phi(\mathbf{x}') = \phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}'), \end{aligned} \quad (2.7)$$

$f(\mathbf{x})$  and  $f(\mathbf{x}')$  therefore are jointly Gaussian with the zero mean and covariance given by  $\phi(\mathbf{x})^T \Sigma_p \phi(\mathbf{x}')$ . Function values  $f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)$  corresponding to any number of input points  $n$  are in fact, jointly Gaussian.

In GP, we typically represent the covariance with a kernel function. A kernel function can depict some distinctive characteristics of functions, such as variance, length scales, and periodicity. For instance, the squared exponential (SE) covariance function is a typical example of covariance function.

The SE covariance function specifies the covariance between pairs of random variables with two hyper-parameters,  $\sigma$  and  $l$  where the length scale hyper-parameter  $l$  controls the smoothness of the functions.

$$\text{cov}_{SE}(f(\mathbf{x}_p), f(\mathbf{x}_q)) = k_{SE}(\mathbf{x}_p, \mathbf{x}_q) = \sigma^2 \exp\left(-\frac{|\mathbf{x}_p - \mathbf{x}_q|^2}{2l}\right). \quad (2.8)$$

Note that the covariance between outputs is described as a function of inputs. For this particular covariance function, the covariance is almost unity between variables when corresponding inputs are very close each other and it decreases as the distance of the input space expands.

Indeed, for every positive definite covariance function  $k(\cdot, \cdot)$ , there exists a (possibly infinite) expansion in terms of basis functions (see Mercer's theorem in section 2.3.4)

### 2.3.1 Smooth Functions

Smooth functions can be described in terms of the mean square continuity and differentiability of stochastic processes, following [Adler, 2010].

Suppose that a sequence of points  $\mathbf{x}_1, \mathbf{x}_2, \dots$  and  $\mathbf{x}_*$  are in  $\mathbb{R}^D$  such that  $\lim_{k \rightarrow \infty} |\mathbf{x}_k - \mathbf{x}_*| = 0$ . Then a process  $f(\mathbf{x})$  is continuous in the mean square at  $\mathbf{x}_*$  if  $\mathbb{E}[|f(\mathbf{x}_k) - f(\mathbf{x}_*)|^2] \rightarrow 0$  as  $k \rightarrow \infty$ . If this can apply to all  $\mathbf{x}_* \in A$  where  $A$  is a subset of  $\mathbb{R}^D$ ,  $f(\mathbf{x})$  is continuous in the mean square over  $A$ .

A random field is continuous in the mean square at  $\mathbf{x}_*$  if and only if its covariance function  $k(\mathbf{x}, \mathbf{x}')$  is continuous at the point  $\mathbf{x} = \mathbf{x}' = \mathbf{x}_*$ . For stationary covariance functions this reduces to checking continuity at  $k(\mathbf{0})$ .

The mean square derivative of  $f(\mathbf{x})$  in the  $i$ th direction is defined as

$$\frac{\partial f(\mathbf{x})}{\partial x_i} = \text{l.i.m.}_{h \rightarrow \infty} \frac{f(\mathbf{x} + h\mathbf{e}_i) - f(\mathbf{x})}{h}, \quad (2.9)$$

when the limit exists, where l.i.m denotes the limit in mean square and  $\mathbf{e}_i$  is the unit vector in the  $i$ th direction. The covariance function of  $\partial f(x)/\partial x_i$  is given by  $\partial^2 k(\mathbf{x}, \mathbf{x}')/\partial x_i \partial x'_i$ .

### 2.3.2 Positive Semidefinite Kernels

A kernel is a function  $k$  of two arguments mapping a pair of inputs  $\mathbf{x} \in \mathcal{X}$ ,  $\mathbf{x}' \in \mathcal{X}$  into  $\mathbb{R}$ . This general term arises in the theory of integral operators, where the operator  $T_k$  is defined as

$$(T_k f)(\mathbf{x}) = \int_{\mathcal{X}} k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}') d\mu(\mathbf{x}'), \quad (2.10)$$

where  $\mu$  denotes a measure. A real kernel is assumed to be symmetric if  $k(\mathbf{x}, \mathbf{x}') = k(\mathbf{x}', \mathbf{x})$ ; covariance functions then must be symmetric as in the definition.

We can compute the Gram matrix  $K$  whose entries are  $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$  given a set of input points  $\{\mathbf{x}_i | i = 1, \dots, n\}$  for some kernel function  $k$ . When  $k$  is a covariance function the matrix  $K$  is defined as the covariance matrix.

A real  $n \times n$  matrix  $K$  which satisfies  $Q(\mathbf{v}) = \mathbf{v}^T \mathbf{K} \mathbf{v} \geq 0$  for all vectors  $\mathbf{v} \in \mathbb{R}^n$  refers to positive semidefinite. If  $Q(\mathbf{v}) = 0$  only when  $\mathbf{v} = \mathbf{0}$ , the matrix is positive definite.  $Q(\mathbf{v})$  is termed as a quadratic form associated with  $\mathbf{K}$ . A symmetric matrix is positive semidefinite if and only if all of its eigenvalues are non-negative. A Gram matrix corresponding to a general kernel function is not necessarily positive semidefinite while that corresponds to a covariance function should be positive semidefinite.

A positive semidefinite kernel is stated as

$$\int k(\mathbf{x}, \mathbf{x}') f(\mathbf{x}) f(\mathbf{x}') d\mu(\mathbf{x}) d\mu(\mathbf{x}') \geq 0, \quad (2.11)$$

for all  $f \in L_2(\mathcal{X}, \mu)$ . For any choice of  $n \in \mathbb{N}$  and  $\mathcal{D}$ , a kernel function is thus positive semidefinite when it generates positive semidefinite Gram matrices which can be observed by letting  $f$  be the weighted sum of delta functions at each  $\mathbf{x}_i$ . The fact that those are limit functions in  $L_2(\mathcal{X}, \mu)$  allows Equation (2.11) indicate that the Gram matrix of any  $\mathcal{D}$  is supposed to be positive semidefinite.

### 2.3.3 Reproducing Kernel Hilbert Space

We now introduce reproducing kernel Hilbert space (RKHS). The theory was developed by [Aronszajn, 1950].

**Definition 2.3.2** (RKHS). Let  $\mathcal{H}$  be a Hilbert space of real functions  $f$  defined on an index set  $\mathcal{X}$ . Then  $\mathcal{H}$  is called a RKHS endowed with an inner product  $\langle \cdot, \cdot \rangle_{\mathcal{H}}$  (and norm  $\|f\|_{\mathcal{H}} = \sqrt{\langle f, f \rangle_{\mathcal{H}}}$ ) if there exists a function  $k : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R}$  with the following properties:

1. for every  $\mathbf{x}$ ,  $k(\mathbf{x}, \mathbf{x}')$  as a function of  $\mathbf{x}'$  belongs to  $\mathcal{H}$ , and
2.  $k$  has the reproducing property  $\langle f(\cdot), k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} = f(\mathbf{x})$ .

Note also that as  $k(\mathbf{x}, \cdot)$  and  $k(\mathbf{x}', \cdot)$  are in  $\mathcal{H}$  we have that  $\langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot) \rangle_{\mathcal{H}} = k(\mathbf{x}, \mathbf{x}')$  by following [Schölkopf et al., 2002] and [Wegman, 2014]. The RKHS uniquely determines  $k$ , and vice versa, as stated in the following

**Theorem 2.3.1** (Moore-Aronszajn theorem, [Aronszajn, 1950]). *Let  $\mathcal{X}$  be an index set. Then for every positive definite function  $k(\cdot, \cdot)$  on  $\mathcal{X} \times \mathcal{X}$  there exists a unique RKHS, and vice versa.*

Consider a real positive semidefinite kernel  $k(\mathbf{x}, \mathbf{x}')$  with an eigenfunction expansion  $k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^N \lambda_i \phi_i(\mathbf{x}) \phi_i(\mathbf{x}')$  relative to a measure  $\mu$ . From Mercer's theorem in section 2.3.4 that the eigenfunctions are orthonormal w.r.t.  $\mu$ , i.e. we have  $\int \phi_i(\mathbf{x}) \phi_j(\mathbf{x}) d\mu(\mathbf{x}) = \delta_{ij}$ . We now consider a Hilbert space comprised of linear combinations of eigenfunctions, i.e.  $f(\mathbf{x}) = \sum_{i=1}^N f_i \phi_i(\mathbf{x})$  with  $\sum_{i=1}^N f_i^2 / \lambda_i < \infty$ . The inner product  $\langle f, g \rangle_{\mathcal{H}}$  between functions  $f(\mathbf{x})$  and  $g(\mathbf{x}) = \sum_{i=1}^N g_i \phi_i(\mathbf{x})$  in the Hilbert space is alleged to be defined as

$$\langle f, g \rangle_{\mathcal{H}} = \sum_{i=1}^N \frac{f_i g_i}{\lambda_i}, \quad (2.12)$$

this Hilbert space is therefore equipped with a norm  $\|f\|_{\mathcal{H}}$  where  $\|f\|_{\mathcal{H}}^2 = \langle f, f \rangle_{\mathcal{H}} = \sum_{i=1}^N f_i^2 / \lambda_i$ .

This Hilbert space should be proven to be the RKHS corresponding to the kernel  $k$ , i.e. whether it has aspect of reproducing. It is readily attained as

$$\langle f(\cdot), k(\cdot, \mathbf{x}) \rangle_{\mathcal{H}} = \sum_{i=1}^N \frac{f_i \lambda_i \phi_i(\mathbf{x})}{\lambda_i} = f(\mathbf{x}). \quad (2.13)$$

Similarly,

$$\langle k(\mathbf{x}, \cdot), k(\mathbf{x}', \cdot) \rangle_{\mathcal{H}} = \sum_{i=1}^N \frac{\lambda_i \phi_i(\mathbf{x}) \lambda_i \phi_i(\mathbf{x}')}{\lambda_i} = k(\mathbf{x}, \mathbf{x}'). \quad (2.14)$$

Note that  $k(\mathbf{x}, \cdot)$  is in the RKHS in that it has the norm  $\sum_{i=1}^N (\lambda_i \phi_i(\mathbf{x}))^2 / \lambda_i = k(\mathbf{x}, \mathbf{x}) < \infty$ . We have now ascertained that the Hilbert space consists of linear combinations of the eigenfunctions under  $\sum_{i=1}^N f_i^2 / \lambda_i < \infty$  accomplished two conditions presumed in Definition 2.3.2. This Hilbert space is undoubtedly that RKHS by the fact that RKHS is to be unique with  $k(\cdot, \cdot)$ .

## 2.3.4 Eigenfunctions of Kernels

Bayesian linear regression bearing infinite number of basis functions can be considered as GP regression. Eigenfunctions of the covariance function is one of possible basis sets. A function  $\phi(\cdot)$  that complies the integral equation

$$\int k(\mathbf{x}, \mathbf{x}') \phi(\mathbf{x}) d\mu(\mathbf{x}) = \lambda \phi(\mathbf{x}'), \quad (2.15)$$

is specified as an eigenfunction of kernel  $k$  with eigenvalue  $\lambda$  with respect to measure  $\mu$ . The two particular measures of our interest will be (i) Lebesgue measure over a compact subset  $\mathcal{C}$  of  $\mathbb{R}^D$ , or (ii) when there is a density  $p(\mathbf{x})$  so that  $d\mu(\mathbf{x})$  can be written  $p(\mathbf{x}) d\mathbf{x}$ .

It would not be uncommon if we would label infinite number of eigenfunctions as  $\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots$  of which the order is determined in accordance with condition such that  $\lambda_1 \geq \lambda_2 \geq \dots$ . These eigenfunctions are orthogonal with respect to  $\mu$ . Eigenfunctions can be chosen to be normalized so that  $\int \phi_i(\mathbf{x})\phi_j(\mathbf{x})d\mu(\mathbf{x}) = \delta_{ij}$  where  $\delta_{ij}$  is the Kronecker delta.

Mercer's theorem (see, e.g. [König, 1986]) allows us to express the kernel  $k$  in terms of the eigenvalues and eigenfunctions.

**Theorem 2.3.2** (Mercer's theorem). *Let  $(\mathcal{X}, \mu)$  be a finite measure space and  $k \in L_\infty(\mathcal{X}^2, \mu^2)$  be a kernel such that  $T_k : L_2(\mathcal{X}, \mu) \rightarrow L_2(\mathcal{X}, \mu)$  is positive definite (see Equation (2.11)). Let  $\phi_i \in L_2(\mathcal{X}, \mu)$  be the normalized eigenfunctions of  $T_k$  associated with the eigenvalues  $\lambda_i > 0$ . Then:*

1. *the eigenvalues  $\{\lambda_i\}_{i=1}^\infty$  are absolutely summable*

2.

$$k(\mathbf{x}, \mathbf{x}') = \sum_{i=1}^{\infty} \lambda_i \phi_i(\mathbf{x}) \phi_i^*(\mathbf{x}'), \quad (2.16)$$

*holds  $\mu^2$  almost everywhere, where the series converges absolutely and uniformly  $\mu^2$  almost everywhere.*

This decomposition is just the infinite-dimensional analogue of the diagonalization of a Hermitian matrix. The sum may either be infinite or terminate at some value  $N \in \mathbb{N}$  (i.e. the eigenvalues beyond  $N$  are zero).

## 2.4 Deep Neural Networks

A DNN is the stacked transformation functions as follows.

$$F(\mathbf{x}) = f_L(f_{L-1}(\dots f_1(\mathbf{x}))), \quad (2.17)$$

where  $\mathbf{x}$  represents inputs of given data  $\{y, \mathbf{x}\}$ ,  $f_i$  represents of transformation by  $i^{th}$  layer.  $F$  stands for the DNN which is an approximation of  $y$  that we want to find as  $y \approx F(\mathbf{x})$ .

Typically,  $i^{th}$  layer contains  $\mathbf{W}_i$  matrix as a weight matrix and  $\mathbf{b}_i$  as a bias term.  $g(\cdot)$  represents an activation function. Conventionally, the activation function can be chosen among the logistic sigmoid ( $\frac{1}{1+e^{-x}}$ ), the tanh, and the rectified linear unit (ReLU) ( $\max(0, x)$ ) [Glorot et al., 2011]. Then we can

represent equations of layers; from the first layer to the output layer as follows,

$$\begin{aligned}
 f_1(\mathbf{x}) &= g(\mathbf{W}_1\mathbf{x} + \mathbf{b}_1), \\
 f_i(\mathbf{x}) &= g(\mathbf{W}_i(f_{i-1}(\dots f_1(\mathbf{x})) + \mathbf{b}_i), \\
 \phi(\mathbf{x}) &= g(\mathbf{W}_{L-1}(f_{L-2}(\dots f_1(\mathbf{x})) + \mathbf{b}_{L-1}), \\
 F(\mathbf{x}) &= \phi(\mathbf{x})^T \mathbf{w}_L + \mathbf{b}_L.
 \end{aligned}
 \tag{2.18}$$

The hidden layers  $f_i(\cdot)$  are the layers between the first layer  $f_1(\cdot)$  and output  $F(\cdot)$ . Thus,  $\phi(\mathbf{x})$  is the output of the last hidden layer. In this paper, we called given DNN as the fully-connected NNs. Since each of  $i$ -th layers contains a weight matrix  $\mathbf{W}_i$  and it denotes the matrix multiplication.

Solving regression problems, the activation function of the output layer  $F(\cdot)$  is a linear activation function  $g(\mathbf{x}) = \mathbf{x}$ . Therefore, there is a linear relationship between the last hidden layer and the output as  $F(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}_L + \mathbf{b}_L$ .

## 2.4.1 Convolutional Neural Networks

The CNNs are commonly composed with the convolutional layer, pooling layer and fully-connected layer. The main difference between the convolutional layer and fully-connected layer is the receptive field. The receptive field has restriction in terms of the filter (or kernel) size. Only the local regions of the input computed a dot product with the filters of convolutional layer. Then the filters are shared along the width or height of the input volume. The number of channels in output from the convolutional layer is depend on the number of kernels which are included in the convolutional layer.

Conventional CNNs are composed with the convolutional layer and the pooling layer. The pooling layer reduces the dimension of output from the convolutional layer. The max pooling is often used recently, which has been shown to work better in practice. The largest activated value from the convolutional layer is selected from the max pooling operation. Finally, the fully-connected layer (same with DNNs) is connected between the output value of model and extracted features from previous hidden layers of CNN.

## 2.4.2 Long Short-Term Memory Networks

Recurrent Neural Networks (RNNs) are not capable of handling long-term dependencies in practice [Bengio et al., 1994]. To overcome the difficulty of learning long-term dependencies with gradient de-

scent, LSTM networks were introduced by [Hochreiter and Schmidhuber, 1997].

A common LSTM unit is composed of a cell, an input gate, an output gate and a forget gate. The forget gate  $f_t$  decides what information to remember or remove from hidden state  $h_{t-1}$  and input  $x_t$ ,

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f). \quad (2.19)$$

The next step is to decide what new information to be stored in cell state. The input gate  $i_t$  decides which values be updated by a sigmoid layer output from previous hidden state and input value. A tanh layer creates a new candidate value  $\tilde{C}_t$  from hidden state and input.

$$\begin{aligned} i_t &= \sigma(W_i \cdot [h_{t-1}, x_t] + b_i), \\ \tilde{C}_t &= \tanh(W_C \cdot [h_{t-1}, x_t] + b_C). \end{aligned} \quad (2.20)$$

The output from input gate and the tanh layer are multiplied before cell state will be updated. Then, to update the old cell state  $C_{t-1}$ , the old cell state is multiplied with output of forget gate  $f_t$  and added into the multiplication of the output from input gate  $i_t$  and the tanh layer  $\tilde{C}_t$ ,

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t. \quad (2.21)$$

After update the cell state  $C_t$ , the output gate  $o_t$  will generate the new hidden state (or output)  $h_t$  of LSTM. First, output from hidden state and input go through a sigmoid layer to make a decision of output value  $o_t$ . Then, output from a tanh layer of the new cell state is multiplied with the decision from the sigmoid layer,

$$\begin{aligned} o_t &= \sigma(W_o \cdot [h_{t-1}, x_t] + b_o), \\ h_t &= o_t * \tanh(C_t). \end{aligned} \quad (2.22)$$



## Chapter 3

# Related Work

Relationship between GPs and DNNs are introduced by [Neal, 1995]. If a fully-connected NN is consisted of a single-layer which are the infinite number of hidden units with an i.i.d. prior over its parameters, then the DNN is equivalent to a GP. He proved that a one-hidden layer NN equipped with infinite number of hidden units becomes a nonparametric GP model by placing independent zero-mean Gaussian priors to all the weights of the network and integrating them out.

[Neal, 1995] further suggested that a similar correspondence might hold for deeper networks. Thus, [Lee et al., 2017] derive the exact equivalence between infinitely wide deep networks and GPs. They also derive the GP kernel for multi-hidden-layer NN with general non linearity based on signal propagation theory [Cho and Saul, 2009], called as NNGP

Similar approach with our proposed model, [Huang et al., 2015] proposed a scalable GP model for regression by applying a DNN as the feature-mapping function. The DNN is pre-trained as a stacked denoising auto-encoder in an unsupervised way. Then, a Bayesian linear regression is performed with the last hidden layer of the pre-trained deep network. The resulting model, Deep-Neural-Network-based Gaussian Process (DNN-GP), can learn much more meaningful representation of the data by the finite-dimensional but deep-layered feature-mapping function. Unlike standard GPs, their proposed model scales well with the size of the training set due to the avoidance of kernel matrix inversion. Also, as research done by recently, the kernel perspective can be adopted for regularizing DNNs [Bietti et al., 2019]. They propose a new point of view for regularizing DNNs by using the norm of a RKHS.

## Chapter 4

# Temporal Covariance Loss

In Chapter 2, we have introduced the relationship between the GPs and DNNs briefly. GP can model smooth functions, including smooth time series, well [Williams and Rasmussen, 2006]. We will exploit the fundamental principles of GP, by representing the empirical covariance of the target by the feature map, the second last layer, of DNNs. We regard the feature map as basis functions in the GP kernel.

In this chapter, we will show our proposed model, Temporal Covariance loss function. To solve a regression problem  $\{y|\mathbf{x}\}$ , a conventional DNN is trained to predict target data  $y$ . However, our goal is to provide the opportunity to the DNN model to reflect the covariance of the target  $\text{cov}(y, y')$  during the training procedure.

### 4.1 Learning the Basis Functions in Deep Neural Networks

During the training of a DNN, the network receives a batch of input data from the input dataset. Then, the loss function obtains the gradient of batch data and employs back-propagation algorithm to update the weight of networks. In DNNs, there are various types of layers which are stacked with activation functions such as sigmoid. Although there are many variations of network design, in the regression problem, the last hidden layer (before the output layer) is mainly constructed by fully-connected layers in the DNNs as described in section 2.4. The relation between the last hidden layer and output  $\hat{y}$  can be represented by the linear regression problem. Figure 4.1 shows the general structure of a DNN.

In the linear relation between the last hidden and the output layer, the last hidden layer can be considered as the encoded vector of input data from the DNNs. Thus, we are able to utilize the properties of linear regression.

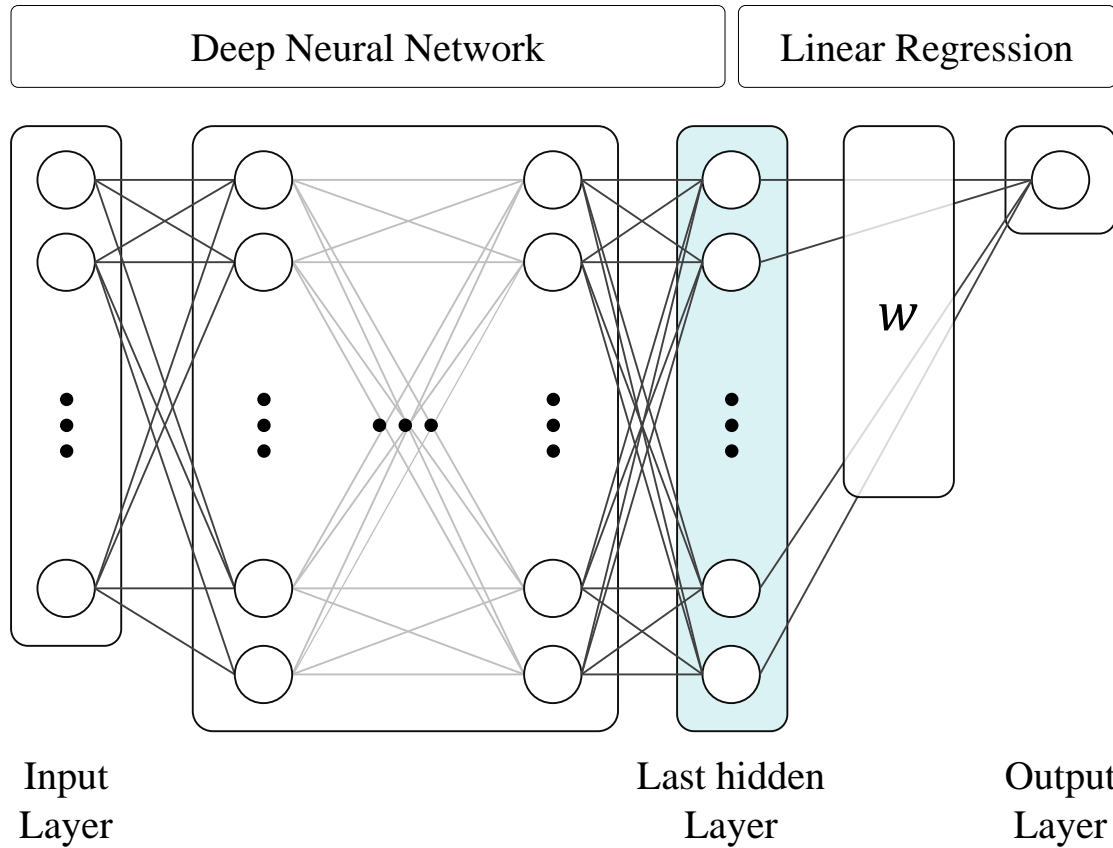


Figure 4.1: An architecture of a DNN model for regression tasks. There are various types of layers such as convolutional or fully-connected layers in the DNNs (grayed line in the figure). When we consider the blue colored layer as the last layer of DNN, the relation between the last hidden and output layer can be represented by the linear regression.

**Definition 4.1.1** (Basis function). Let the output of the last hidden layer be  $\phi(\mathbf{x})$ . Then the relation between  $\phi(\mathbf{x})$  and output  $f(\mathbf{x})$  is represented as linear regression form  $f(\mathbf{x}) = \phi(\mathbf{x})^T \mathbf{w}$ . We can say that the output of the last hidden layer  $\phi(\mathbf{x})$  is same as the basis function expansion for the linear regression.

Assuming the targets of DNNs follow a GP, we have the mean and covariance with prior  $\mathbf{w} \sim \mathcal{N}(0, \sigma I)$  as,

$$\begin{aligned} \mathbb{E}[f(\mathbf{x})] &= \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}] = 0, \\ \mathbb{E}[f(\mathbf{x})f(\mathbf{x}')] &= \phi(\mathbf{x})^T \mathbb{E}[\mathbf{w}\mathbf{w}^T] \phi(\mathbf{x}') = \sigma \phi(\mathbf{x})^T \phi(\mathbf{x}'). \end{aligned} \tag{4.1}$$

Thus  $f(\mathbf{x})$  and  $f(\mathbf{x}')$  are jointly Gaussian with the zero mean and covariance matrix given by  $\sigma\phi(\mathbf{x})^T\phi(\mathbf{x}')$ . Indeed, the function values  $f(\mathbf{x}_1), \dots, f(\mathbf{x}_n)$  corresponding to any number of input points  $n$  are jointly Gaussian. The covariance matrix is the kernel matrix where the basis functions are trained by the DNN.

## 4.2 Temporal Covariance Loss

**Definition 4.2.1** (Temporal Covariance). Where the length of time series data is  $N$ . The number of  $m$  data points  $(\mathbf{x}_1, y_1), \dots, (\mathbf{x}_m, y_m)$  are selected as a batch  $(\mathbf{x}_i, y_i)$  is the  $e_i$ -th data point in the time series data for randomly selected  $e_i \in [1, N]$ . Then the Temporal Covariance  $\tilde{\Sigma}_y$  is defined as

$$\tilde{\Sigma}_y = \widetilde{\text{cov}}[y] = \frac{1}{m}(\mathbf{y} - \bar{\mathbf{y}})(\mathbf{y} - \bar{\mathbf{y}})^T. \quad (4.2)$$

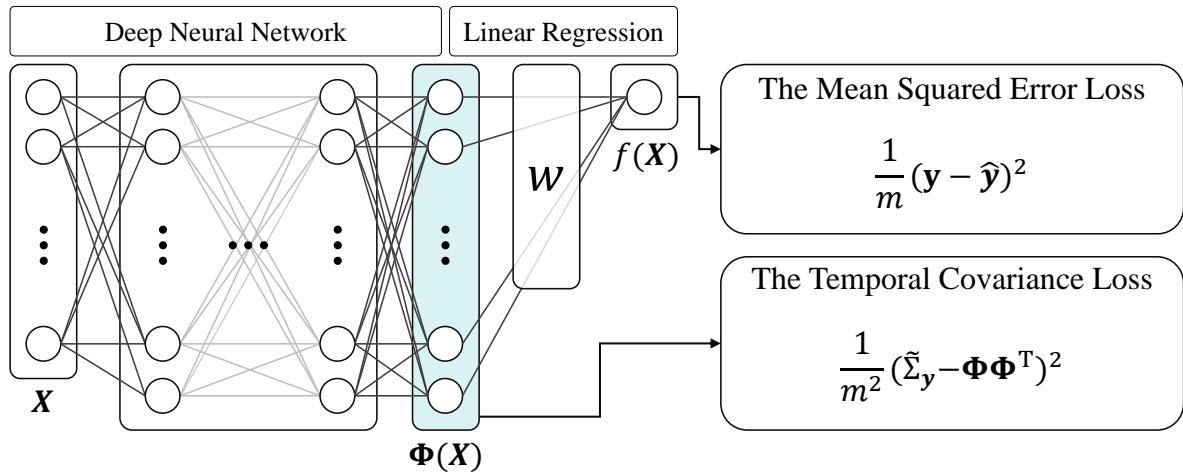


Figure 4.2: The proposed structure of DNN with the Temporal Covariance loss function. The DNN is trained to reduce the MSE loss between the Temporal Covariance and the outer product of the basis function matrix.

To utilize the Temporal Covariance of target data while training the DNNs, we propose a loss function which is calculated from the residual between a kernel matrix of the DNNs and the temporal covariance matrix. The networks utilize the MSE loss function which is defined as Equation 4.3. After calculating the MSE, the error is back-propagated to the networks.

$$\text{MSE} = \frac{1}{k} \sum_{i=1}^k (a_k - b_k)^2, \quad (4.3)$$

when it comes to matrices  $\mathbf{A}, \mathbf{B} \in \mathbb{R}^{m \times n}$  to compute the MSE where  $a_k, b_k$  are the elements of them, the MSE loss compares the given matrices element-wise.

Each of the basis functions  $\phi_k(\mathbf{x})$  is one of the nodes in the last hidden layer of networks. A set of basis functions  $\phi(\mathbf{x}) = [\phi_1(\mathbf{x}), \phi_2(\mathbf{x}), \dots, \phi_h(\mathbf{x})]^T$  is composed of a vector with the number of nodes  $h$  in the last hidden layer (the number of basis functions that DNN learns during training phase). Then the inner product of basis functions can be written as  $\phi(\mathbf{x}_i)\phi(\mathbf{x}_j) = \sum_{k=1}^h \phi_k(\mathbf{x}_i)\phi_k(\mathbf{x}_j)$ . We further denote the basis function matrix which is stacked with the evaluated basis functions in a batch of size  $m$  as  $\Phi(\mathbf{X}) = [\phi(\mathbf{x}_1), \phi(\mathbf{x}_2), \dots, \phi(\mathbf{x}_m)]^T$ . Then we have the Temporal Covariance loss as:

**Definition 4.2.2.** (Temporal Covariance Loss) With data  $(\mathbf{X}, \mathbf{y})$  as defined in the Definition 4.2.1, we define the Temporal Covariance loss as follows,

$$\begin{aligned} \text{MSE}(\tilde{\Sigma}_{\mathbf{y}}, \Phi\Phi^T) &= \frac{1}{m^2} (\widetilde{\text{cov}}[\mathbf{y}] - \Phi(\mathbf{X})\Phi(\mathbf{X})^T)^2 \\ &= \frac{1}{m^2} \sum_{i=1}^m \sum_{j=1}^m \left( \frac{1}{m} (y_i - \bar{y})(y_j - \bar{y}) - \phi(\mathbf{x}_i)\phi(\mathbf{x}_j) \right)^2. \end{aligned} \quad (4.4)$$

Finally, we add the Temporal Covariance loss into the general MSE loss of the DNN with the hyper-parameter  $\lambda$  which can regulate the importance of the Temporal Covariance loss. Where  $\hat{\mathbf{y}}$  is the prediction output from the model. Therefore, the total loss function of the proposed DNN is as follows,

$$\frac{1}{m} (\mathbf{y} - \hat{\mathbf{y}})^2 + \lambda \left( \frac{1}{m^2} (\tilde{\Sigma}_{\mathbf{y}} - \Phi\Phi^T)^2 \right). \quad (4.5)$$

### 4.3 Principles of a Temporal Covariance Loss Based Neural Networks

The Temporal Covariance loss function proposed earlier aims to find the basis functions which have similar properties with functions in RKHS of the covariance kernel function of the target  $\mathbf{y}$ . Therefore, we wish to guide the basis functions of DNNs with the following principles.

If we assume that the given target vector as  $\mathbf{y} \sim \mathcal{GP}(\mu, \Sigma)$  and training DNNs with that given data, then the MSE loss can be regarded as guiding  $\hat{\mathbf{y}}$  to  $\mu$  during the training phase as it reduces  $\frac{1}{m} (\mathbf{y} - \hat{\mathbf{y}})^2$ . By contrast, we expect that the basis functions can represent the variance and covariance of  $\mathbf{y}$  with our proposed loss which measures the difference between the Temporal Covariance  $\tilde{\Sigma}_{\mathbf{y}}$  and regenerated kernel  $\Phi\Phi^T$ .

The advantages of our proposed model are twofold. The first advantage of our proposed loss function is that it is straightforward in implementation. There had been research which try to connect GP with

DNNs. For example, [Lee et al., 2017] (NNGP) shows that the exact equivalence between infinitely wide deep networks and GPs. Unlike previous studies, our proposed loss function operates on the Networks and no need to the derivation from the covariance functions or matrices of the GP. Also note that we do not set prior of the given data pair  $\{y, \mathbf{x}\}$  to find the basis functions. Rather than changing DNNs architecture or model itself, the covariance loss function can be attached to the conventional loss functions with the hyper-parameter during training phase.

Second, the DNN parts of our proposed model can be replaced by other NNs architecture (e.g. CNNs). If the last hidden layer of replaced NNs architecture is in a vector form, then we can derive the regenerated kernel to get the Temporal Covariance loss.

In principle, our Temporal Covariance loss based learning impose the DNN trains on top of MSE loss of the target  $y$ . That is, whenever two outputs  $y_i$  and  $y_j$  of data points  $\mathbf{x}_i$  and  $\mathbf{x}_j$  have high positive (negative) covariance value, we make the inner product of feature maps  $\phi(\mathbf{x}_i)$  and  $\phi(\mathbf{x}_j)$  has high positive (negative) value. When they are uncorrelated (covariance is set to zero), the inner product of feature map  $\phi(\mathbf{x}_i)$  and  $\phi(\mathbf{x}_j)$  becomes zero (i.e. orthogonal). In this way, our model includes the important principles of GP, although our model does not explicitly model and implement GP.

## Chapter 5

# Experimental Results

In this section, our proposed loss function is merged with a conventional loss function of DNN. Two DNN models are chosen to compare the prediction accuracy for the two real-world datasets. The first model is fully-connected NN and the second baseline approach model is CNN. Both of DNN models can adopt our proposed loss function because of that we can design the last hidden layer as a vector form. Therefore, The difference between a baseline approach model and a proposed model is the existence of the Temporal Covariance loss function. The first real-world dataset is groundwater dataset which is collected from US territory and second real-world dataset is NASDAQ 100 stock dataset which is the collection of stock prices of corporations under NASDAQ 100.

### 5.1 Experimental Setup

The objective for experiment is a multivariate regression problem which is defined as Equation 5.1.

$$\begin{aligned}
 y_t &= \mathbf{x}_t^a, \quad \text{for some } a \in \{1, 2, \dots, n\} \text{ at time } t \\
 \hat{y}_t &= F(y_{t-1}, \dots, \mathbf{x}_{t-l}^1, \mathbf{x}_{t-l}^2, \dots, \mathbf{x}_{t-l}^n),
 \end{aligned} \tag{5.1}$$

where  $y_t$  is a target data,  $a$  is a randomly selected number from 1 to the number of columns (features) from given data,  $n$  as an input dataset.  $\hat{y}_t$  is a prediction by function  $F$  which is approximated from our model. The model predict a target value as  $l$  times afterward from the input data.

The evaluation metric that is used to compare regression performance is the RMSE, which is defined as Equation 5.2,

$$\text{RMSE} = \sqrt{\mathbb{E}[(y - \hat{y})^2]} = \sqrt{\frac{1}{n} \sum_{i=1}^n (y_i - \hat{y}_i)^2}. \quad (5.2)$$

We divide the given dataset into training set, validation set and testing set taking 80%, 10% and 10% of the original data, respectively. During the training phase of the DNNs, we select the best performance in terms of the RMSE in the validation set. Accordingly, the best performance model is compared with other models in the testing set.

## 5.2 Groundwater Dataset

The groundwater dataset [Yi et al., 2017] provided by the United States Geological Survey (USGS). The dataset is daily collected from the US territories. We select the depth of groundwater levels for 28 years (1987-2015). If data sequences are not recorded longer than two months, then it were excluded from dataset. Also, empty records shorter than two months were filled using linear interpolation. After refined, The dataset contains 88 numbers of monitoring sensor values for 10,228 days. A model predicts after 10 days groundwater level of one station from the whole dataset.

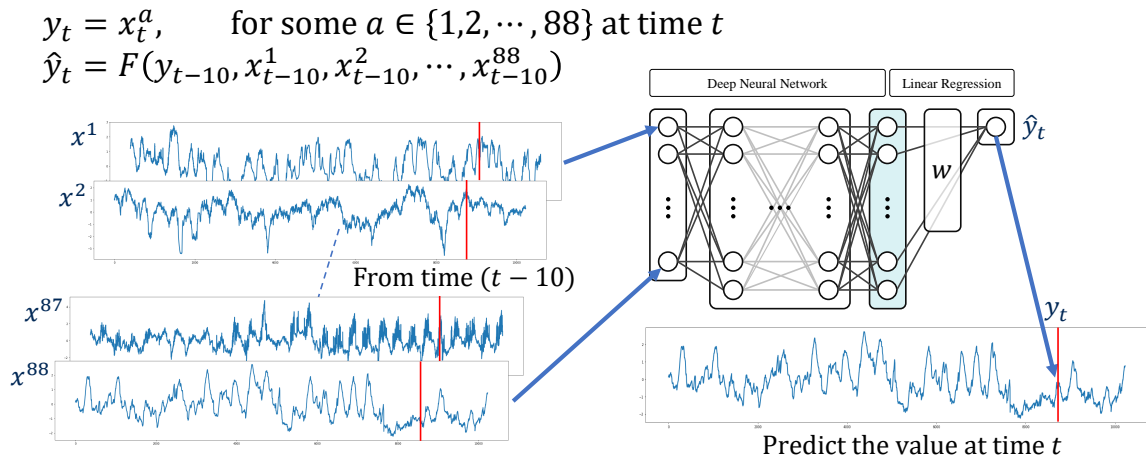


Figure 5.1: The details of experiments setting on the groundwater dataset with DNNs. Where target  $\mathbf{y}$  is chosen in given dataset  $\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^{88}\}$ . The purpose of the experiments is finding  $F(\cdot)$  by DNNs. The function  $F(\cdot)$  represents the relationship between  $\mathbf{y}$  and  $\{\mathbf{x}^1, \mathbf{x}^2, \dots, \mathbf{x}^{88}\}$  with given 10 days delayed.

While training a model, we select the best performance in terms of the RMSE for the validation set. The model with the best performance was compared with other models by given testing set.



Three different models are chosen to compare the performance with the proposed model; first one is a linear regression model, the second model is LSTM model and baseline approach fully-connected NN without the Temporal Covariance loss. For a fair comparison, the architecture of the proposed model is same as the DNN baseline model, but varying in the loss function containing the Temporal Covariance loss function. The number of nodes in each fully-connected layer is (88, 100, 100, 100, 1), three hidden layers consisted of 100 nodes. The activation function is exponential linear unit (ELU) activation [Clevert et al., 2015] except the output layer. Dropout [Srivastava et al., 2014] is applied on the first two hidden layers. The optimizer is Adam optimizer [Kingma and Ba, 2014] with learning rate as  $1e - 5$ . The model is trained with 10000 epochs. Each epoch contains randomly permuted training set with batch size 1024.

For the case of comparisons with CNN models, also three different models are chosen to compare the performance with the proposed model; first one is a linear regression model, the second model is LSTM model and the baseline approach CNN without the Temporal Covariance loss. For a fair comparison, the architecture of the proposed model is same as the CNN baseline model, but varying in the loss function containing the Temporal Covariance loss function. The length of input time series are 10 and the number of channels output of each convolutional layer is (100, 100, 100) with kernel size as 4. Without padding 0 values each input of layer, the output length of the final convolutional layer is 1. Accordingly, the number of nodes (or channel) in the last hidden layer is 100, same as DNN experiments. The activation function is ELU activation except the output layer. Dropout is applied on the first two convolutional layers. The optimizer is Adam optimizer with learning rate as  $1e - 6$ . The model is trained with 10000 epochs. Each epoch contains randomly permuted training set with batch size 2048.

### 5.2.1 Quantitative Analysis

We design experiments for comparing our proposed model with four baseline approach models. We randomly select 10 numbers for target data among  $\{0, 1, \dots, 87\}$ . The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.1 and Figure 5.2 (box-plot).

Table 5.1: Comparisons the mean of RMSE between the linear regression, the LSTM model, the baseline approach DNN, CNN model and the proposed models on the number of 10 randomly selected groundwater dataset. The baseline approach DNN, CNN model does not contain the Temporal Covariance loss and the proposed models with hyper-parameter 0.1 are denoted as “Proposed DNN model” and “Proposed CNN model”. A **bold** font indicates the best result obtained.

Target number	0	17	23	64	66	70	75	78	83	84
Linear Regression	0.3692	0.6413	0.7802	0.9927	0.7993	0.8232	0.7743	0.3995	0.9387	0.4431
LSTM model	0.5459	0.6309	0.8330	1.0228	1.3467	0.6536	0.7892	0.8332	0.9414	0.5293
Baseline DNN model	0.4511	0.3287	0.7038	0.9476	0.8351	0.5007	0.7707	0.5007	0.5449	<b>0.2622</b>
Baseline CNN model	0.4362	0.3629	0.7367	0.8879	0.8927	0.6011	0.7398	0.5323	0.6608	0.2997
Proposed DNN model	<b>0.3234</b>	<b>0.2666</b>	<b>0.5930</b>	0.7865	<b>0.6614</b>	<b>0.4814</b>	0.7092	<b>0.3882</b>	<b>0.4487</b>	0.2636
Proposed CNN model	0.4083	0.3475	0.6593	<b>0.7803</b>	0.8041	0.5573	<b>0.7029</b>	0.4720	0.6067	0.3946

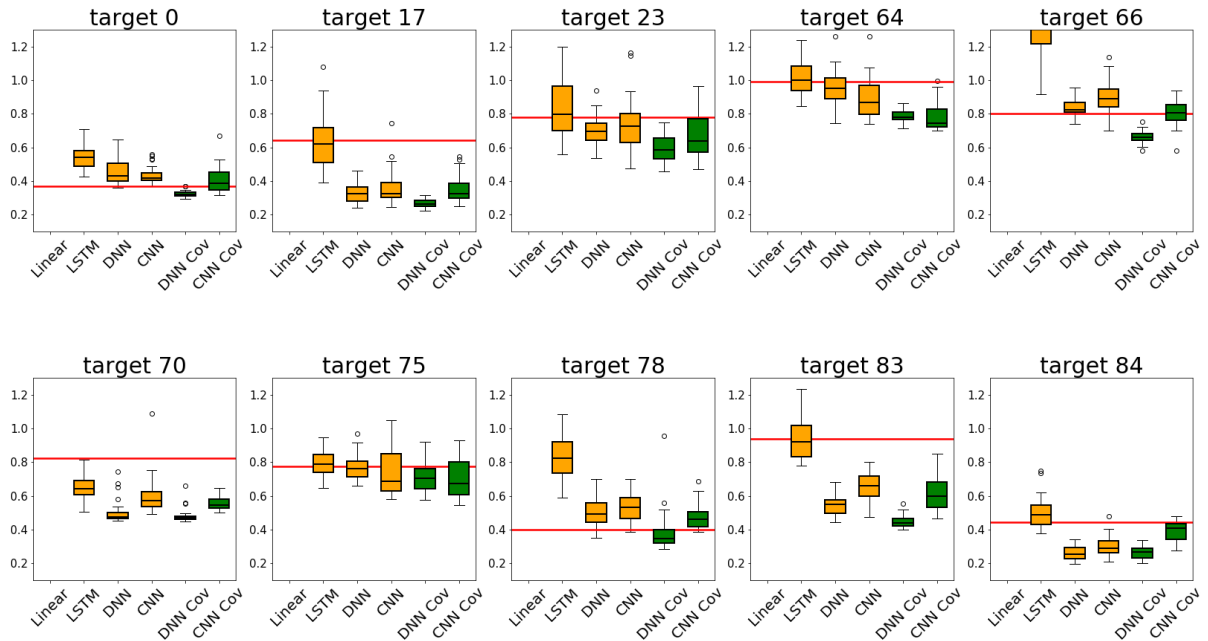


Figure 5.2: Comparisons the RMSE of proposed models with four baseline approach models. The  $y$ -axis is RMSE of each model and the linear regression as “Linear” and the horizontal red line, the baseline approach models are denoted as yellow color, the proposed models with hyper-parameter 0.1 are labeled by “DNN Cov” and “CNN Cov” with green color.

### 5.2.2 Quantitative Analysis with Fully Connected Neural Networks

We design experiments for comparing our proposed model with the baseline approach DNN model. We randomly select 10 numbers for target data among  $\{0, 1, \dots, 87\}$ . The first baseline model is the fully-connected NN. The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.2 and Figure 5.3 (box-plot).

Table 5.2: Comparisons the mean of RMSE between the linear regression, the baseline approach DNN model and the proposed model on the number of 10 randomly selected groundwater dataset. The baseline approach DNN model does not contain the Temporal Covariance loss and the proposed model with hyper-parameter 0.1 is denoted as “Proposed with  $\lambda$  0.1”. A **bold** font indicates the best result obtained.

Target number	0	17	23	64	66	70	75	78	83	84
Linear Regression	0.3692	0.6413	0.7802	0.9927	0.7993	0.8232	0.7743	0.3995	0.9387	0.4431
Baseline DNN model	0.4511	0.3287	0.7038	0.9476	0.8351	0.5007	0.7707	0.5007	0.5449	<b>0.2622</b>
Proposed with $\lambda$ 0.1	<b>0.3234</b>	<b>0.2666</b>	<b>0.5930</b>	<b>0.7865</b>	<b>0.6614</b>	<b>0.4814</b>	<b>0.7092</b>	<b>0.3882</b>	<b>0.4487</b>	0.2636

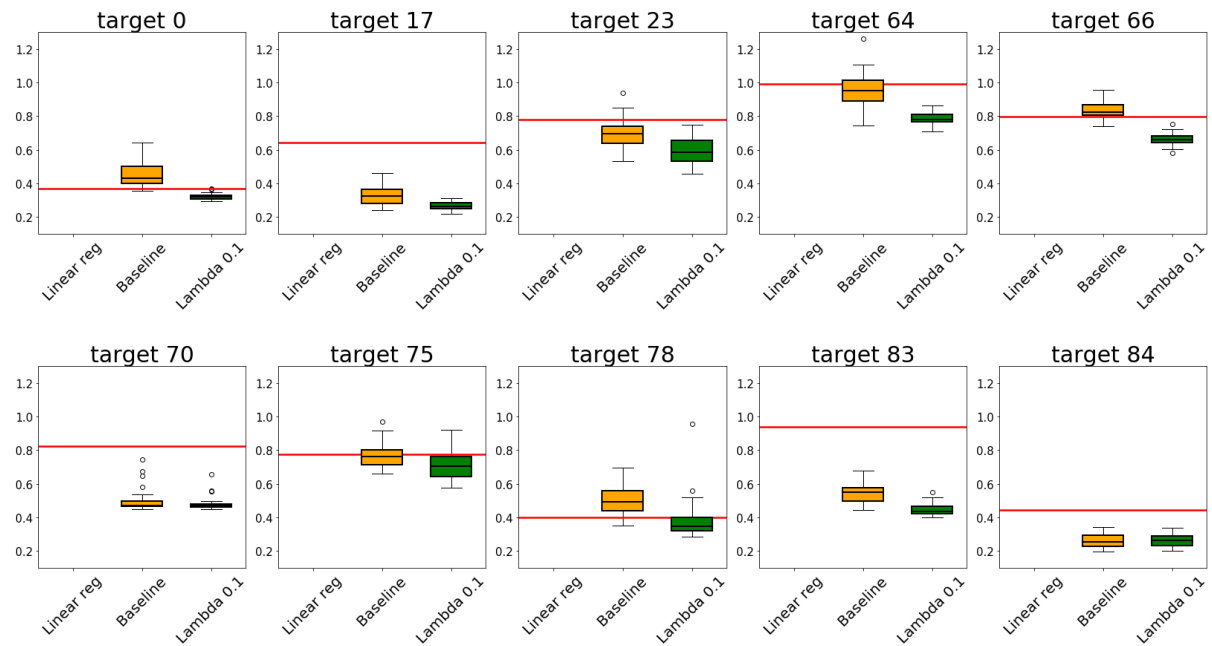


Figure 5.3: Comparisons the RMSE between the linear regression, the baseline approach DNN model and the proposed model on the number of 10 randomly selected groundwater dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the horizontal red line, the baseline approach DNN model is denoted “Baseline” and the color is yellow, the proposed model with hyper-parameter 0.1 is labeled by “Lambda 0.1” with green color.

Also, We design experiments for comparing our proposed model with three different hyper-parameters  $\lambda$  as  $\{0.1, 1, 10\}$  for randomly select 10 numbers for target data among  $\{0, 1, \dots, 87\}$ . The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.3 and Figure 5.4 (box-plot).

Table 5.3: Comparisons the mean of RMSE for the proposed DNN model with three different hyper-parameters  $\lambda$  on the number of 10 randomly selected groundwater dataset. The proposed DNN model with different hyper-parameters  $\{0.1, 1, 10\}$  is denoted as “Proposed with  $\lambda \{0.1, 1, 10\}$ ”. A **bold font** indicates the best result obtained.

Target number	0	17	23	64	66	70	75	78	83	84
Proposed with $\lambda$ 0.1	0.3234	<b>0.2666</b>	<b>0.5930</b>	0.7865	<b>0.6614</b>	0.4814	0.7092	0.3882	<b>0.4487</b>	<b>0.2636</b>
Proposed with $\lambda$ 1	0.3222	0.2674	0.6452	0.7574	0.6705	<b>0.4777</b>	0.6646	0.3486	0.4566	0.2992
Proposed with $\lambda$ 10	<b>0.3154</b>	0.3206	0.6297	<b>0.7503</b>	0.7028	0.4803	<b>0.6613</b>	<b>0.3075</b>	0.4744	0.3379

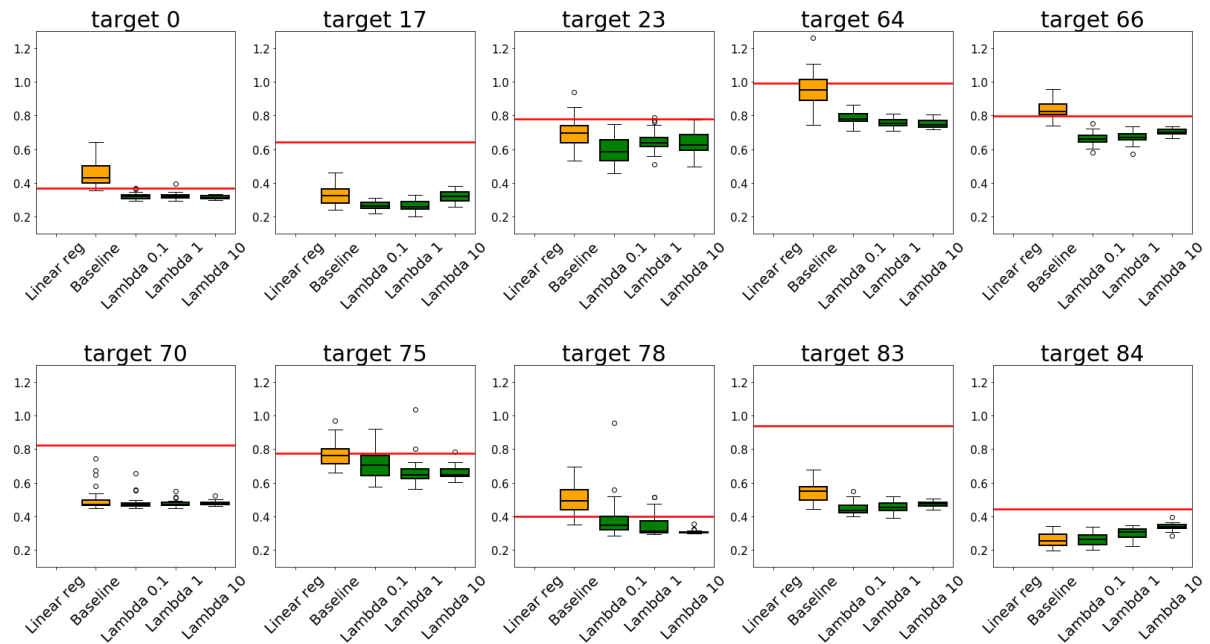


Figure 5.4: Comparisons the RMSE of the proposed DNN model with three different hyper-parameters on the number of 10 randomly selected groundwater dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the red horizontal line, the baseline approach model as “Baseline” and the color is yellow, and the proposed model with different hyper-parameters  $\{0.1, 1, 10\}$  as “Lambda  $\{0.1, 1, 10\}$ ” with green color.

### 5.2.3 Quantitative Analysis with Convolutional Neural Networks

We design experiments for comparing our proposed model with the baseline approach CNN model. We randomly select 10 numbers for target data among  $\{0, 1, \dots, 87\}$ . The second baseline model is CNN. The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.4 and Figure 5.5 (box-plot).

Table 5.4: Comparisons the mean of RMSE between the linear regression, the baseline approach CNN model and the proposed model on the number of 10 randomly selected groundwater dataset. The baseline approach CNN model does not contain the Temporal Covariance loss and the proposed model with hyper-parameter 0.1 is denoted as “Proposed with  $\lambda$  0.1”. A **bold** font indicates the best result obtained.

Target number	0	17	23	64	66	70	75	78	83	84
Linear Regression	<b>0.3692</b>	0.6413	0.7802	0.9927	<b>0.7993</b>	0.8232	0.7743	<b>0.3995</b>	0.9387	0.4431
Baseline CNN model	0.4362	0.3629	0.7367	0.8879	0.8927	0.6011	0.7398	0.5323	0.6608	<b>0.2997</b>
Proposed with $\lambda$ 0.1	0.4083	<b>0.3475</b>	<b>0.6593</b>	<b>0.7803</b>	0.8041	<b>0.5573</b>	<b>0.7029</b>	0.4720	<b>0.6067</b>	0.3946

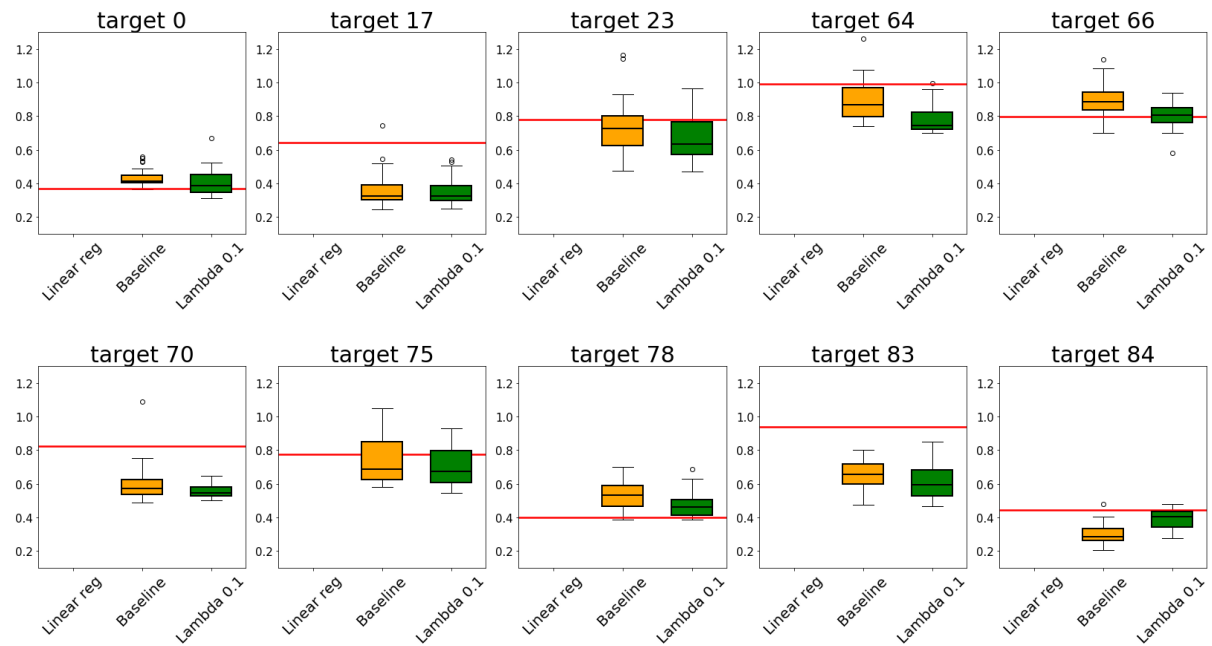


Figure 5.5: Comparisons the RMSE between the linear regression, the baseline approach CNN model and the proposed model on the number of 10 randomly selected groundwater dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the horizontal red line, the baseline approach CNN model is denoted “Baseline” and the color is yellow, the proposed model with hyper-parameter 0.1 is labeled by “Lambda 0.1” with green color.

Also, We design experiments for comparing our proposed model with three different hyper-parameters  $\lambda$  as  $\{0.1, 1, 10\}$  for randomly select 10 numbers for target data among  $\{0, 1, \dots, 87\}$ . The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.5 and Figure 5.6 (box-plot).

Table 5.5: Comparisons the mean of RMSE for the proposed CNN model with three different hyper-parameters  $\lambda$  on the number of 10 randomly selected groundwater dataset. The proposed CNN model with different hyper-parameters  $\{0.1, 1, 10\}$  is denoted as “Proposed with  $\lambda \{0.1, 1, 10\}$ ”. A **bold font** indicates the best result obtained.

Target number	0	17	23	64	66	70	75	78	83	84
Proposed with $\lambda$ 0.1	<b>0.4083</b>	<b>0.3475</b>	<b>0.6593</b>	0.7803	<b>0.8041</b>	<b>0.5573</b>	0.7029	0.4720	<b>0.6067</b>	0.3946
Proposed with $\lambda$ 1	0.4200	0.3830	0.7207	<b>0.7751</b>	0.8389	0.5788	0.7239	<b>0.4664</b>	0.6080	<b>0.3938</b>
Proposed with $\lambda$ 10	0.4117	0.4097	0.6820	0.8058	0.8737	0.5626	<b>0.6929</b>	0.4817	0.6434	0.3766

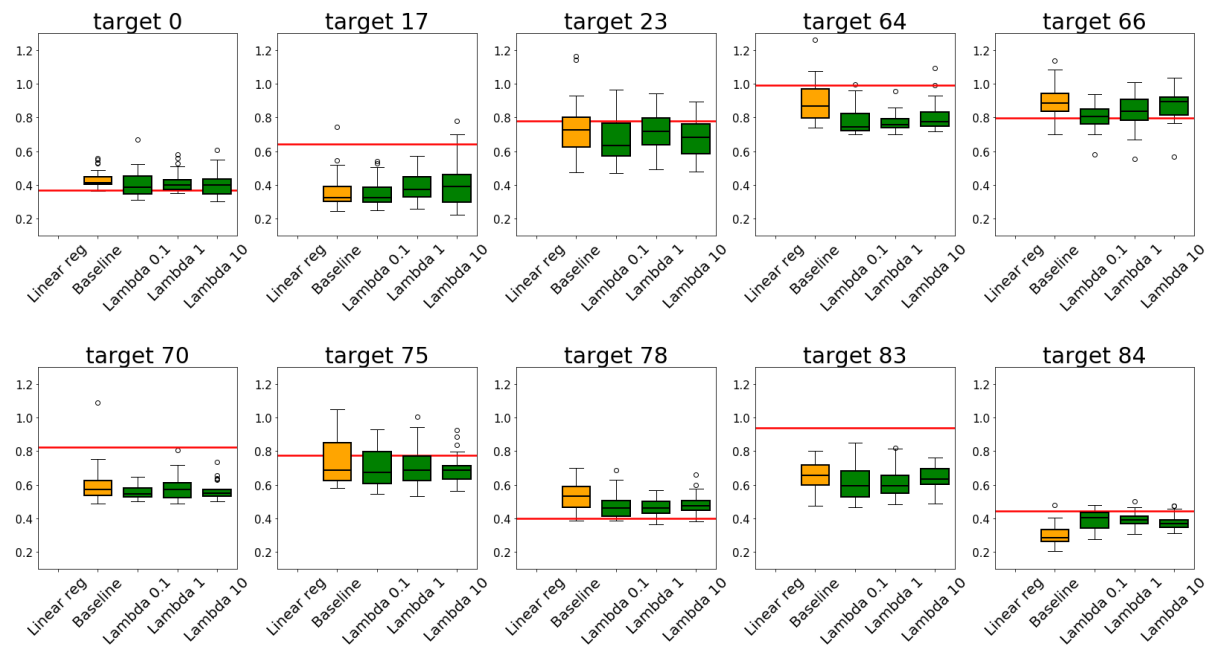


Figure 5.6: Comparisons the RMSE of the proposed CNN model with three different hyper-parameters on the number of 10 randomly selected groundwater dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the red horizontal line, the baseline approach model as “Baseline” and the color is yellow, and the proposed model with different hyper-parameters  $\{0.1, 1, 10\}$  as “Lambda  $\{0.1, 1, 10\}$ ” with green color.

### 5.3 Stock Market Dataset

The stock market dataset [Qin et al., 2017] is collected the stock prices of 81 companies are included in NASDAQ 100. The frequency of the time series is minute-by-minute. Stock prices are selected 105 days in total, from July 26, 2016 to December 22, 2016. The dataset contains 81 numbers of stock prices for 40,560 minutes. Each day contains 390 minutes from the opening to the closing of the market. A model predicts after 1 hour (60 minutes) stock price of one corporation from the whole dataset. While training a model, we select the best performance in terms of the RMSE for the validation set. The model with the best performance was compared with other models by given testing set.

Three different models are chosen to compare the performance with the proposed model; first one is a linear regression model, the second model is LSTM model and baseline approach fully-connected NN without the Temporal Covariance loss. For a fair comparison, the architecture of the proposed model is same as the DNN baseline model, but varying in the loss function containing the Temporal Covariance loss function. The number of nodes in each fully-connected layer is (81, 100, 100, 100, 1), three hidden layers consisted of 100 nodes. The activation function is ELU activation [Clevert et al., 2015] except the output layer. Dropout [Srivastava et al., 2014] is applied on the first two hidden layers. The optimizer is Adam optimizer [Kingma and Ba, 2014] with learning rate as  $1e - 6$ . The model is trained with 10000 epochs. Each epoch contains randomly permuted training set with batch size 2048.

For the case of comparisons with CNN models, also three different models are chosen to compare the performance with the proposed model; first one is a linear regression model, the second model is LSTM model and the baseline approach CNN without the Temporal Covariance loss. For a fair comparison, the architecture of the proposed model is same as the CNN baseline model, but varying in the loss function containing the Temporal Covariance loss function. The length of input time series are 10 and the number of channels output of each convolutional layer is (100, 100, 100) with kernel size as 4. Without padding 0 values each input of layer, the output length of the final convolutional layer is 1. Accordingly, the number of nodes (or channel) in the last hidden layer is 100, same as DNN experiments. The activation function is ELU activation except the output layer. Dropout is applied on the first two convolutional layers. The optimizer is Adam optimizer with learning rate as  $1e - 6$ . The model is trained with 10000 epochs. Each epoch contains randomly permuted training set with batch size 2048.

### 5.3.1 Quantitative Analysis

We design experiments for comparing our proposed model with four baseline approach models. We randomly select 10 numbers for target data among  $\{0, 1, \dots, 80\}$ . The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.6 and Figure 5.7 (box-plot).

Table 5.6: Comparisons the mean of RMSE between the linear regression, the LSTM model, the baseline approach DNN, CNN model and the proposed models on the number of 10 randomly selected stock market dataset. The baseline approach DNN, CNN model does not contain the Temporal Covariance loss and the proposed models with hyper-parameter 0.1 are denoted as “Proposed DNN model” and “Proposed CNN model”. A **bold** font indicates the best result obtained or the best result among NNs.

Target number	1	2	21	28	29	33	37	48	60	72
Linear Regression	<b>0.5806</b>	0.4507	0.5447	1.0298	<b>0.3065</b>	0.6369	<b>0.5755</b>	<b>0.5756</b>	<b>0.2560</b>	<b>0.6290</b>
LSTM model	1.1771	0.5259	0.9783	0.6275	0.4974	0.5680	1.7285	2.3761	0.6123	1.0957
Baseline DNN model	1.0456	0.4022	0.4824	0.5872	0.5518	0.3548	1.3726	1.8978	0.5004	0.7580
Baseline CNN model	1.1184	0.6348	0.8305	0.6379	0.4722	<b>0.3368</b>	<b>1.0778</b>	1.7003	0.4818	0.8638
Proposed DNN model	<b>1.0111</b>	<b>0.3386</b>	<b>0.3758</b>	<b>0.4736</b>	<b>0.4612</b>	0.3534	1.2810	<b>1.4763</b>	<b>0.4367</b>	<b>0.6493</b>
Proposed CNN model	1.0642	0.6121	0.5138	0.5271	0.4768	0.3641	1.1861	1.6525	0.5634	0.6675

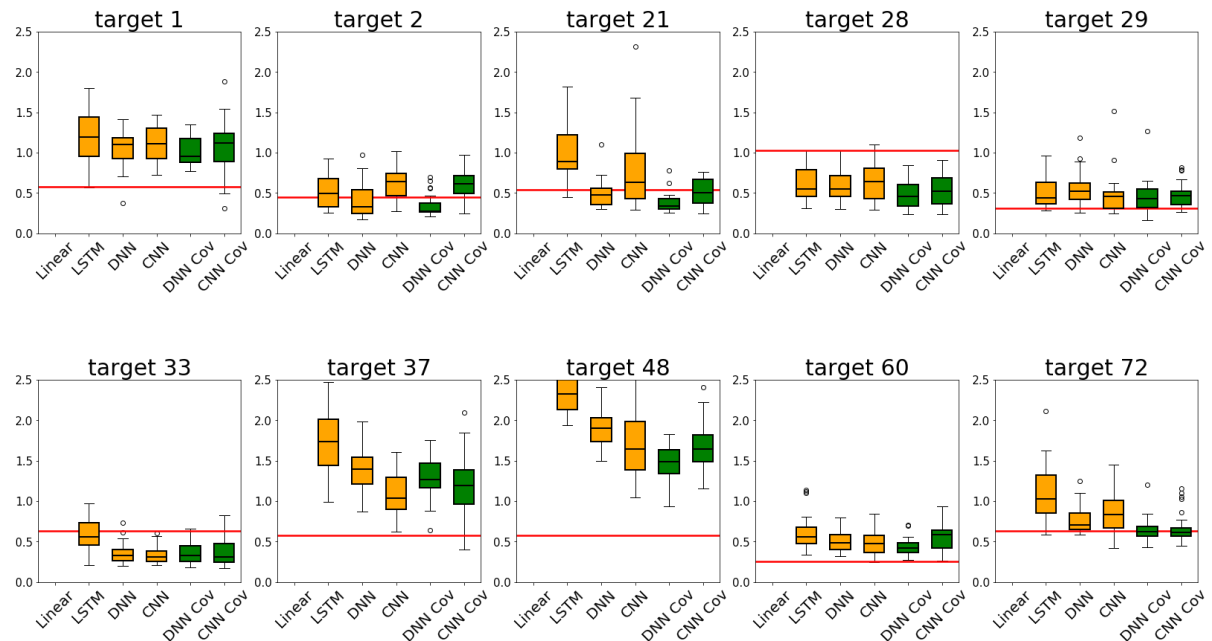


Figure 5.7: Comparisons the RMSE of proposed models with four baseline approach models. The  $y$ -axis is RMSE of each model and the linear regression as “Linear” and the horizontal red line, the baseline approach models are denoted as yellow color, the proposed models with hyper-parameter 0.1 are labeled by “DNN Cov” and “CNN Cov” with green color.



### 5.3.2 Quantitative Analysis with Fully Connected Neural Networks

We design experiments for comparing our proposed model with the baseline approach DNN model. We randomly select 10 numbers for target data among  $\{0, 1, \dots, 80\}$ . First baseline model is the fully-connected NN. The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.7 and Figure 5.8 (box-plot).

Table 5.7: Comparisons the mean of RMSE between the linear regression, the baseline approach DNN model and the proposed model on the number of 10 randomly selected stock market dataset. The baseline approach DNN model does not contain the Temporal Covariance loss and the proposed model with hyper-parameter 0.1 is denoted as “Proposed with  $\lambda$  0.1”. A **bold** font indicates the best result obtained.

Target number	1	2	21	28	29	33	37	48	60	72
Linear Regression	<b>0.5806</b>	0.4507	0.5447	1.0298	<b>0.3065</b>	0.6369	<b>0.5755</b>	<b>0.5756</b>	<b>0.2560</b>	<b>0.6290</b>
Baseline DNN model	1.0456	0.4022	0.4824	0.5872	0.5518	0.3548	1.3726	1.8978	0.5004	0.7580
Proposed with $\lambda$ 0.1	1.0111	<b>0.3386</b>	<b>0.3758</b>	<b>0.4736</b>	0.4612	<b>0.3534</b>	1.2810	1.4763	0.4367	0.6493

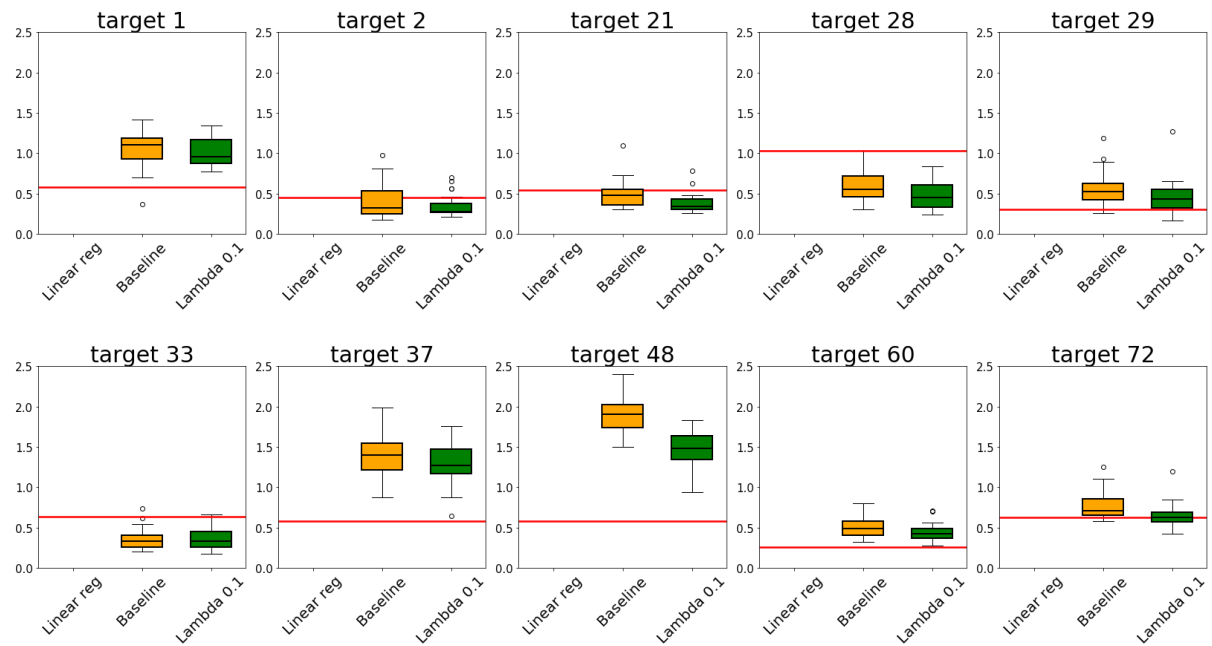


Figure 5.8: Comparisons the RMSE between the linear regression, the baseline approach DNN model and the proposed model on the number of 10 randomly selected stock market dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the horizontal red line, the baseline approach DNN model is denoted “Baseline” and the color is yellow, the proposed model with hyper-parameter 0.1 is labeled by “Lambda 0.1” with green color.

Also, We design experiments for comparing our proposed model with three different hyper-parameters  $\lambda$  as  $\{0.1, 1, 10\}$  for randomly select 10 numbers for target data among  $\{0, 1, \dots, 80\}$ . The experiments repeated 30 times for each of models. The quantitative results shown as Table 5.8 and Figure 5.9 (box-plot).

Table 5.8: Comparisons the mean of RMSE for the proposed DNN model with three different hyper-parameters  $\lambda$  on the number of 10 randomly selected stock market dataset. The proposed DNN model with different hyper-parameters  $\{0.1, 1, 10\}$  is denoted as “Proposed with  $\lambda \{0.1, 1, 10\}$ ”. A **bold font** indicates the best result obtained.

Target number	1	2	21	28	29	33	37	48	60	72
Proposed with $\lambda$ 0.1	<b>1.0111</b>	<b>0.3386</b>	<b>0.3758</b>	0.4736	0.4612	0.3534	1.2810	<b>1.4763</b>	0.4367	0.6493
Proposed with $\lambda$ 1	1.0276	0.4224	0.3840	0.4815	<b>0.3869</b>	<b>0.3183</b>	<b>1.0749</b>	1.5207	0.4962	0.6438
Proposed with $\lambda$ 10	1.1166	0.3961	0.4034	<b>0.4612</b>	0.4785	0.3521	1.1573	1.5685	<b>0.4364</b>	<b>0.6328</b>

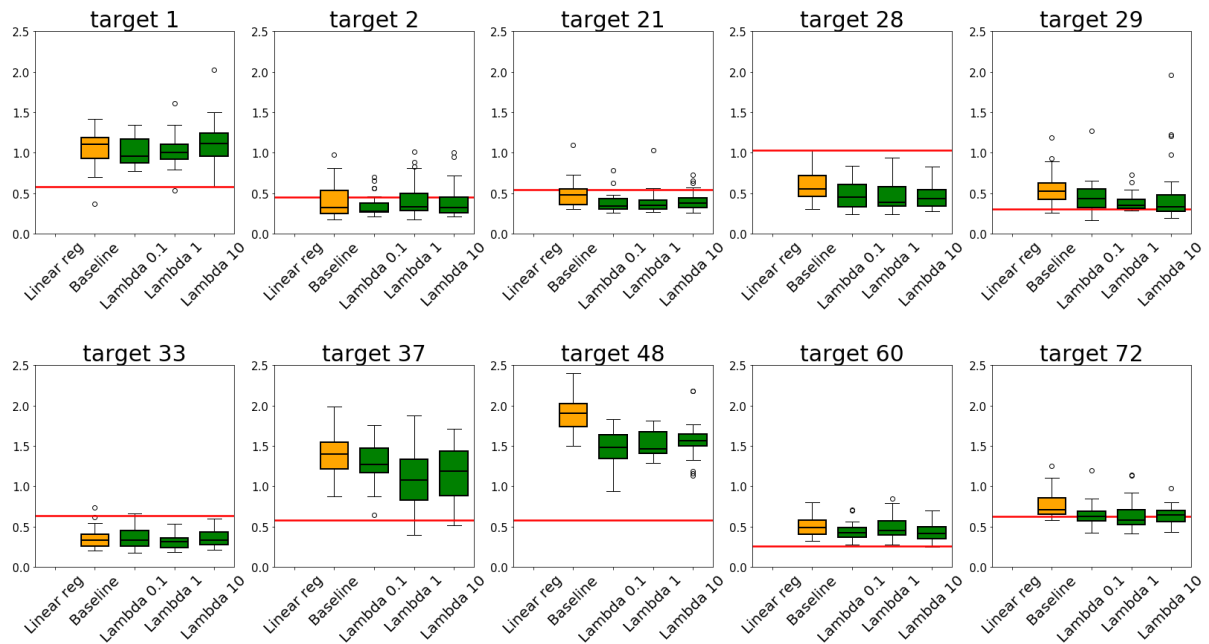


Figure 5.9: Comparisons the RMSE of the proposed DNN model with three different hyper-parameters on the number of 10 randomly selected stock market dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the red horizontal line, the baseline approach model as “Baseline” and the color is yellow, and the proposed model with different hyper-parameters  $\{0.1, 1, 10\}$  as “Lambda  $\{0.1, 1, 10\}$ ” with green color.

### 5.3.3 Quantitative Analysis with Convolutional Neural Networks

We design experiments for comparing our proposed model with the baseline approach CNN model. We randomly select 10 numbers for target data among  $\{0, 1, \dots, 80\}$ . The second baseline model is CNN. The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.9 and Figure 5.10 (box-plot).

Table 5.9: Comparisons the mean of RMSE between the linear regression, the baseline approach CNN model and the proposed model on the number of 10 randomly selected stock market dataset. The baseline approach CNN model does not contain the Temporal Covariance loss and the proposed model with hyper-parameter 0.1 is denoted as “Proposed with  $\lambda$  0.1”. A **bold** font indicates the best result obtained.

Target number	1	2	21	28	29	33	37	48	60	72
Linear Regression	<b>0.5806</b>	<b>0.4507</b>	0.5447	1.0298	<b>0.3065</b>	0.6369	<b>0.5755</b>	<b>0.5756</b>	<b>0.2560</b>	<b>0.6290</b>
Baseline CNN model	1.1184	0.6348	0.8305	0.6379	0.4722	<b>0.3368</b>	1.0778	1.7003	0.4818	0.8638
Proposed with $\lambda$ 0.1	1.0642	0.6121	<b>0.5138</b>	<b>0.5271</b>	0.4768	0.3641	1.1861	1.6525	0.5634	0.6675

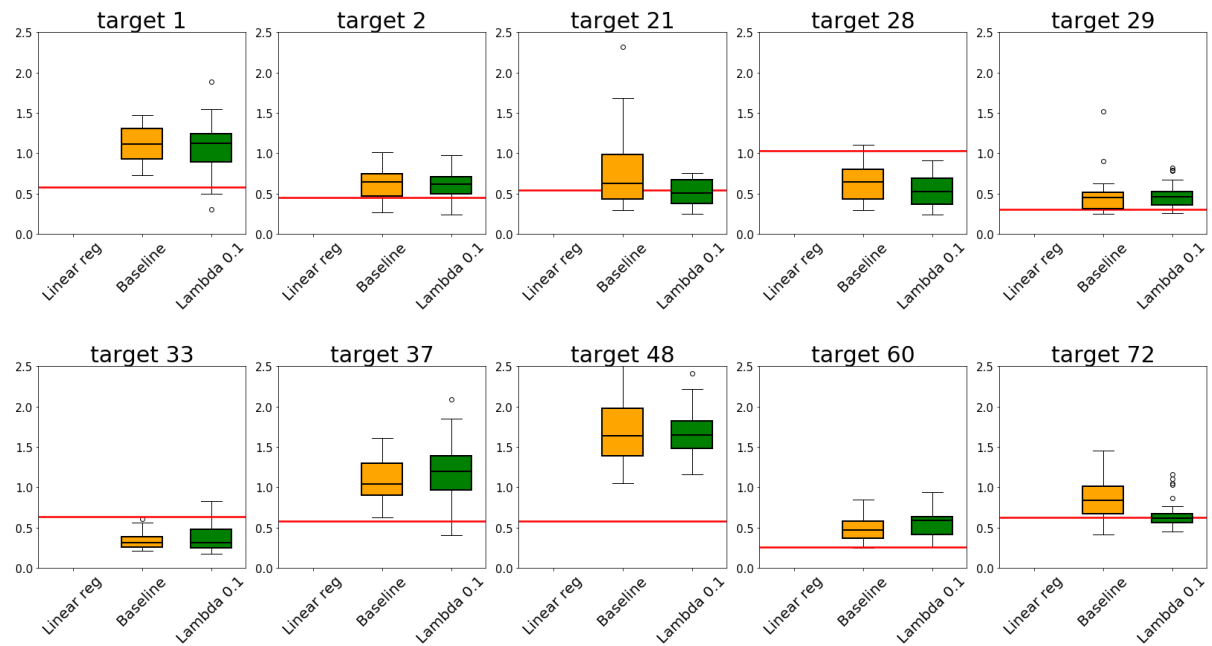


Figure 5.10: Comparisons the RMSE between the linear regression, the baseline approach CNN model and the proposed model on the number of 10 randomly selected stock market dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the horizontal red line, the baseline approach CNN model is denoted “Baseline” and the color is yellow, the proposed model with hyper-parameter 0.1 is labeled by “Lambda 0.1” with green color.

Also, We design experiments for comparing our proposed model with three different hyper-parameters  $\lambda$  as  $\{0.1, 1, 10\}$  for randomly select 10 numbers for target data among  $\{0, 1, \dots, 80\}$ . The experiments repeated 30 times for each of the models. The quantitative results shown as Table 5.10 and Figure 5.11 (box-plot).

Table 5.10: Comparisons the mean of RMSE for the proposed CNN model with three different hyper-parameters  $\lambda$  on the number of 10 randomly selected stock market dataset. The proposed CNN model with different hyper-parameters  $\{0.1, 1, 10\}$  is denoted as “Proposed with  $\lambda \{0.1, 1, 10\}$ ”. A **bold font** indicates the best result obtained.

Target number	1	2	21	28	29	33	37	48	60	72
Proposed with $\lambda$ 0.1	<b>1.0642</b>	0.6121	<b>0.5138</b>	<b>0.5271</b>	0.4768	<b>0.3641</b>	1.1861	1.6525	<b>0.5634</b>	<b>0.6675</b>
Proposed with $\lambda$ 1	1.1047	0.6086	0.5555	0.6631	0.5720	0.3711	1.0234	<b>1.5970</b>	0.5852	0.8881
Proposed with $\lambda$ 10	1.0877	<b>0.5775</b>	0.6107	0.6600	<b>0.4605</b>	0.5214	<b>0.8879</b>	1.9327	0.9202	1.3151

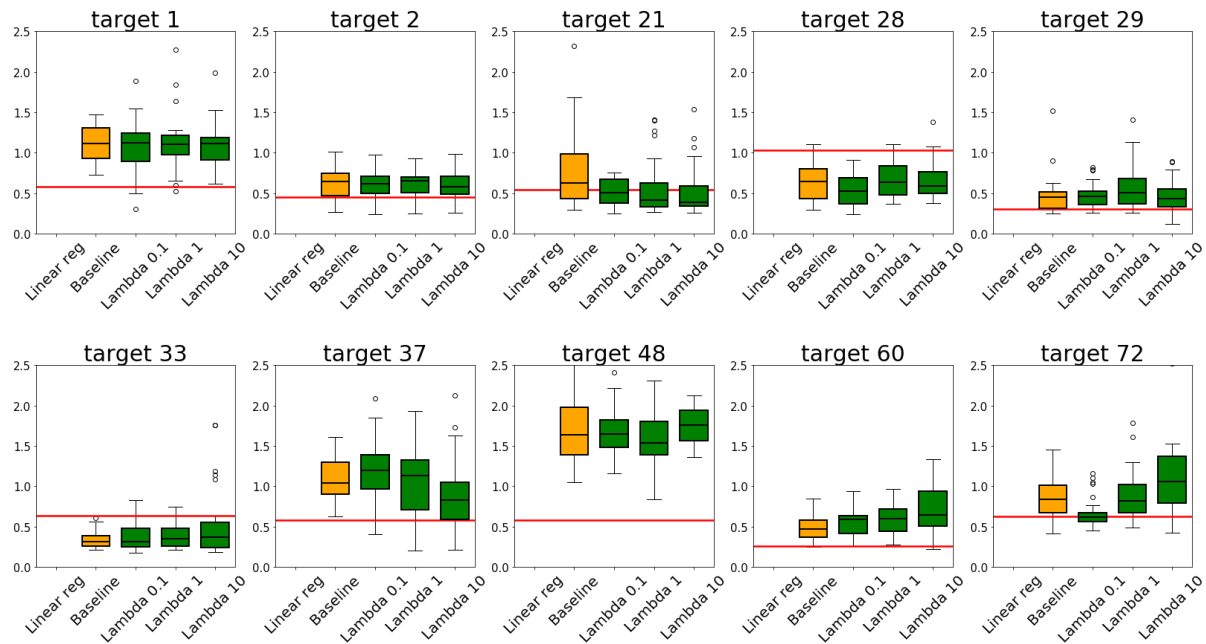


Figure 5.11: Comparisons the RMSE for the proposed CNN model with three different hyper-parameters on the number of 10 randomly selected stock market dataset. The  $y$ -axis is RMSE of each model and the linear regression as “Linear reg” and the red horizontal line, the baseline approach model as “Baseline” and the color is yellow, and the proposed model with different hyper-parameters  $\{0.1, 1, 10\}$  as “Lambda  $\{0.1, 1, 10\}$ ” with green color.

## Chapter 6

# Conclusion

Time series data is one of the most commonly used data that we can obtain from nature and given systems from temperature sensor data to stock market price. The characteristic of time series data can be described by not only sampled values but also covariance of themselves.

Rather than changing the architecture of DNNs or model itself, we proposed the Temporal Covariance loss function to learn the basis functions utilizing Temporal Covariance of target data. Also, the results of experiments have shown that our approach yields more accurate results in terms of prediction for real-world time series dataset. Yet, there are more possibilities to apply our approach to other kinds of DNNs which have the final hidden layer in a vector form. For example, RNNs or LSTM models also can apply our Temporal Covariance loss function.

# Bibliography

- [Adler, 2010] Adler, R. J. (2010). *The geometry of random fields*. SIAM.
- [Aronszajn, 1950] Aronszajn, N. (1950). Theory of reproducing kernels. *Transactions of the American mathematical society*, 68(3):337–404.
- [Bengio et al., 1994] Bengio, Y., Simard, P., Frasconi, P., et al. (1994). Learning long-term dependencies with gradient descent is difficult. *IEEE transactions on neural networks*, 5(2):157–166.
- [Bietti et al., 2019] Bietti, A., Mialon, G., Chen, D., and Mairal, J. (2019). A kernel perspective for regularizing deep neural networks. In *International Conference on Machine Learning*, pages 664–674.
- [Cho and Saul, 2009] Cho, Y. and Saul, L. K. (2009). Kernel methods for deep learning. In *Advances in neural information processing systems*, pages 342–350.
- [Clevert et al., 2015] Clevert, D.-A., Unterthiner, T., and Hochreiter, S. (2015). Fast and accurate deep network learning by exponential linear units (elus). *arXiv preprint arXiv:1511.07289*.
- [Glorot et al., 2011] Glorot, X., Bordes, A., and Bengio, Y. (2011). Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323.
- [Hochreiter and Schmidhuber, 1997] Hochreiter, S. and Schmidhuber, J. (1997). Long short-term memory. *Neural computation*, 9(8):1735–1780.
- [Huang et al., 2015] Huang, W., Zhao, D., Sun, F., Liu, H., and Chang, E. (2015). Scalable gaussian process regression using deep neural networks. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.
- [Kingma and Ba, 2014] Kingma, D. P. and Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.

- [König, 1986] König, H. (1986). Eigenvalue distribution of compact operators.
- [Lee et al., 2017] Lee, J., Bahri, Y., Novak, R., Schoenholz, S. S., Pennington, J., and Sohl-Dickstein, J. (2017). Deep neural networks as gaussian processes. *arXiv preprint arXiv:1711.00165*.
- [Murphy, 2012] Murphy, K. P. (2012). *Machine learning: a probabilistic perspective*. MIT press.
- [Neal, 1995] Neal, R. M. (1995). *Bayesian learning for neural networks*. PhD thesis, Citeseer.
- [Qin et al., 2017] Qin, Y., Song, D., Chen, H., Cheng, W., Jiang, G., and Cottrell, G. (2017). A dual-stage attention-based recurrent neural network for time series prediction. *arXiv preprint arXiv:1704.02971*.
- [Rasmussen, 2003] Rasmussen, C. E. (2003). Gaussian processes in machine learning. In *Summer School on Machine Learning*, pages 63–71. Springer.
- [Schölkopf et al., 2002] Schölkopf, B., Smola, A. J., Bach, F., et al. (2002). *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press.
- [Srivastava et al., 2014] Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., and Salakhutdinov, R. (2014). Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958.
- [Wegman, 2014] Wegman, E. J. (2014). Reproducing kernel hilbert spaces. *Wiley StatsRef: Statistics Reference Online*.
- [Williams and Rasmussen, 2006] Williams, C. K. and Rasmussen, C. E. (2006). *Gaussian processes for machine learning*, volume 2. MIT Press Cambridge, MA.
- [Yi et al., 2017] Yi, S., Ju, J., Yoon, M.-K., and Choi, J. (2017). Grouped convolutional neural networks for multivariate time series. *arXiv preprint arXiv:1703.09938*.

## Acknowledgements

My deepest debts of gratitude to my advisor, Jaesik Choi. He has been guiding me for more than two years and showing me the path of a good researcher by being one. I'm sure his advice and what I've learned have influenced my life a lot.

I am also grateful to everyone of Statistical Artificial Intelligence Lab for their unconditional support. They are helpful and encouraging. It was luck to have them as academic colleagues for the past few years of my studying and research.

I owe thanks most of all to my family and my wife for helping me focus on research in every way for a not short period of time. There were many setbacks and difficulties that I would have not been able to work through without their support and patience.

Ulsan National Institute of Science and Technology gave me lots of opportunities and advantages. I wish my friends and researchers in UNIST success in all their future endeavors.



