이학박사 학위논문

Time Series Data Analysis using Deep Learning in Industry

(산업에서 딥러닝을 이용한 시계열 데이터 분석)

2019년 2월

서울대학교 대학원

협동과정 계산과학전공

김상연

Time Series Data Analysis using Deep Learning in Industry

(산업에서 딥러닝을 이용한 시계열 데이터 분석)

지도교수 강 명 주

이 논문을 이학박사 학위논문으로 제출함

2018년 10월

서울대학교 대학원

협동과정 계산과학전공

김상연

김상연의 이학박사 학위논문을 인준함

2018년 12월

위 원 장	____________________	(인)
부 위 원 장	____________________	(인)
위 원	____________________	(인)
위 원	____________________	(인)
위 원	____________________	(인)

Time Series Data Analysis using Deep Learning in Industry

A dissertation
submitted in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
to the faculty of the Graduate School of
Seoul National University

by

Sangyeon Kim

Dissertation Director : Professor Myungjoo Kang

Department of Computational Science & Technology
Seoul National University

Feb 2019

Abstract

Deep learning, also called as artificial neural networks, is one of the most important and powerful subjects in industrial in recent years. Deep learning starts to show a great performance from image classification and in these days it have been applied to fields including computer vision, natural language process, speech recognition and etc. The performance is better than not only previous machine learning techniques, but also human experts in some cases. For an area with time series data, recurrent neural networks is widely used algorithm of deep learning. The aim of this theseis is to apply deep learning, especially with recurrent neural networks, for an industrial such as anomaly detection and trend prediction in financial market, with time series data . Its main contributions are (1) a new model for anomaly detection in time series data even for various length inputs, (2) various neural architectures for prediction in finance, and (3) attention networks and model analysis with attention vectors. Each experimental results of applications show better performances than previous machine learning techniques.

Key words: Deep Learning, LSTM, Attention, Time series, Anomaly detection, Trend Prediction
Student Number: 2015-30123

Contents

Chapter 1

Introduction

From introducing an appearance of concept about deep learning in 1950s, through the ice age in 1970s, in recent years it becomes one of the most important and powerful subjects in industrial. Deep learning could have developed with various neural network models, faster hardware computing process and a big data. Primarily, deep learning, also called artificial neural networks, includes in machine learning and deal with algorithms inspired by the architecture and flow of the brain. One of important differences between machine learning and deep learning model is on the feature extraction area since machine learning done the process by human whereas deep learning model figure out by itself with a big data. Therefore, deep learning can select various features of data that human cannot define with an algorithm. Mathematically, non-linearity of neural networks with multiple layers and various non-linear functions inside the deep learning model is the reason of better performance than other machine learning techniques. From beginning of simple neural networks, in these days we have various kinds of neural networks for various input data such as an image, voice, sentence and etc. It starts with simple neural networks with a few layers which is called as Multi Layer Perceptron. For image area, convolutional neural networks improve the accuracy for various tasks and this makes deep learning as the important subjects for industrial research. For sequential data, recurrent neural networks which is modelling Markov model, are able to learn and it developed to Long Short Term Memory networks and attention networks for analyzing longer sequences. In this thesis, we will introduce neural networks and varioius models for sequential data. Then we will apply deep learning models in several industrial areas. For anomaly detection as unsupervised learning approach, we suggest a new ensemble model, Deep Correlation Mapping, which is combined LSTM and t-SNE. For predicting trend in financial market, we compare various deep learning models and we got the best performance with weighted attention model. The thesis is organized as follows: In Chapter 2, basic backgrounds for neural networks and some of techniques for stablizing training are introduced. In Chapter **??**, we introduce various deep learning models for time series data analysis. We will explain each models with simple models of each networks. First application in industrial is given in Chapter 4 with anomaly detection in unsupervised learning approach. Second application for financial market is in Chapter 5 with predicting trend of KOSPI 200. Finally, Chapter 6 contains conclusion and discussion for the future works.

Chapter 2

Deep Learning Background

2.1 Neural Networks

In this section, we will explain a basic introduction of neural networks with simple figures. Fig 2.1 shows a single neuron with inputs and the output. Denote the input with n-dimensional vector as $x \in R^n$ and activation function f to generate output z. Basically, inputs pass through a simple layer which consists with bunch of neurons and activation functions to output for next layer. The output is computed by the following function :

$$z = f(W^T x + b)$$

Weight matrix W and the bias vector b are trainable parameters and that implies these parameters moves towards to minimizing errors between model outputs and target outputs.

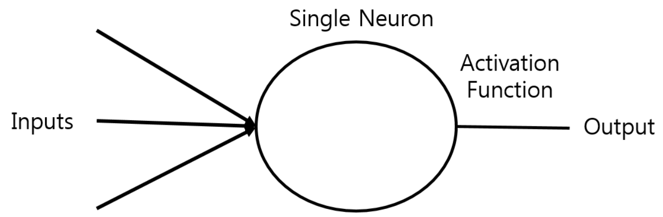


Figure 2.1: A single neuron with inputs, activation function and outputs.

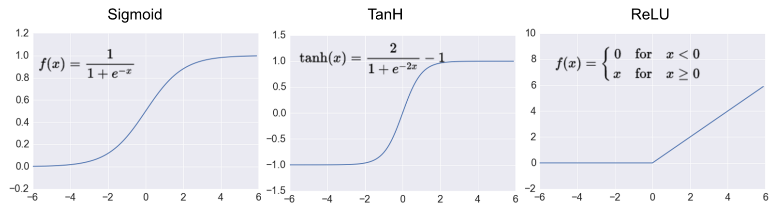


Figure 2.2: Common activation functions.

2.2 Various Activation Functions

Activation functions can be a linear function but normally we use non-linear function for being able to find out complex non-linear features in data. The activation function plays a major role in the success of training deep neural networks and we will show some of major non-linear functions and their properties.
Those activation functions are plotted in Fig 2.2.

1) Sigmoid function Sigmoid function looks smooth version of step function and its expression is following :

$$sigmoid(x) = \frac{1}{1 + e^{-x}}$$

It has non-linearity and differentiable function with bounded in range (0, 1).
2) Tanh Function Tanh function is similar with sigmoid function since it composed with exponential functions as following :

$$tanh(x) = \frac{1 - e^{-2x}}{1 + e^{-2x}}$$

It has close relation with sigmoid function and actually it can be expressed with each other as $tanh(x) = 2sigmoid(2x) - 1$. Difference with sigmoid function is tanh function is bounded to range (-1, 1).
3) Relu Rectified Linear Unit (Relu) is one of the most used activation functions.

$$relu(x) = max(0, x)$$

It has range (0, ∞) and espeically ourput zero with negative input implies this part is nothing changed when update weights and it can make the network lighter and faster optimization to target output. Also, for large

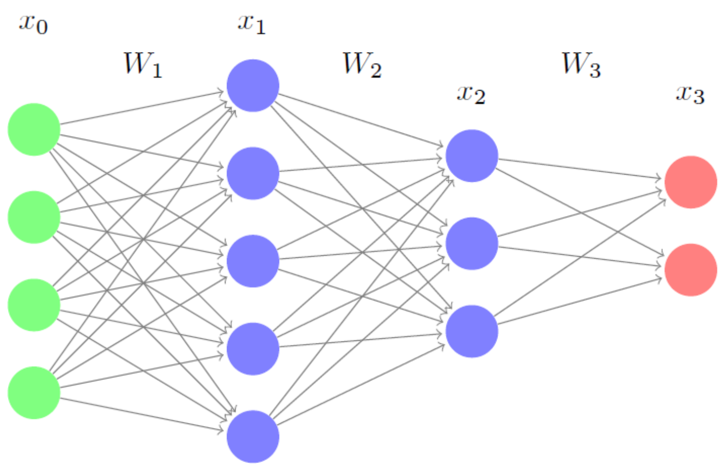


Figure 2.3: Example of backpropagation.

input, relu conserve its amount to update weights that helps us for training faster and avoid vanishing gradients. However, sometimes 0 with negative input can delete important features with negative value, so variations in Relu like LeakyRelu [19], pRelu [10] are suggested.

2.3 Error Backpropagation

The learning process of neural networks takes the inputs and the target outputs and updates trainable parameters inside the model towards to making model output get as close as possible from the target output. The backpropagation algorithm is the inverse direction of the feedforward process in order to compute the gradients of whole network parameters. First, we compute the error with target output y and model output $haty$. For a simple method with sum of squared error, we can calculate error as

$$E[w] = \frac{1}{2}(\hat{y} - y)^2$$

To reduce the value of the error function, we have to change these weights in the negative direction of the gradient of the loss function with respect to these weights. Let's look at the simple example in Figure 2.3.

We denote input is x_0 then forward propagation equations are as follows:

$$\begin{aligned} \text{Input} &= x_0 \\ \text{Hidden Layer1 output} &= x_1 = f_1(W_1 x_0) \\ \text{Hidden Layer2 output} &= x_2 = f_2(W_2 x_1) \\ \text{Output} &= \hat{y} = f_3(W_3 x_2) \end{aligned}$$

Backpropagation equations can be derived by repeatedly applying the chain rule. First we start with the derivatives of final output error with respect to W_3.

$$\begin{aligned} \frac{\partial E}{\partial W_3} &= (\hat{y} - y)\frac{\partial \hat{y}}{\partial W_3} \\ &= [(\hat{y} - y) \circ f_3'(W_3 x_2)]\frac{\partial W_3 x_2}{\partial W_3} \\ &= [(\hat{y} - y) \circ f_3'(W_3 x_2)]x_2^T \\ \text{Let } \delta_3 &= (\hat{y} - y) \circ f_3'(W_3 x_2) \\ \frac{\partial E}{\partial W_3} &= \delta_3 x_2^T \end{aligned}$$

Here $\circ$ is the Hadamard product. We use chain rule for calculating each partial derivatives effectively. Let us check the dimension of each matrixes. A dimension of W_3 is 2×3 and $\frac{\partial E}{\partial W_3}$ must be same. Dimensions of $(\hat{y} - y)$ and $f_3'(W_3 x_2)$ are 2×1, so δ_3 is also 2×1, so dimensions of $\delta_3 x_2^T$ is 2×3 which is same with W_3. For the weights in W_2, we do same calculations:

$$\begin{aligned} \frac{\partial E}{\partial W_2} &= (\hat{y} - y)\frac{\partial x_3}{\partial W_2} \\ &= [(\hat{y} - y) \circ f_3'(W_3 x_2)]\frac{\partial (W_3 x_2)}{\partial W_2} \\ &= \delta_3 \frac{\partial (W_3 x_2)}{\partial W_2} \\ &= W_3^T \delta_3 \frac{\partial x_2}{\partial W_2} \\ &= [W_3^T \delta_3 \circ f_2'(W_2 x_1)]\frac{\partial W_2 x_1}{\partial W_2} \\ &= \delta_2 x_1^T \end{aligned}$$

And for W_1 which is a weight matrix of the first layer:

$$\begin{aligned}\frac{\partial E}{\partial W_1} &= [W_2^T \delta_2 \circ f_1'(W_1 x_0)] x_0^T \\ &= \delta_1 x_0^T\end{aligned}$$

We do this process recursively and with L layers with weight matrices $W_1, W_2, .., W_L$ and activation functions $f_1, f_2, .., f_L$ respectively.
Forward Pass:

$$\begin{aligned}x_i &= f_i(W_i x_{i-1}) \\ E &= \frac{1}{2}\|\hat{y} - y\|_2^2\end{aligned}$$

Backward Pass:

$$\begin{aligned}\delta_L &= (\hat{y} - y) \circ f_L'(W_L x_{L-1}) \\ \delta_i &= W_{i+1}^T \delta_{i+1} \circ f_i'(W_i x_{i-1})\end{aligned}$$

Finally, we update each weights with gradient descent rules:

$$\begin{aligned}\frac{\partial E}{\partial W_i} &= \delta_i x_{i-1}^T \\ W_i &= W_i - \eta \circ \frac{\partial E}{\partial W_i}\end{aligned}$$

2.4 Regularization

2.4.1 Dropout

As deep learning models are going deeper, there are enormous trainabel parameters to be updated and as epoch increases, the model overfits to train dataset and can apart from test dataset. Dropout [29] helps these problems by ignoring neurons during the training stage randomly. As in Figure 2.4 (b) "Ignoring" implies neurons that choiced are not considered during a forward or backward pass during a epoch which is difference with standard neural networks in (a). Therefore we do not train whole neurons in layers where we apply dropout and that helps prevent overfitting by reducing correlation between neurons. Also, we can save memories while we training with less neurons for an epoch. However, we need more numbers of epochs to converge

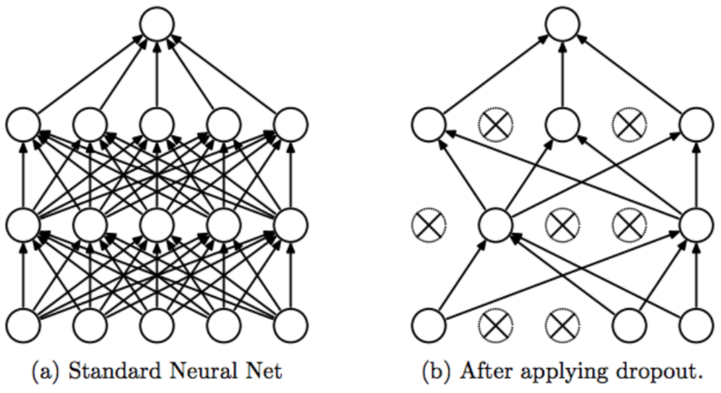
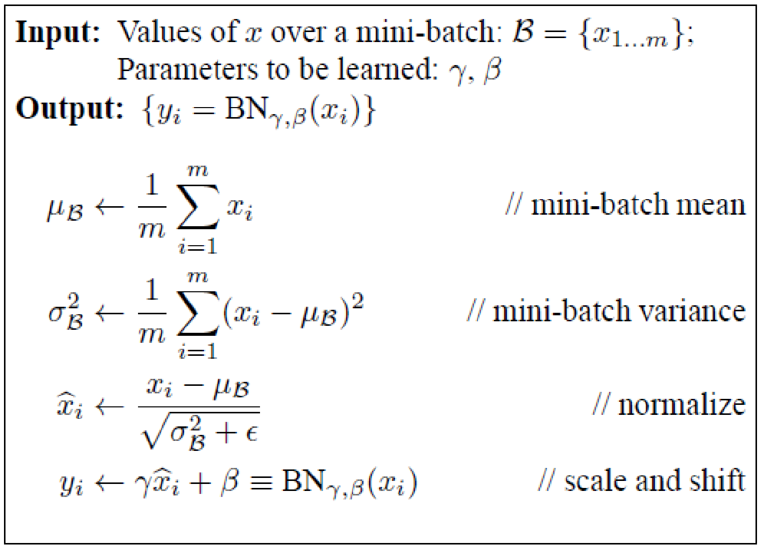


Figure 2.4: Dropout in Deep Learning.

because of randomness of dropout.

2.4.2 Batch Normalization

While dropout is a simple method of regularization and it is a view point from computing algorithms, batch normalization [13] is a regularization method with adjusting mean and variance for preventing covariance shift. To make training process being stabilized, we need stable range of inputs of each layer without losing features among them. As algorithm in Figure 2.5, batch normalization normalizes the output of a previous activation layer by subtracting the batch mean and dividing by the batch standard deviation. Therefore the process just change scale and move data to inside a unit circle. However, this fixed range of output(also, input of next layer) cannot be enough with scale or variance aspects for next layer. Therefore they multiply a parameter for a standard deviation and add a parameter for a mean and both parameters are trainable parameters. Batch normalization reduces the amount of differences of same classes inputs if the model is trained well for extracting features and it implies not only for stablizing training process, but also batch normalization makes faster convergence.



Input: Values of x over a mini-batch: $\mathcal{B} = \{x_{1\ldots m}\}$;
Parameters to be learned: γ, β

Output: $\{y_i = \mathrm{BN}_{\gamma,\beta}(x_i)\}$

$$\mu_{\mathcal{B}} \leftarrow \frac{1}{m}\sum_{i=1}^{m} x_i \qquad \text{// mini-batch mean}$$

$$\sigma_{\mathcal{B}}^2 \leftarrow \frac{1}{m}\sum_{i=1}^{m} (x_i - \mu_{\mathcal{B}})^2 \qquad \text{// mini-batch variance}$$

$$\widehat{x}_i \leftarrow \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}} \qquad \text{// normalize}$$

$$y_i \leftarrow \gamma\widehat{x}_i + \beta \equiv \mathrm{BN}_{\gamma,\beta}(x_i) \qquad \text{// scale and shift}$$

Figure 2.5: Batch Normalization Algorithm.

Chapter 3

Deep Learning Models

In this chapter 3, we present various deep leraning models that we use for analyzing time series data in industrial. Each sections will introduce simple forms of each models and we explain their features about time series data.

3.1 Multi Layer Perceptron

Multi Layer Perceptron(MLP) is a simple deep learning model which consists of input layer, hidden layers, and output layer. (Figure 3.1) It contiains one or more hidden layers and that enables model can learn non-linear functions that a single layer perceptron cannot. We also called this network as fully-connected layer since each neurons in a layer is connected to all other neurons in the next layer. The connection(edge) has a weight and the network is feed-forward neural networks. An MLP can be considered as a function that maps from input to output vectors. Since the behaviour of the function is parameterised by the connection weights, a single MLP can be express

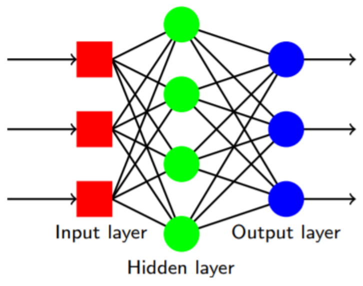


Figure 3.1: Example of Multi Layer Perceptron.

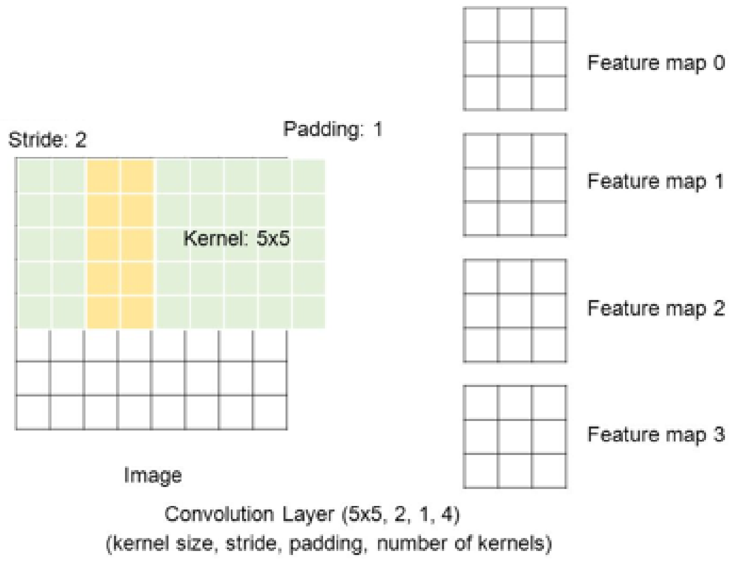


Figure 3.2: Example of Convolutional Neural Networks.

various kinds of different functions. However the output of an MLP depends only on the current input, and independent with any past or future inputs, MLP is not a useful model for time series data. We will test MLP with other models in our second application for financial market which is in Chapter 5.

3.2 Convolutional Neural Networks

Although MLP can implements various non-linear functions, we need more complex and reasonable networks for solving problems. Especially, if positions are the important factors in features of data, MLP does not work well because of the reason we mentioned at previoius section. Convolutional Neural Networks(CNNs), which use convolution operator with receptive field in input domain, can overcome limitations of MLP and it becomes one of the main reasons why deep learning is so popular today. Let us denote a convolution layer which accpets an input with volume of size $W_i \times H_i \times D_i$. Convolution layer need four hypermarameters such as number of filters K, their spatial extent or size F, the stride S and padding type P. First, filter size F contains a dimension for width, hegith, and depth and filter operates convolutions through input as sliding window for all regions. Number of filters decide how many features we extract through this layer by convolution operation. The operation overlays as the filter size to the input and perform element wise multiplication and add the result. Stride S decideds a length that we move the filter in the input to calculate the next region of input. We have two options of padding type P, "same" and "valid". Same padding increases the size of input with horizontally and vertically to maintain the size after convolution. CNNs have great advantages of parameter sharing and sparsity. Parameter sharing in CNNs controls the number of parameters and that makes we need less memory for computing compared to MLP. Sparsity of connections means unlike MLP, CNNs only need a weight matrix for a receptive field with a size F not whole connections between input neurons and output neurons. We provide simple example figures of CNNs in Figure 3.2. CNN is a very effective class of neural networks that is highly effective at classifying structured data where the order of arrangement matters like images, audio and video. Many tasks with image data give the best results with CNNs and even its performance on image recognition tasks has surpassed human performance on standard datasets recently. We use time series data as input so convolutional dimension depth will be 1. We call it as 1D CNNs and this is very effective when we derive features from fixed length segments of the overall data set and where the location of the feature within the segment is not of high relevance. Therefore, it applies well with time sequences such as sensor signal (if signals are fixed length), NLP and audio data. We will build 1D CNNs model for our second application, trend prediction in financial market, which is in Chapter 5.

3.3 Recurrent Neural Networks

Time series data have a correlation itself with past or future inputs and it deals with markov chain models. Especially, Hidden Markov Model(HMM) was widely used as machine learning method for time series prediction. However, HMM considers relations only for previous step of input and do linear operation to get output. Therefore, we need developed algorithm for complex features in time series data. Recurrent Neural Networks (RNNs) are popular models that have shown great promise in many sequential data tasks like machine translation, speech recognition, sentence classification and etc. While traditional neural network assume that all inputs (and outputs) are independent of each other but RNNs perform the same task for every element of a sequence, with the output being depended on the previous computations. This is why we called it "Recurrent" and that helps us to predict what comes next by extracting features of input time series data. A simple RNN looks

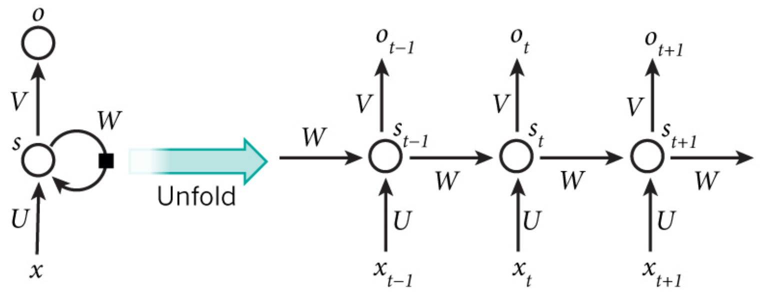


Figure 3.3: Example of Simple Recurrent Neural Networks.

like in Figure 3.3 with a folding and unfolding figure. x_t is the input at time step t, and s_t is the hidden state at time step t. s_t is calculated based on the input at time t and previous hidden state s_{t-1} :

$$s_t = f(Ux_t + Ws_{t-1})$$

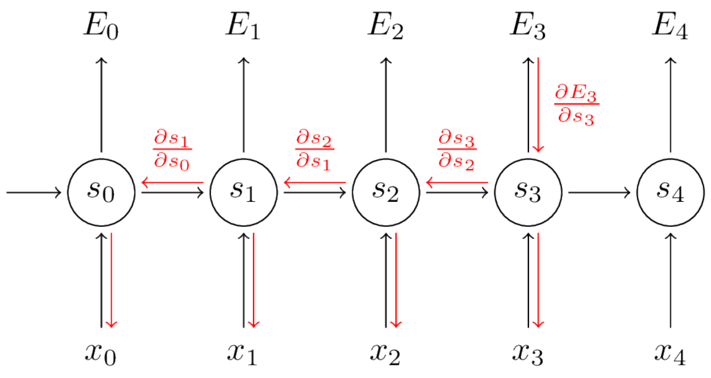
The function f is an activation function which is usually a nonlinearity function such as tanh or Relu. o_t is the output at step t. If we want to get probability distributions with final outputs, $o_t = softmax(Vs_t)$. For update weights in RNN, we use backpropation through time (BPTT) that implies the total error is just the sum of the errors at each time step.

$$\frac{\partial E}{\partial W} = \sum_t \frac{\partial E_t}{\partial W}$$

For an instance, take a look at Figure 3.4. To update weights through backpropagation, we need to calculate the gradients of the error respect to trainable weights U, V and W. Take a look at the gradient of E_3 respect to W :

$$\begin{aligned}\frac{\partial E_3}{\partial W} &= \sum_{k=0}^{3} \frac{\partial \hat{y}_3}{\partial E_3} \frac{\partial \hat{y}_3}{\partial s_3} \frac{\partial s_3}{\partial s_k} \frac{\partial s_k}{\partial W} \\ &= \sum_{k=0}^{3} \frac{\partial \hat{y}_3}{\partial E_3} \frac{\partial \hat{y}_3}{\partial s_3} (\prod_{k=0}^{3} \frac{\partial s_j}{\partial s_{j-1}}) \frac{\partial s_k}{\partial W}\end{aligned}$$

Since derivative of tanh is bounded by 1, $\|\frac{\partial s_j}{\partial s_{j-1}}\| \leqslant 1$ and this makes the gradient $\frac{\partial E_3}{\partial W}$ can be small value. Especially, as sequence get longer, higher



E_0
E_1
E_2
E_3
E_4
$\frac{\partial E_3}{\partial s_3}$
$\frac{\partial s_1}{\partial s_0}$
$\frac{\partial s_2}{\partial s_1}$
$\frac{\partial s_3}{\partial s_2}$
s_0
s_1
s_2
s_3
s_4
x_0
x_1
x_2
x_3
x_4

Figure 3.4: Example of Vanishing Gradient Problem.

chance to derivative goes to zero and that occurs weight saturating even they have high error. (If you want more details, check (On the difficulty of training recurrent neural networks).) This problem is called "Vanishing Gradient Problem". Therefore, we need more sophiscated networks for keeping important features at the head of inputs until our model runs to the tail.

3.4 Long Short Term Memory

The Long Short Term Memory(LSTM) network is a recurrent neural network is made for overcoming the vanishing gradient problem. LSTM is an extension for recurrent neural networks, which basically extends their memory and that makes this network well suited to learn from important experiences that have even very long time steps in between.

As in Figure 3.5, LSTM has three gates: forget, input and output gate. Forget gate decides what information to discard from the unit. Input gate decides which values from the input to update the momory state and output gate decides what to output based on input and the memory of the unit. Same with simple RNNs, each gates are decided by previous hidden state and current input but each gate have trainable weights itself and they decide the proportion how much they forget or add or pass for the next step. By using those gates, LSTM then combine the previous state, the current memory, and the input. It turns out that LSTMs are very efficient at capturing long-term dependencies and this is why LSTM networks are widely used for sequential data including time series data.

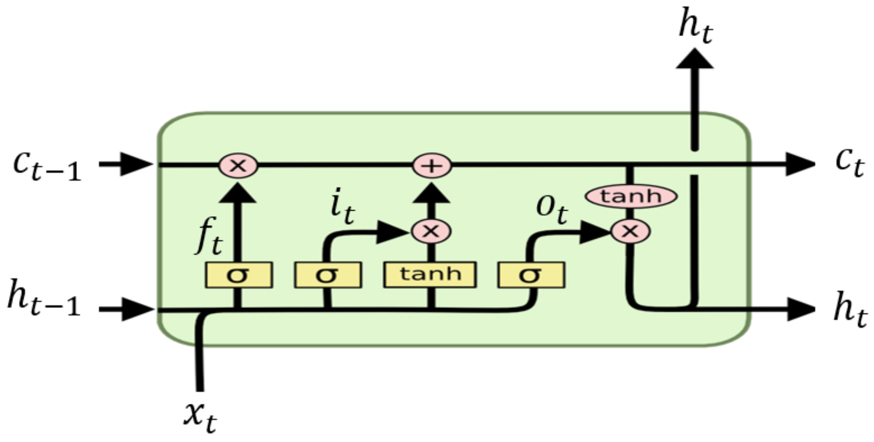


Figure 3.5: A diagram of Long Short Term Memory.

3.5 Attention Networks

The attention mechanism was made for helping memorize long sequences with focusing important features of inputs. The main difference between LSTM networks and attention networks is that "Attention" pays attention to particular parts rather than treating the whole inputs with same weights. For example, when we want to predict a financial market movement with previous time series data, we want to pay attention to important days than normal days that affects insignificantly to our target output. Then we need to give higher weights for the important days than normal days and this can be done by trained model which understand relations between inputs and outputs correctly. To compute the weighted features for the attention networks, we put an attention layer as in Figure 3.6. Let us denote inputs as $\{x_1, x_2, \cdots x_T\}$ and previous hidden state as h_{t-1} then we compute a score s_i to measure how much attention for x_i:

$$s_i = \tanh(W_h h_{t-1} + W_x X_i)$$

For making scores to weights α_i with summation as 1, we normalize s_i:

$$\alpha_i = softmax(s_1, s_2, \cdots, s_T)$$

Finally, we use Z to a new input to next layer:

$$Z = \sum_i \alpha_i x_i$$

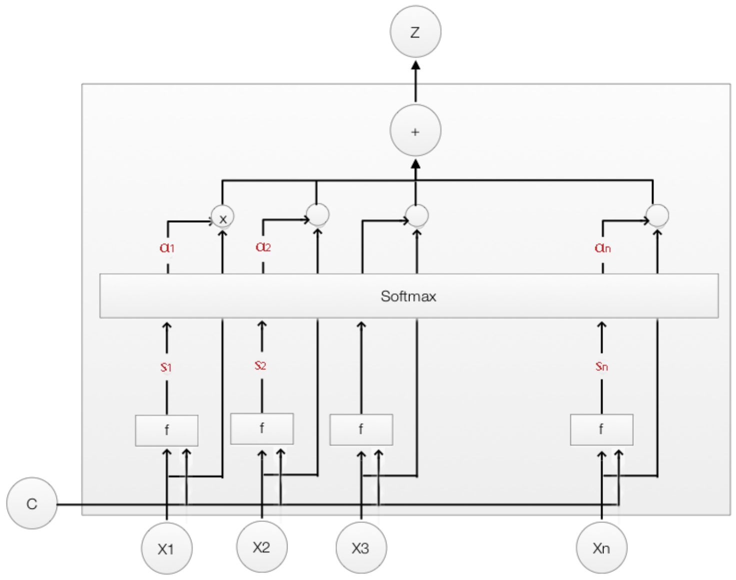


Figure 3.6: A diagram of attention networks.

In this section, we mentioned about attention networks for LSTM networks and we also use simple attention networks for a simple input without hidden state in Chapter 5.

Chapter 4

Anomaly Detection

In industrial, as operation systems get more complex and developed, we need investigation systems faster and more efficient than before. The most basic knowledge for investigating system, we need to be able to handle properties of sensors based on recorded time series data. Sometimes, defective products come out from some processes and we look into recorded sensors, time series data, to find out root causes. We compare the data between normal products and defective products and odd sensors are suspected as causes of defects. Anomaly detection is the problem of finding rare items, events or observations, which are differing significantly from the major patterns of the data. Figure 4.1 explains anomalies in a simple two-dimensional data set. We can regard the data has two groups, G_1 and G_2, since almost data contains in these two regions. Points that are sufficiently far away from these regions, for example, points p_1 and p_2 can be regarded as anomalies.

However, time series data of each sensors have various shapes or lengths and it makes us hard to analyze correlation between them even they correlate each other as knowledge. For analyzing this problem, various methods for calculating similarity or correlation coefficient between time series data proposed. There are some papers based on deep learning to do anomaly detection or unsupervised learning for time series data. [20] used mixture model of LSTM and encoder-decoder that learns to reconstruct normal time series behavior for anomaly detection. For anomaly detection in noisy highly periodic data, [28] tested various models including DNNs, RNNs, and LSTMs to perform regression and predict the expected value then compare the actual values in the time series. Sensor data have complex features like different lengths and consist with time steps, clean or noisy, various scale, time delaying and so on even on the same manufacture process. Therefore, the model

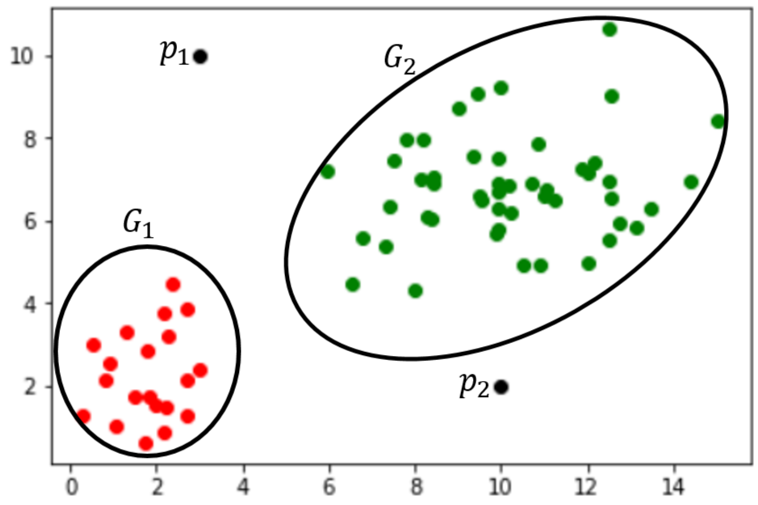


Figure 4.1: An example of anomalies in a two-dimensional data set.

to analyze sensor data need to consider those features and that can lead us to detect abnormal data that can cause serious problems like making bad products. In this chapter, we build a new deep learning model for extracting features of complex signal data that is needed for reasonable embedding result and define the metric to distinguish an abnormal data between them. One of the most popular networks for processing sequential data is recurrent neural networks and Long-Short Term Memory (LSTM) network [12] is especially widely used among them. The networks are used for various aspects using sequential data in deep learning like speech analysis, sentimental analysis, voice recognition and financial analysis because of its characteristics that can prevent forward features by long term memory and keep short term memory well as simple recurrent neural networks. We used LSTM to deal with our complex time series signal data with respect to each steps. After we obtained features through the deep learning model, we used them to optimize KL (Kullback-Leibler) divergence between original distribution where computed by correlation between them and feature distribution where computed by distance between feature vectors. We take inspiration from parametric t-SNE [30], t-Stochastic Neighbor Embedding, which uses deep neural network to parametrize the embedding. T-SNE minimizes the KL divergence between an original data distribution and an embedded distribution for optimizing the clustering process with features obtained by deep neural networks. The model towards learning for extracting features, which can optimize KL divergence and its result, show us the simple map for clustering complex original data. This map shows us which data can be regarded as similar movement with steps and which data can be regarded as abnormal data. We define a new metric to classify normal clusters and find out abnormal data as results. In this chapter 4, we propose the ensemble model of LSTM and t-Stochastic Neighbor Embedding method to obtain correlation between sensors. Deep learning model will help us to extract features of each steps and then we used them for representing features of sensor data even they have complex properties. After feature extraction, t-SNE will do reduction of dimensionality from high dimensional feature vector space to low dimensional projected vector space as minimizing KL divergence. With trained DeepCorr model, we get results with low dimensional space that presents how much each sensors are close or relate. We calculate correlations respect to a new function which we defined and then use them to do anomaly detection.

4.1 Related Works of Anomaly Detection

4.1.1 Anomaly detection

While supervised anomaly detection has relation with classification techniques, unsupervised anomaly detection is a quite difficult problem since it has no target label in the data. To do unsupervised anomaly detection many approaches with clustering and then find out data which is out of cluster. For clustering complex data, many machine learning techniques like PCA, linear discriminant analysis, t-SNE are widely used to mapping lower dimensional space where we do clustering. [20] used mixture model of LSTM and encoder-decoder that learns to reconstruct normal time series behavior for anomaly detection. The outputs are from reconstruction by LSTM Encoder-Decoder model and difference with original data implements anomaly scores of the sensor. The data they used were periodic time series is difference with our non-periodic data and noisy sensors will have high anomaly scores even they are normal sensors with the model. For anomaly detection in noisy highly periodic data, [28] tested various models including DNNs, RNNs, and LSTMs to perform regression and predict the expected value then compare the actual values in the time series. Moreover, they ensemble different models that one with statistical backing and another that can be easily modified by the user and get robust method to detect anomalies. However, their approaches are based on supervised learning and not end-to-end model makes high computational cost which is not proper for faster industrial business.

4.1.2 t-SNE

t-SNE is the popular method for dimension reduction and visualization for big data by embedding data in high dimensional space to low dimesional space. It develops from SNE(Stochastic Neighbor Embedding) which makes low dimension probablity space similar to high dimension probability space stochastically using loss function as Kullback-Leiber divergence. However, SNE assumes both distribution as Gaussian distribution and it implies j^th element which is nearby i^th element in original high dimension space and k^th element which is nearby i^th element in original high dimension space are not quite different from the probability view point as chosen neighborhood of i^th element. Therefore t-SNE [30] suggests t-distribution instead of Gaussian distribution with low dimensional distribution and remains high dimensional distribution as Gaussian distribution. This method is the state-of-the-art in data visualization and dimensionality reduction for non-formulaic data. However, datasets which they used were focused on image datasets or only simple text data which is quite simple than sensor data.

4.1.3 Clustering

For unsupervised learning for anomaly detection, autoencoder method is widely used because it has a strong point to capture proper features of input to decode itself or similar domain. In [5], they used Stacked Denoising Autoencoders to perform anomaly detection traking from low level features. However, there are not abundant researches using deep learning techniques to Anomaly detection.[7] [33] gave us a key idea to how to use deep learning techniques to anomaly detection. They used KL divergence for loss function to optimize parameters and they clusterized with k-means clustering features from deep neural networks. However, they still used autoencoder model for parameter initialization that cannot be hard to adjust to our data because of a variable length property.

4.2 Deep Correlation Mapping

Consider we have signal datasets $X = \{X_1, X_2, \cdots, X_k\}$ and each signal data can be expressed as $X_i = \{x^i_{1,0}, \cdots, x^i_{1,t_1}, x^i_{2,0}, \cdots, x^i_{2,t_2}, \cdots, x^i_{n,0}, \cdots, x^i_{n,t_n}\}$ which have n steps with time length $\{t_1, t_2, \cdots, t_n\}$. Instead of calculating correlations directly in the data X, we transform the data with nonlinear mapping $f_\theta : X \rightarrow Z$ where θ are learnable space and Z is the latent space. Our model consider the following problems to find out proper f_θ First, we need to consider how to extract features that consist latent space with reflectling properties of our original data which in high dimensional space. We need to use time analysis model with variable length inputs because some of them look similar but different length for each steps. Also, the target function to train the model within obtatained latent space is important for anomaly detection. Our model should solve not only complex problems, but also preserve the simple problems that highly correlated data even with a simple method.

4.2.1 LSTM

Denote time series of one data sample as $\{x_0^k, \cdots, x_t^k, x_{2,0}^k, \cdots, x_{t_k}^k\}$ where k is the time step number of the data. The simple forms of the equations for the foward of an LSTM unit with a forget gate as follows:

$$f_t^k = \sigma_g(W_f^k x_t^k + U_f^k h_{t-1}^k + b_f^k) \tag{4.2.1}$$
$$i_t^k = \sigma_g(W_i^k x_t^k + U_i^k h_{t-1}^k + b_i^k) \tag{4.2.2}$$
$$o_t^k = \sigma_g(W_o^k x_t^k + U_o^k h_{t-1}^k + b_o^k) \tag{4.2.3}$$
$$c_t^k = f_t^k \odot c_{t-1}^k + i_t^k \odot \sigma_c(W_c^k x_t^k + U_c^k h_{t-1}^k + b_c^k) \tag{4.2.4}$$
$$h_t^k = o_t^k \odot \sigma_h(c_t^k) \tag{4.2.5}$$

where f_t^k is a forget gate, i_t^k is an input gate, o_t^k is an output gate, c_t is a cell state, h_t^k is a hidden state and the operator $\odot$ denotes the Hadamard product. We trained each weight matrices W, U and b.

We train an LSTM model for extracting features of steps respectively and they represent each steps even they have different lengths. Given X, h_k^t is the hidden state of k^{th} step at time $t \in \{0, \cdots, t_k\}$, where $h_k^t \in R^c$, c is the number of LSTM units in the hidden layer. The final state $h_k^{t_k}$ is used as the feature of k^{th} step. The concatenate layer on top of the LSTM layer of each steps is used to combine each step features to feature vector of sensor data, i.e. $concatenate_i = \{h_0^{t_0}, \cdots, h_n^{t_n}\}$. From various length inputs, we get fixed size output of each LSTM that represents entire time series data with steps. Then we use fully connected layer to encode concatenate vector to obtain v^{out} which is a small dimensional vector to represents reducted dimensional space. We can get final output vector that contains each steps features and features among each steps.

4.2.2 t-SNE

We gather features through LSTM and fully connected layer,

$$V^{out} = \{v_1^{out}, v_2^{out}, \cdots, v_k^{out}\}.$$

We use t-SNE not only for dimension reduction but also visualize to check low dimension distribution. While optimize t-SNE process, we need to conserve close sensor pairs which get high value with the Pearson correlation coefficient. Moreover, we need to preprocess the data to compare their similarity in the high dimensional space. Therefore, we preprocess the data to make signals with same length and normalize with same amplitude. This preprocessing only uses for calculating high dimension probability space. During training, LSTM and fully connected layer uses X_i as input to obtain the v_i^{out}, and learn towards to minimize KL divergence between original space and small dimensional space. We let P denote target distribution which is representing high dimensional space with preprocessed X and Q denote output distribution which is representing low dimensional space obtained by V^{out}. As in [12], we compute the conditional probability p_{ij} with $p_{ij} = \frac{p_{j|i}+p_{i|j}}{2k}$ where

$$p_{j|i} = \frac{\exp(-\|x_i - x_j\|^2/2\sigma_i^2)}{\sum_{l\neq i}\exp(-\|x_i - x_l\|^2/2\sigma_i^2)}.$$

In above equation, σ_i is the variance of the Gaussian that is centered on datapoint x_i. t-SNE aims to learn the distribution Q where

$$q_{ij} = \frac{(1 + \|v_i^{out} - v_j^{out}\|^2)^{-1}}{\sum_{m,n}(1 + \|v_m^{out} - v_n^{out}\|^2)^{-1} - 1}$$

We define our objective as a KL divergence loss between the output probability q_{ij} and the target probability p_{ij} as follows:

$$KL(P||Q) = \sum_{i\neq j} p_{ij} \log \frac{p_{ij}}{q_{ij}}$$

4.2.3 Full Model Architecture

The data we are going to handle are split with steps and each step have to be investigating locally and globally along with whole data. Therefore we consist LSTM for each steps first. Moreover, each steps have different features, for example the first time step have steady state before operation starts and the next time signal can be dramatically move with operation.

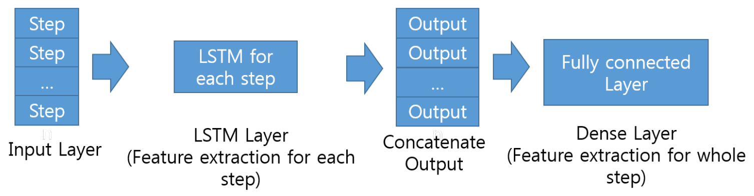


Figure 4.2: Feature extraction model for time series data.

Therefore we train LSTMs for each steps separately to be able to consider each steps features. Then concatenate those outputs and pass through fully connected layers to get the feature vector of whole signal data. We take low dimension as two and train our model towards to minimize KL divergence loss with adam optimizer. We run on batch as sensor data within process of a same product for preserving simple correlation among them. The full modle architecture can be seen as Figure 4.2.

Pseudo code for Deep Correlation Mapping as follows :

Algorithm 1 Deep Correlation Mapping Pseudo Code

```
Input: Recipe - Wafer1 - Sensor1 = (Step1, Step2, ..., Step n)
                        - Sensor2 = (Step1, Step2, ..., Step n)
       ...
                - Wafer2 - Sensor1 = (Step1, Step2, ..., Step n)
Output: low-dimensional data representation (Q)
 1: Preprocessing : Make all signal data same length, and same amplitude.
    (P)
    LOOP Process
 2: for wafer in Recipe : do
 3:     Compute distribution of Original high dimensional data (P)
 4:     for number of epoch do
 5:         Achieve final output using Deep learning Model
 6:         Run T-SNE with final output
 7:         Compute distribution of low dimensional data (Q)
 8:         Compute cost function ( $KL(P \parallel Q)$ )
 9:         Update weights in Deep learning Model
10:     end for
11: end for
```

As in algoritm 1, we can build end-to-end deep learning model with differet length dataset and our model conserve the simple correlation property. Correlations between sensors are defined using Euclidean distance in low dimensional space which is achieved results from DeepCorr model.

$$Correlation(X_i, X_j) = 1/(1 + \alpha d(out_i, out_j))$$

We set $\alpha = 10$ in our experiment.

4.2.4 Anomaly detection using Deep Correlation Mapping

We use Deep Correlation Mapping model for anomaly detection. We trained whole sensor signal data of different wafer with the same recipe and the results that projected in low dimensional space apply to do anomaly detection. If the sensor data is abnormal one, the result in low dimensional space will locate far from other normal data results. Therefore, for same sensors with different wafers, we calculate variance of the each sensor results and if it is bigger than threshold then we alert the sensor as abnormal one. If data is from normal sensor, they put in similar locations and if not, they put in various regions in low dimensional space.

4.3 Experimental Results

In our deep learning model, we have an option about LSTM with using a shared LSTM or different LSTMs for each steps. If each steps are regarded as similar features, we can use shared LSTM and it will be effective for reducing memory cost. However, our dataset have different features for each steps, so we get better result when we train the model with different LSTMs.

4.3.1 Correlation

As in Figure 4.3, simple pearson correlation coefficient map shows us that only few sensors are related and almost sensors do not have common features. However, there are some of sensors which have common features as properties and even look similar but with small correlation due to noise and delayed time movement.

Therefore, we compare our results with the Pearson Correlation Coefficient to check 1) DeepCorr model have similar coefficients with high Pearson Correlation Coefficient. 2) DeepCorr model can handle highly correlated sensors that have low correlation in Pearson Correlation Coefficient.

The heat map in Figure 4.4 shows us that not only nearby sensors, but also various sensors can be highly correlated with proposed model. The plots

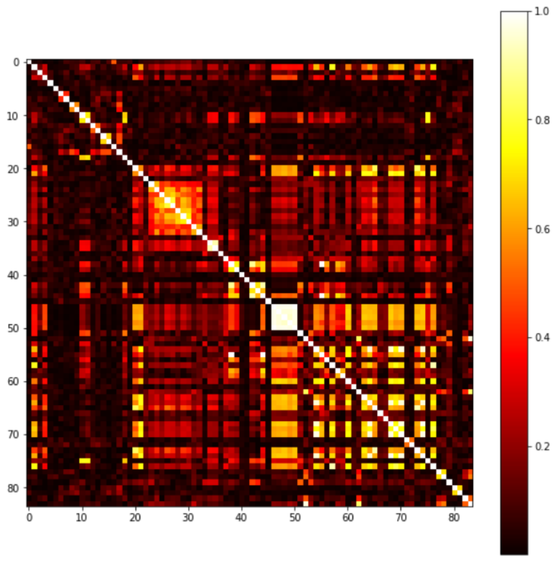


Figure 4.3: Heat map for Pearson correlation coefficient.

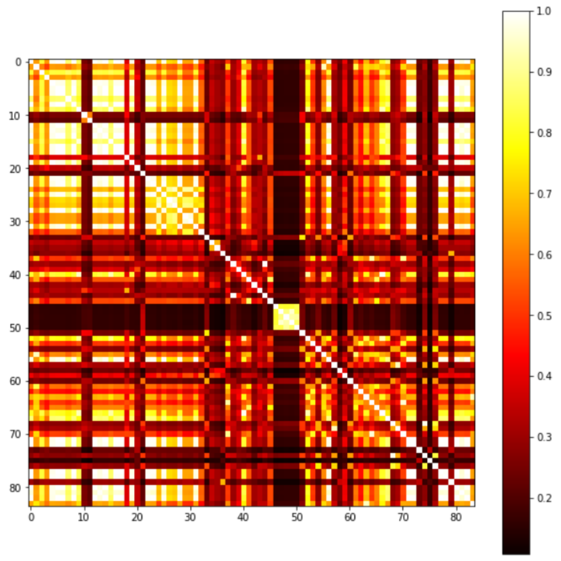


Figure 4.4: Heat map from Deep Correaltion Mapping result.

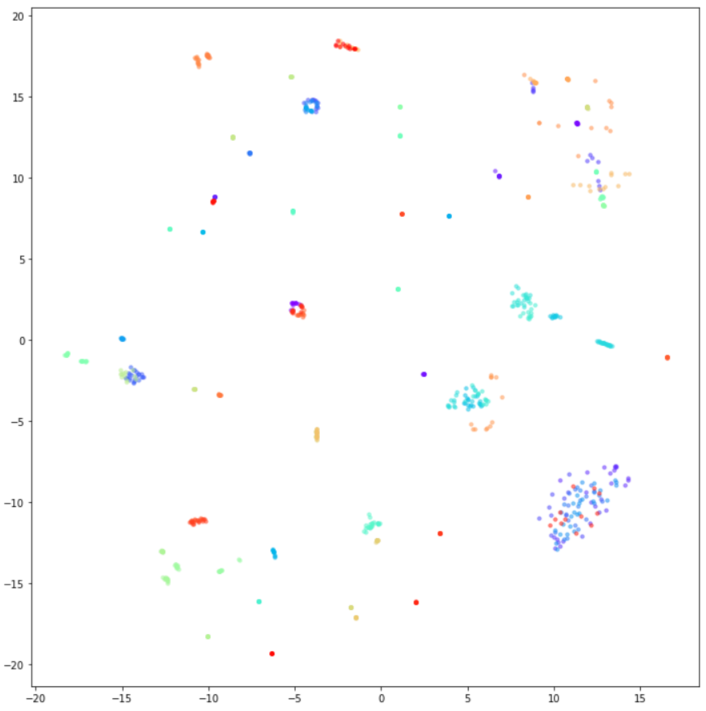


Figure 4.5: t-SNE map from original data.

in the figure show that our model can be used for noised or time delayed data to find out correlation between them since our model gather features of each steps while preserving the original time data.

The Figure 4.6 shows the visualization the results of low dimensional space from original high dimensional space which is obtained through our model after training. For understanding visualization easily, we set low dimensional space as 2-dimensional space and same sensors with same color. As we analyzed with correlation heat map, same sensors are only grouped in low dimensional space in Figure 4.5 when we plotted with preprocessed raw data. It seems sensors are not quite related and they are just seperating even on same process with correlated work. After we trained our model and get output to 2-dimensional space, some of sensors flock according to their correlation with preserving group with same sensors. Therefore, we can analyze our result to find out correlation among sensors.

The following examples of some specific high relative sensors will show us difference between simple correlation model and our model.

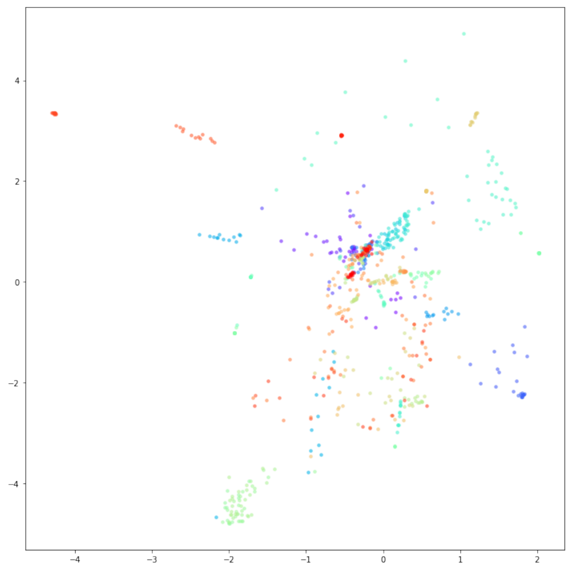


Figure 4.6: t-SNE map from Deep Correlation Mapping.

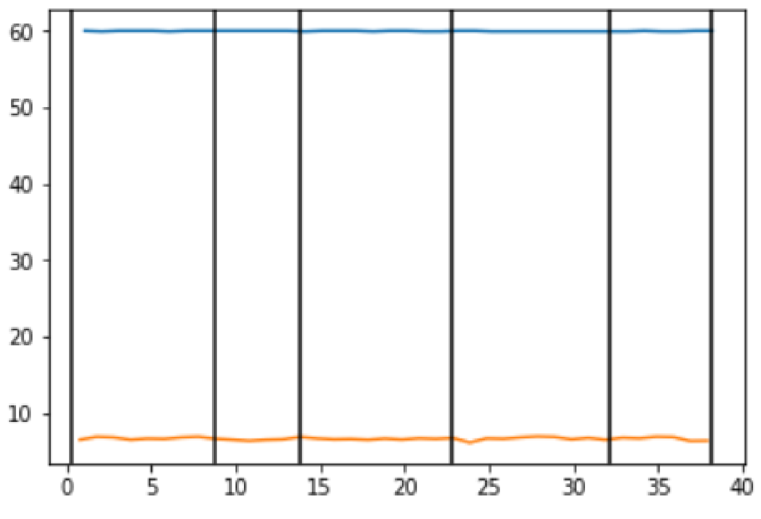


Figure 4.7: An example of related sensors.

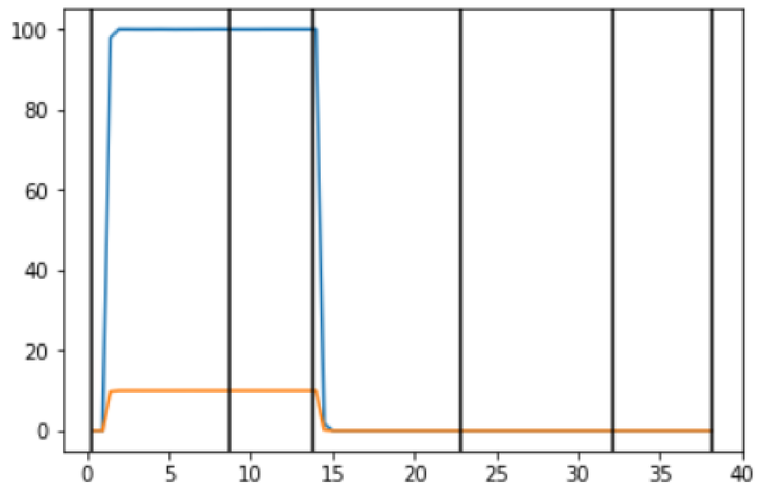


Figure 4.8: An example of related sensors.

At Figure 4.7, there are two different sensors which look flat shapes but they have pearson correlation coefficient -0.5 because of small noises in each steps. We can preprocess denoising preprocessing to compare flat sensors with noise efficiently but we cannot define the noise is small enough to ignore or small but important value. Therefore we let our model to decide how to handle noise by learned from whole data. As a result, our model correlation value is 0.76 and it means the noise is not an important feature for seperating those sensors as low correlated sensors. This example shows that our model can figure out noise which should be ignored to calculate correlation between them.

At Figure 4.8, two different sensors have pearson correlation coefficient 0.69 which is high but not as we expected and the correlation value comes from their slight delayed time at peaks. In sensor signals, some of sensors depend on other sensors and move after other sensor signals or before. Therefore simple correlation method have a problem to handle these pairs of sensors. We can fix this problem through synchronize with peaks but then other parts also modified even they have important features. To handle this problems, our model encodes each steps features to vectors and gather them to reperesent the sensor features. Therefore, our model succeed to find out high correlated sensors like Figure 4.8, by giving us correlation value as 0.91. This example shows us even delayed time difference, our model used LSTMs for each steps and they can extract features then gather them to find out their

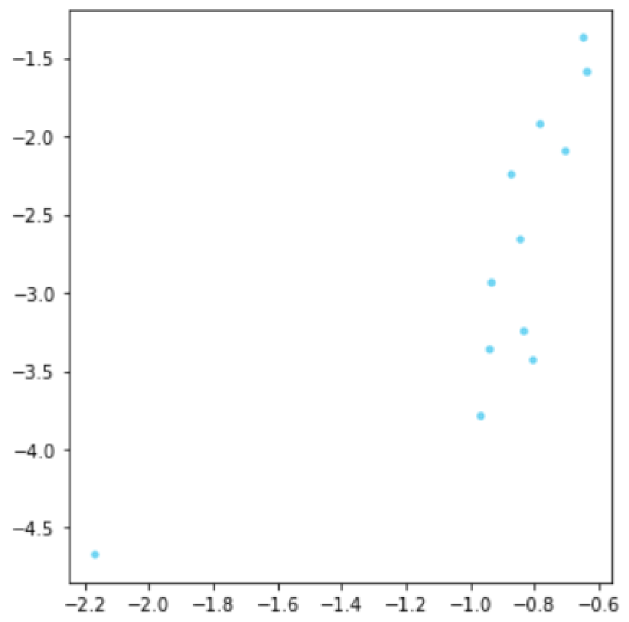


Figure 4.9: t-SNE map of an abnormal sensor.

correlation efficiently.

4.3.2 Anomaly detection using DeepCorr

We defined correlation between sensors upon low dimensional space through our model and as we mentioned previous subsection, the result preserve simple challenges like similar shape correlation property and complex challenges like delayed time property. As we mapped in Figure 4.6, t-SNE map shows us most of sensors are grouped together but some of them are outside the group appreciably like Figure 4.9. As in Figure 4.9, one data quitely far from other data even they are obtained from same sensor and it implies we need to check this sensor as abnormal one.

Therefore, signal data of same sensors project in almost similar points in the low dimensional space but few of them do not which can be regarded as abnormal data. In our experiments, if the variance of the results of same sensors are bigger than the threshold (we take 0.5 with Euclidean distance) we regard it as abnormal one.

Figure 4.10, and Figure 4.11 show abnormal sensors that we obtained from our experiments.

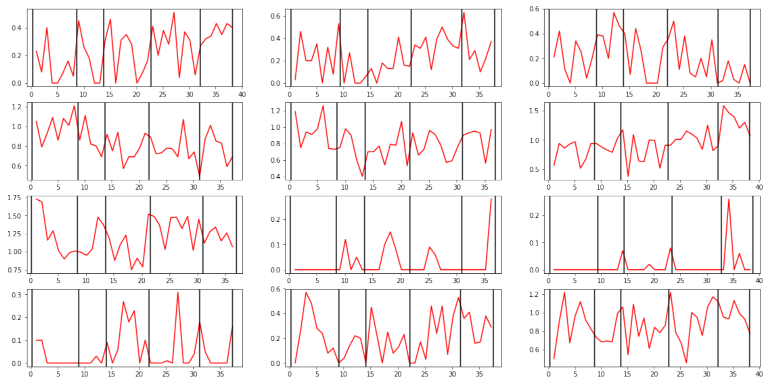


Figure 4.10: An example of noisy abnormal sensor data.

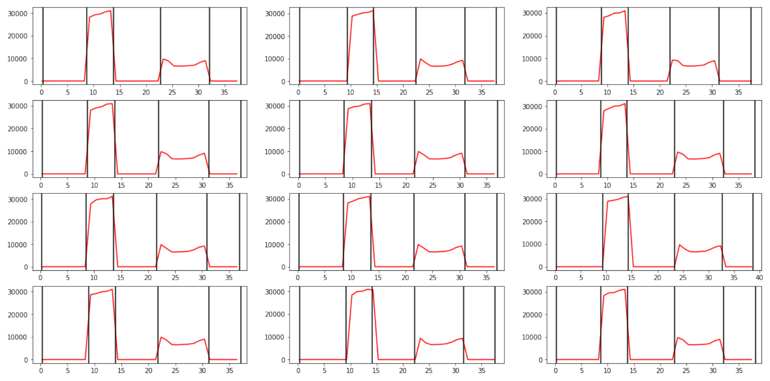


Figure 4.11: An example of different time abnormal data.

The abnormal sensor in Fig 4.10. are caused by different shapes and that make this sensor data have various points distribution in low dimensional space with our model results. They are not close even they are named same sensor. In this result, it can be regarded as our model still preserve simple correlation with small coefficient. On the other hand, sensor data in Fig 4.11. seems like quite similar shapes between them but it is slightly different time to peak at each steps. We can detect this differences with spread points in low dimensional space because of investigating each steps properties and then gather it to compute correlation. Their difference in low dimensional space is caused by difference in high dimensional original space. If we regard the data without considering steps they have similar shapes with high correlation coefficient. However, our model consider not only for the entire shape but also time difference at each time steps we can find abnormal sensor data successfully.

4.4 Conclusion

In this chapter , we suggests a new model, called Deep Correlation Mapping, to find out relations between signals and make an application to anomaly detection. Deep Correlation Mapping works well not only for noisy signals but also for time delayed sequences that cannot be analyzed by traditional correlation methods since our model captures each steps features with LSTM which can handle inputs with various lengths and various featurese due to its properties. We define a new correlation for low dimensional space derived from high dimensional space through our model regardless their various lengths and different time steps. Finally, our method can also suggest a new approach to recognize abnormal signals using the correlation. While our model can be used for time series anomaly detection, the model can be used to clustering various time length dataset.

Chapter 5

Trend Prediction

Predicting the trends of financial markets is one of the most important tasks for investors. Prediction of the trends is useful not only for real investing but also analyze general direction of other indexes in financial markets. They have tried to predict the trends using various methods like technical analysis and fundamental analysis. Technical analysis is one of the traditional methods that uses historical stock prices and trading volumes to determine the trend of future stock movement. This analysis is based on supply and demand in financial markets and easy to build a model since this approach only considers numerical parameters. Fundamental analysis predicts stock prices by using intrinsic values. The stock values are determined by financial news or sentiment of market and economic factors. Investors estimate the profits of firms and evaluate whether they are proper. Methodologies to forecast stock prices have researched for a long time and a number of techniques in various academic fields have been proposed and applied in real markets. For quantitative methods in finance are using machine learning techniques frequently.

In these days, deep learning has been widely used in classification problems in taking advantage of nonlinearity and has given out of standing performances than other previous classification methods. Some researches have compared deep learning with time series models for predicting time series data. For example, Kohzadi [17] compared the performance of Artificial Neural Networks(ANN) with that of AutoRegressive Integrated Moving Average(ARIMA) model for forecasting commodity prices. Kara [14] have applied ANN and Support Vector Machines(SVM) to predict Istanbul Stock Exchange(ISE) National 100 Index prices and also ANN shows better accuracy than SVM with polynomial kernel respectively. Deep learning tech-

niques for time series data, especially using LSTM model, have shown better results than previous machine learning techniques in speech recognition [9], sentimental analysis [23], and time series prediction. Moreover, attention mechanism is widely used for analyzing data both in images and time series data [34] and leads us to better results when we combine attention with LSTM than other plain deep learning models and moreover attention networks help us to be able to visualization as plotting attention vector weights for understanding the model.
In this paper, we predict the trends of KOSPI 200 with various deep learning models and compare them with accuracy. The data set consists of various index parameters like currency, global index, and commodities. We trained data from start of 2007 to end of 2016 and test data from start trading day of 2017 to end of July, 2018. We get the best trends accuracy with weighted attention networks and also explain the reason of prediction through visualizing attention vectors that we used.

5.1 Related works of Trend Prediction

Kohzadi [17] tested ANN and ARIMA model for forecasting commodity prices and compare the results of them. The result was that ANN gave a 27% and 56% lower mean squared error than an ARIMA model. Kara [14] have applied ANN and support vector machines (SVM) to predict Istanbul Stock Exchange (ISE) National 100 Index prices. Ten technical indicators were used as inputs and they got maximum 75.74% and 71.52% in ANN and SVM with polynomial kernel respectively. The inputs were only based on the technical factors which use historical index prices and volume data. However, experimental procedures in [17] and [14] are not practical to investors in that training and test data was used without considering time series data. Training sets should not be later than test set in analyzing time series data because of high correlation between training set and test data set can be occured. It means to invest in real markets, we need to define training set with previous dates sooner than predict date of time series data to hide test data set from the model definitely. In paper [25], they compare various models include traditional machine learning techniques like ARIMA, SVM and deep learning techniques like DAE (Denoising Autoencoder) and mixture both of them. From their results in paper, DAE-SVM shows higher results than other methods and they normally get better result with time series model than simple machine learning models. In paper [24], they predict KOSPI 200 index with SVMs and ANN with google trends. They got the best result 52% accuracy with shorter periods and using google trends. However, their accuracy is not enough for who wants to real investment. Although above papers can be explained as deep learning technique is more effective than other machine learning techniques in extracting high level representation of input features, so that it can enhance their performances.

Sometimes 1D CNN show better performances for classifying sequential data. A convolutional neural network (CNN) widely used in image tasks, for example, image classification, image segmentation, denoising, super-resolution, and etc. In these days, not only for image area, but also 1D CNN is very effective to derive important features from segments of a whole sequential data and where the location of the feature within the segment is not that important. This applies well to the analysis of sequential sensor data, fixed length periodic signals, and NLP. In recent papers [8] and [27] describe how to learn semantically meaningful representations of sentences by using CNNs in NLP. The models given in the papers recommend potentially interesting documents to users based on what they are currently reading. In [16], evaluates a CNN architecture on sentiment analysis and topic categorization tasks. The 1D CNN architecture achieves a remarkable performance than previous papers. Also, the network used in this paper is quite simple and easy to implement.

Time series classification tasks have been accomplished with recurrent neural networks in these days. Especially LSTM is widely used in sequential data like sequence labelling [15], speech recognition [9], anomaly detection [21], and also in financial time series prediction [4]. Many kinds of time series problems have used simple LSTM model or stacked LSTM model to predict their goals successfully. However, LSTM model have some problems with vanishing gradient and losing simple features in long sequences even their advantages for time series data. Vaswani [31] suggested "attention" mechanism and they told us with using attention is useful for many ways and solve those problems. Attention is a simple vector, sometimes represents probability distribution using softmax function. Recurrent neural networks or other deep learning models should take an input as a complete sequential data and compress all information into a fixed-length vector as output of the model before. It implies even a data with hundreds length represented by fixed-length or sometimes only output of final time step, and maybe the output is much shorter than an input length, will surely lead to information loss. However, attention partially fixes this problem. It allows model to analyze all the information of the original data, then generate the proper output. Attention network is getting widely used in these days image captioning [32], neural machine translation [3], question and answering [35]. In a paper

[31], they build multi-head attention module for replacing the recurrent or convolution neural networks most commonly used in encoder-decoder architectures. The attention model comes between the encoder and the decoder and helps the decoder to pick the encoded inputs that are important for each step of the decoding process. They showed a better performance for translation task than previous machine translation tasks with lower computational cost. The model not only trained faster but also outperformed even all previously reported ensembles.

5.2 Trend Prediction with Deep Learning Models

We will build multiple deep learning models with multi-layer perceptron (MLP) model, 1D convolutional neural network (1D CNN), stacked Long-short term memory (stacked LSTM), attention networks and weighted attention networks. Those methods are quite popular for sequential data tasks and shows better results than traditional machine learning techniques in various tasks. In this section, we will introduce our datasets and then describe details of various methods.

5.2.1 Dataset

The KOSPI 200 index is a weighted combination of the 200 most traded securities in Korea Stock Exchange market. We take inputs as change ratios of KOSPI 200 and various indexes like currency, commodity and global indexes, which are quite related with Korean financial market in a view point of fundamental analysis. We gather our data using pandas-datareader [1] and calculate a return of a day as $r[t+1] = \frac{close\ price[t+1] - close\ price[t]}{close\ price[t+1]}$ and initial data $r[0]$ is given. We set look back days as p trading days of target index and if a day is not include in trading days of target index, we remove the day and re-calculate the return with a previous close price (We take p=10 in this paper). We can write our input $x[t] = [R^t[0]; \cdots ; R^t[p]]$ where $R^t[i]$ is a collection of input indexes return at the day t. The target output is defined by KOSPI trends whether larger than 0 or not. We will build up binary classification model by analyzing time series data as input. We predict q trading days (we take q=19 in this paper) after the final day of each input and define trend as

$$trend[t] = \frac{close\ price[t+q] - close\ price[t]}{close\ price[t]}$$

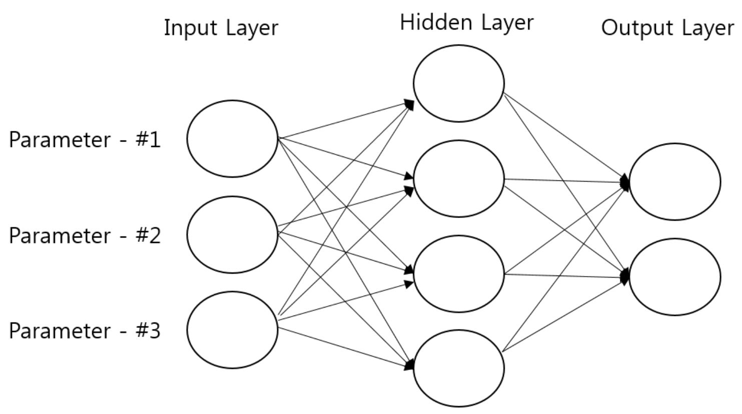


Figure 5.1: An example of a simple Multi layer perceptron with one hidden layer which consists of 4 hidden neurons.

and target label is

$$y[t] = \begin{cases} [1, 0] & if\ trend[t] < 0 \\ [0, 1] & otherwise \end{cases}$$

that we describe as an one-hot vector.

5.2.2 MLP

Multi-layer perceptron consists of fully connected layers, at least three, and each layers, except input and output, contains hidden neurons inside which are trainable. Figure 5.1 shows an example of a MLP model with one hidden layer with 3 neurons. Neurons in hidden layer take the values of inputs parameters, sums them up with multiplying assigned weights, and adds a bias. By applying the transfer function, the value of the outputs would be determined. The number of neurons in input layer corresponded to the number of input parameters [22]. In paper [2], they got better result in compounded annual return with MLP rather than a standard factor model. Sezer[26] predict Dow30 stocks with the model trained data of the daily stock prices between 1997 and 2007 and tested with data from 2007 to 2017. They achieve comparable results against the Buy and Hold strategy in most of the cases with setting the most appropriate technical indicators. We build a simple MLP model with 4 layers and each hidden layer contains 64 hidden neurons for our experimental results. Notice that an input of MLP model should be a

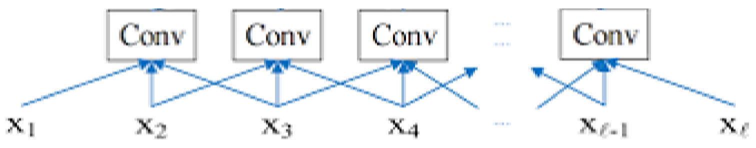


Figure 5.2: An example of a a 1D-CNN with sliding kernel size 3.

vector, so we flatten the input matrix to make an input vector which is going to feed in our model.

5.2.3 1D-CNN

Many applications of convolutional neural networks had focused on images area after imagenet [18] shows successful results for classifying natural images and quite better accuracy than other image classification methods. However, because of time series property that current state relates with before state, only recurrent neural networks used for analyzing time series analysis. In a paper [16], they did sentence-level classification tasks, which include sentiment analysis and question classification with 1D-CNN and they got improved results. A 1D Convolutional Neural Network is expected to capture the data locality well with the kernel sliding across the input data. In Figure 5.2, a sliding kernel with size 3 do convolution operation over input parameters and get output of each locations. We should choose hyper-parameters like kernel size, number of kernels for better results, since output we get from 1D CNN should represent local patterns and types of pattern can be found in various types of kernels. One-dimensional CNNs work with patterns in one dimension, and tend to be useful in signal analysis over fixed length signals. We tried a bunch of 1D CNN models and finally, we recommend the model with 2 convolutional layers and 3 fully connected layers to predict target labels.

5.2.4 LSTM

LSTM [12] is one of the most popular module of recurrent neural networks. The networks are widely applicable for various aspects about sequential data problems like speech analysis, sentimental analysis, voice recognition and financial analysis because of its characteristics that can prevent features that important along with whole sequence by long-term memory and keep short-term memory as well as simple recurrent neural networks. In [6], they used LSTM networks for predicting movements of S&P 500 from 1992 until 2015.

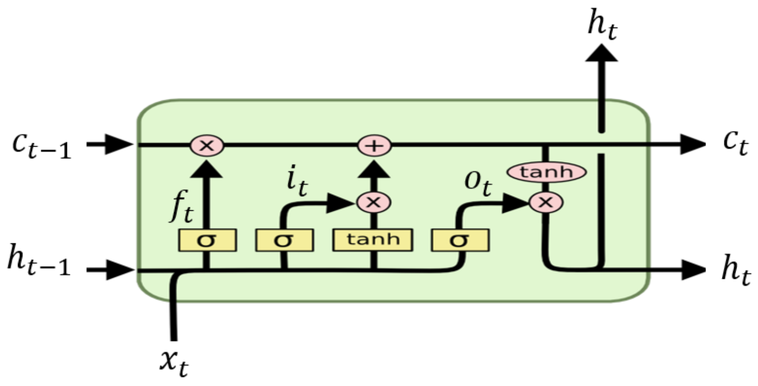


Figure 5.3: An example of unit of LSTM.

LSTM model outperform memory-free classification methods, i.e., a random forest, MLP, and a logistic regression classifier. [11] suggests LSTM model to forecast stock prices and indexing problems. Deep learning outperform than other classical models because of nonlinearity and overcome in-sample approximation quality.

One unit of a time can be figured as in Figure 5.3, and the simple forms of the equations for the forward of an LSTM unit with a forget gate as follows:

$$f_t = \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \tag{5.2.1}$$
$$i_t = \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \tag{5.2.2}$$
$$o_t = \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \tag{5.2.3}$$
$$c_t = f_t \odot c_{t-1} + i_t \odot \sigma_c(W_c x_t + U_c h_{t-1} + b_c) \tag{5.2.4}$$
$$h_t = o_t \odot \sigma_h(c_t) \tag{5.2.5}$$

where f_t is a forget gate, i_t is an input gate, o_t is an output gate, c_t is a cell state, h_t is a hidden state, σ is an activation function and the operator $\odot$ denotes the Hadamard product. We trained each weight matrices W, U and b.

We will train our model with stacked LSTM to understand more complex features among parameters of the data. Stacked LSTM is widely used in sequential data like speech recognition [9], anomaly detection [21], and also in financial time series prediction [4].

5.2.5 Attention Networks

The core of sequential data model is to assign a probability of a data by Markov Assumption. Due to the various lengths of inputs, RNN is naturally introduced to model the conditional probability among times like Markov chain model.

$$P(w_1 w_2 \cdots w_n) \approx \prod_i P(w_i \mid w_{i-k} \cdots w_{i-1})$$

Usually, vanilla RNN have common problems with gradient vanishing/ exploding problems and structure problem like ordering. Similar to add fully connected layer, we set a same number of hidden neurons with previous input length. The attention model helps to pick only the inputs of previous layer that are important for each step of the rest of model. Once we calculate the importance of each encoded vector, we normalize the vectors with softmax and multiply each encoded vector by its weight to obtain a "time dependent" input encoding which is fed to each step of the decoder RNN. There are a few kinds of attention depends on how attention is defined. The alignment weights are learned and placed "softly" over all of the input is called "Soft Attention" and on the other hand, if only selects one part of the input to attend to at a time is called "Hard Attention". Soft attention makes model smooth and differentiable but expensive when the input is large. Hard attention has a low computational cost but requires more complicated techniques such as reinforcement learning to train since the model is non-differentiable. In our experiments, we use soft attention module as in FIgure 5.4 and equations are followings.

$$e_t = tanh(W_a[x_1, x_2, \cdots, x_T] + b) \tag{5.2.6}$$

$$\alpha_t = \frac{exp(e_t)}{\sum_{k=1}^{T} exp(e_k)} \tag{5.2.7}$$

In those equations the attention probabilities $\alpha = (\alpha_1, \ldots, \alpha_T)$ softmax output of a vector based on the input sequence x_t and the trainable weights matrix W_a. Then we get output of attention as $[c_1, c_2, \cdots, c_T] = [x_1, x_2, \cdots, x_T] * [\alpha_1, \alpha_2, \cdots, \alpha_T]$.

Intuitively, this vector summarizes the importance of the different elements in the input. We will visualize this by plotting an attention vector as a bar. We try various attention networks like attention before LSTM networks, after lstm networks and attention for time aspects, factor aspects or both. For final results, we decide to use attention networks for time and

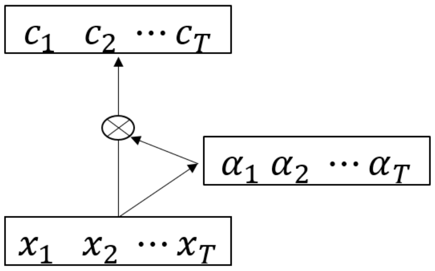


Figure 5.4: Soft attention networks for input vector.

factor aspects both, then multiply attention vectors to get final vectors as an input data for LSTM networks. Before going to the next lstm network layer, we get output with attention networks as following equations :

$$e_{j,t} = V_a \cdot tanh(W_a s_{t-1} + U_a h_j) \tag{5.2.8}$$

$$\alpha_{j,t} = \frac{exp(e_j)}{\sum_{k=1}^{T} exp(e_k)} \tag{5.2.9}$$

where similar with previous equations but input sequence h_j and the internal hidden state of the output cell s_{t-1} and trainable matrix V_a appear. After attention networks with LSTM, we run LSTM one more as stacked LSTM network then pass through fully connected layers to get final prediction. Our full model architecture is in Figure 5.5.

5.2.6 Weighted Attention Networks

After we build the attention network model as in above subsection, we try another model, weighted attention networks with modified loss function. Since we train various models with categorical cross entropy as loss function and labels are defined with one-hot vector, our trained model are focused only on hit ratio. However, in financial markets, we need to care about how much is a change ratio of the day and we want to do right prediction on a large change day than a small change day. Therefore, we custom our loss function with multiplying absolute value of the change ratio of test data and original categorical cross entropy value. Categorical cross entropy function for binary class is defined as

$$H_{y'}(y) := -\sum_i (y'_i \log(y_i) + (1 - y'_i) \log(1 - y_i))$$

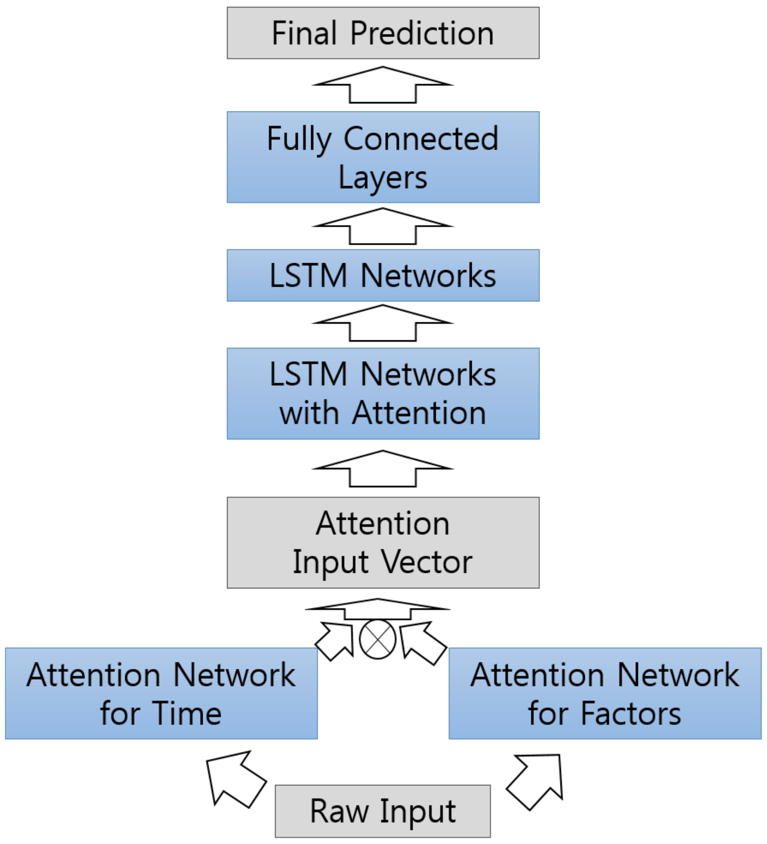

Final Prediction
Fully Connected Layers
LSTM Networks
LSTM Networks with Attention
Attention Input Vector
Attention Network for Time
Attention Network for Factors
Raw Input

Figure 5.5: Attention networks for KOSPI200 prediction.

where y_i is the predicted probability for class i and y'_i is the true probability. Our weighted categorical cross entropy is defined as

$$H_{y'}^{weighted}(y) := -abs(change\_ratio_i) * \sum_i (y'_i \log(y_i) + (1 - y'_i)\log(1 - y_i))$$

where $change\_ratio_i$ is the change ratio of the test data i. Therefore, our model updated to predict right at bigger change ratio data to minimize loss.

5.3 Experimental Results

We will train and test with various deep learning models which we metioned in Section 4. We define our measurement as hit ratio which is the precision of trends prediction and its formulation as followings:

$$hit\ ratio = \frac{\sum_{i=1}^{N} prediction_i}{N},$$

and

$$prediction_i = \begin{cases} 1 & if\ prediction_i \cdot real_i > 0 \\ 0 & otherwise \end{cases}$$

where $prediction_i$ denotes the prediction of i th sample change ratio, and $real_i$ denotes the real market change ratio.

We set our train data as before the test data and test data will be unseened in our model. Therefore we subsampled our training data set to use validation set with split ratio 0.7. (That is 70% for training and 30% for validation, randomly in whole train data set.) We set our train data from start trading day of 2000 until end of 2016 and test data from start trading day of 2017 to end of July, 2018. We choose the best model with highest hit ratio and then test the our target data.

5.3.1 Best Lookback Days

First, we do experiments to find out how many days we should lookback to get best accuracy for deep learning models that we suggest. We set our lookback days as 5, 10, 15, 20, 30, 60 days since 5 trading days are regarded as 1 week and it means we lookback 1, 2, 3 weeks and 1, 2 months for our intput data. For 5, 10 lookback days, we test days from the next day after lookback days to 5, 10 days after lookback days and for 15, 20, 30, 60 days we predict days the next day after lookback days to 1 week, 2 week and so

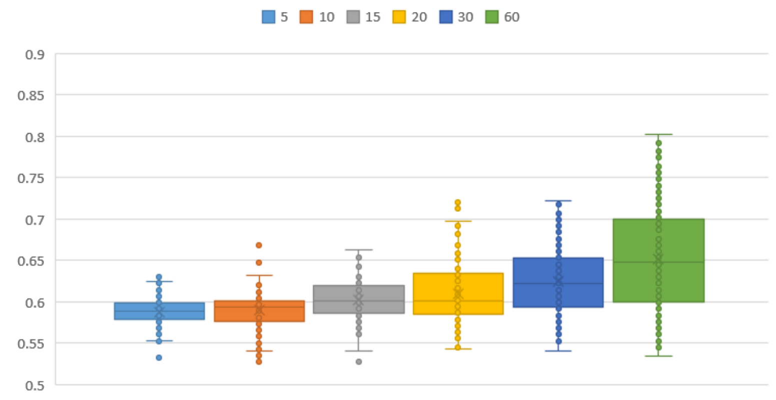


Figure 5.6: Hit ratio with respect to lookback days.

on until predict days reach to same days as we lookback. For example, if lookback days are 15 and prediction days are 10 then our input data is 3 weeks information and we predict a trend over 2 weeks after the last day of our lookback days.

In Figure 5.6, each bar from right to left imply hit ratio of 5, 10, 15, 20, 30, and 60 lookback days. This result is from the best hit ratio for test data with various deep learning models and we do not use the best model for validation data since we are focused on finding out the best lookback days. As we can check in Figure **??**, longer lookback days get better hit ratio and especially on 60 lookback days, our best model for test data approaches almost 80% and that is quite high hit ratio in financial prediction. Therefore, in next subsection, we try to compare various deep learning models with 60 lookback days and various prediction days.

5.3.2 Results of Various Deep Learning Models

As we metioned in above subsection, we will test various deep learing models with 60 lookback days and prediction days from the next day to every weeks until prediction days reach as 60 days. In this experiment, we get hit ratio with the best model for validation data since we want to know which model is the best model without seeing the test data. We trained 5 times for each models to check stability of each models and they got similar hit ratio results. The Figure **??** represents the average of hit ratio of each models with various prediction days.

As we can see at Figure 5.7, attention network outperforms than other models for 60 lookback days. For long time series prediction, we assume that

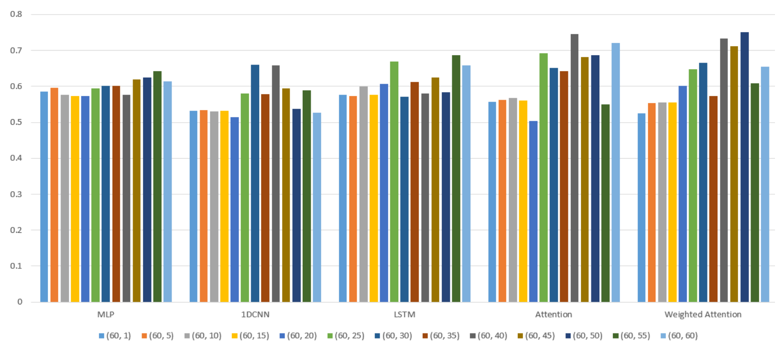


Figure 5.7: Hit ratio with respect to prediction days with various deep learning models.

LSTM has advantages because of its properties with memory cell and LSTM shows better hit ratio than MLP, and 1D CNN which are simple methods. In paper [24], they got best hit ratio results as 0.52 when they use ANN and SVMs with or without google trends. (We assume our MLP model and ANN model in paper [24] are quite similar because their ANN accuracy 0.51 is almost same with our MLP result.) With both attention models (weighted or not), we got best results when we predict 40 days after of our input data ended. We got hit ratio with the test data set 0.715 with attention networks and 0.763 with weighted attention networks with our best validation models. Moreover, we analyze more about those models with positive and negative trends and considering when large change ratio occurs comparably.

	Dataset	Attention Networks	Weighted Attention Networks
Positive	0.602	0.709	0.825
Negative	0.398	0.723	0.671
Total	1.0	0.715	0.763
Earn points	1525.20	656.236	989.724

Table 5.1: Best Model Results.

In table 5.1, our test data set has positive, and negative trend as 6:4 and if we correct all the up and down trends, we can earn 1525.20 points. We got better accuracy with weighted attention networks at positive trends and attention networks at negative trends. We can earn 50.8% more points than plain attention networks if we use weighted attention networks.

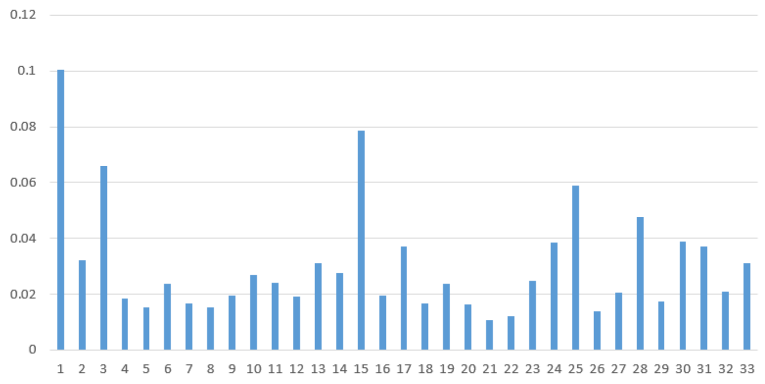


Figure 5.8: Attention vector of factors.

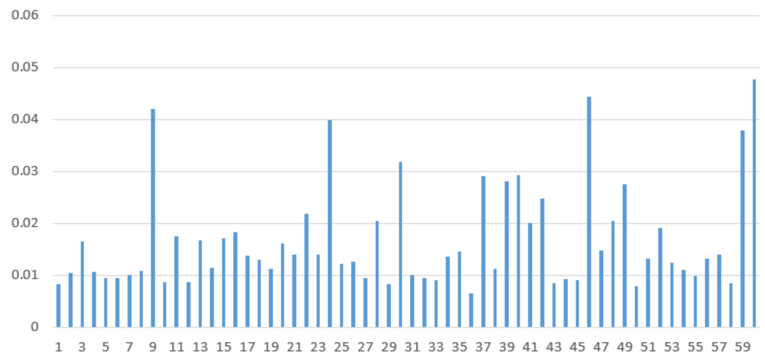


Figure 5.9: Attention vector of times.

5.3.3 Visualization Attention Vectors

In Section 3.5, we mentioned attention network has great advantages of visualizing attention vector to analayze which parameter our model look carefully with a higher weight. If we change input data, the attention vectors change together so in this paper, we just show one example of visualizing attention vectors when we put an input data with the highest change ratio.

There are two kinds of attention vectors which we used in our attention models. Figure 5.8, and Figure 5.9 are representing attention vector of factors and times, respectively when we put the input data with highest change ratio among our test data to weighted attention networks. For this case, with aspects of factors, currency index of dollar and global index S&P 500 are the highest weights factors and with aspects of times, the last day of input data and a few days in the middle have highest weights. From a view point of times, in this example, our result affected most by the last day of lookback days and middle of days but only 4% of whole weights is the largest value. It is still hard to analyze in details with these vectors since they got complex relations and they pass through our model to decide the trend is going up or down but we can figure out which is the most affective factor and time intuitively through visualizing attention vectors.

5.4 Conclusion

In this chapter, we tested various deep learning models for predicting the trends of KOSPI 200 index. We tried MLP, 1D CNN, LSTM and Attention Networks which are widel used in sequential data applications. While short lookback days have low hit ratio with various models, long lookback days with 60 trading days gave us higher hit ratio with various models. It implies look more days with input data, better accuracy with hit ratio. With 60 trading days as lookback days, 40 trading days as prediction days get highest hit ratio with Attention networks model. Not only to get the highest hit ratio, but also get the highest earn points when we use weighted attention networks since loss function is minimized when the model get more accurate at higher change ratio. With our experimental results, we can confirm with following statements. First, LSTM works well with sequential data which depends on time than MLP and 1D CNN. Second, longer lookback days, higher probability to get better hit ratio and normally better results with the model which have LSTM networks inside. Finally, weighted attention networks works better for long sequential data and have an advantage of visualization for analyzing the model intuitively.

Chapter 6

Conclusion and Future Works

In this thesis, we introduces various kinds of deep learning models for time series data and apply to two different industrial areas. First, we suggest a new deep learning model which is an ensemble model of LSTM networks and t-SNE, which is called Deep Correlation Mapping, for anlayzing and anomaly detection time series data. Not only we got reasonable correlation between sensor data, but also we can do anomaly detection with various length inputs even they have noise and time delayed correlation. Second, we tested various deep learning models to predict trend of KOSPI 200 index. Our weighted attention networks achieved the highest hit ratio 0.76 which can be used in real investment. Also, we can visualize attention vectors to understand the reason why the model choice a direction of financial market intuitively. We will keep researching with time series data widely such as smart factory and healthcare.

Bibliography

[1] Panda datareader link. `https://pandas-datareader.readthedocs.io/en/latest/remote_data.html`. Accessed: 2018-11-01.

[2] John Alberg and Zachary C Lipton. Improving factor-based quantitative investing by forecasting company fundamentals. *arXiv preprint arXiv:1711.04837*, 2017.

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

[4] Wei Bao, Jun Yue, and Yulei Rao. A deep learning framework for financial time series using stacked autoencoders and long-short term memory. *PloS one*, 12(7):e0180944, 2017.

[5] Wenhui Feng and Chongzhao Han. A novel approach for trajectory feature representation and anomalous trajectory detection. *2015 18th International Conference on Information Fusion (Fusion)*, pages 1093–1099, 2015.

[6] Thomas Fischer and Christopher Krauss. Deep learning with long short-term memory networks for financial market predictions. *European Journal of Operational Research*, 270(2):654–669, 2018.

[7] John Cristian Borges Gamboa. Deep learning for time-series analysis. *CoRR*, abs/1701.01887, 2017.

[8] Jianfeng Gao, Patrick Pantel, Michael Gamon, Xiaodong He, and Li Deng. Modeling interestingness with deep neural networks. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 2–13, 2014.

[9] Alex Graves, Abdel-rahman Mohamed, and Geoffrey E. Hinton. Speech recognition with deep recurrent neural networks. *CoRR*, abs/1303.5778, 2013.

[10] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[11] JB Heaton, NG Polson, and Jan Hendrik Witte. Deep learning in finance. *arXiv preprint arXiv:1602.06561*, 2016.

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

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

[14] Yakup Kara, Melek Acar Boyacioglu, and Ömer Kaan Baykan. Predicting direction of stock price index movement using artificial neural networks and support vector machines. *Expert Syst. Appl.*, 38(5):5311–5319, May 2011.

[15] Kazuya Kawakami. *Supervised Sequence Labelling with Recurrent Neural Networks.* PhD thesis, PhD thesis. Ph. D. thesis, Technical University of Munich, 2008.

[16] Yoon Kim. Convolutional neural networks for sentence classification. *arXiv preprint arXiv:1408.5882*, 2014.

[17] Nowrouz Kohzadi, Milton S Boyd, Bahman Kermanshahi, and Iebeling Kaastra. A comparison of artificial neural network and time series models for forecasting commodity prices. *Neurocomputing*, 10(2):169–181, 1996.

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

[19] Andrew L Maas, Awni Y Hannun, and Andrew Y Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. icml*, volume 30, page 3, 2013.

[20] Pankaj Malhotra, Anusha Ramakrishnan, Gaurangi Anand, Lovekesh Vig, Puneet Agarwal, and Gautam Shroff. Lstm-based encoder-decoder for multi-sensor anomaly detection. *CoRR*, abs/1607.00148, 2016.

[21] Pankaj Malhotra, Lovekesh Vig, Gautam Shroff, and Puneet Agarwal. Long short term memory networks for anomaly detection in time series. In *Proceedings*, page 89. Presses universitaires de Louvain, 2015.

[22] Amin Hedayati Moghaddam, Moein Hedayati Moghaddam, and Morteza Esfandyari. Stock market index prediction using artificial neural network. *Journal of Economics, Finance and Administrative Science*, 21(41):89–93, 2016.

[23] Hamid Palangi, Li Deng, Yelong Shen, Jianfeng Gao, Xiaodong He, Jianshu Chen, Xinying Song, and Rabab Ward. Deep sentence embedding using long short-term memory networks: Analysis and application to information retrieval. *IEEE/ACM Trans. Audio, Speech and Lang. Proc.*, 24(4):694–707, April 2016.

[24] Sujin Pyo, Jaewook Lee, Mincheol Cha, and Huisu Jang. Predictability of machine learning techniques to forecast the trends of market index prices: Hypothesis testing for the korean stock markets. *PloS one*, 12(11):e0188107, 2017.

[25] Xin-Yao Qian and Shan Gao. Financial series prediction: Comparison between precision of time series models and machine learning methods. *CoRR*, abs/1706.00948, 2017.

[26] Omer Berat Sezer, A Murat Ozbayoglu, and Erdogan Dogdu. An artificial neural network-based stock trading system using technical analysis and big data framework. In *Proceedings of the SouthEast Conference*, pages 223–226. ACM, 2017.

[27] Yelong Shen, Xiaodong He, Jianfeng Gao, Li Deng, and Grégoire Mesnil. A latent semantic model with convolutional-pooling structure for information retrieval. In *Proceedings of the 23rd ACM International Conference on Conference on Information and Knowledge Management*, pages 101–110. ACM, 2014.

[28] Dominique T. Shipmon, Jason M. Gurevitch, Paolo M. Piselli, and Stephen T. Edwards. Time series anomaly detection; detection of anomalous drops with limited features and sparse examples in noisy highly periodic data. *CoRR*, abs/1708.03665, 2017.

[29] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[30] Laurens van der Maaten and Geoffrey Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 9:2579–2605, 2008.

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

[32] Oriol Vinyals, Alexander Toshev, Samy Bengio, and Dumitru Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3156–3164, 2015.

[33] Junyuan Xie, Ross Girshick, and Ali Farhadi. Unsupervised deep embedding for clustering analysis. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, pages 478–487. JMLR.org, 2016.

[34] Kelvin Xu, Jimmy Ba, Ryan Kiros, Kyunghyun Cho, Aaron Courville, Ruslan Salakhudinov, Rich Zemel, and Yoshua Bengio. Show, attend and tell: Neural image caption generation with visual attention. In *International conference on machine learning*, pages 2048–2057, 2015.

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

국문초록

딥러닝은 최근 몇 년간 산업 수학에 있어서 가장 강력하고 중요시 여겨지는 방법이다. 우리는 산업 수학의 시계열 데이터에 있어서 딥러닝 모델을 분석 및 예측 등에 정의하였다. 첫 번째로, 이상 감지를 위한 새로운 딥러닝 모델을 개발하였으며 이는 다양한 길이 뿐만 아니라 노이즈, 시간 차가 있는 데이터에서도 엔지니어에게 필요한 시계열 데이터 분석을 할 수 있었다. 두 번째로, 금융 시장의 트렌드를 예측하기 위한 다양한 딥러닝 모델을 개발 및 시험해보았으며 이 중 가중치 어텐션 네트워크의 경우 높은 예측 정확도 뿐만 아니라 시각화를 통해 직관적으로 모델을 이해 및 예측한 이유를 분석할 수 있었다.

주요어휘: 딥러닝, 엘에스티엠, 어텐션, 시계열, 이상 감지, 트렌드 예측
학번: 2015-30123

감사의 글

지난 대학원 생활을 돌아보면 많은 시간과 그만큼의 우연이 있었지만 제가 졸업하는데 있어서 많은 사람들의 도움이 있었습니다.
가장 먼저 석사, 박사를 통틀어 지금까지 지도해주시고 제가 앞으로도 학문적으로 인간적으로 조언을 많이 부탁드릴 우리 선생님, 강명주 교수님께 무한한 감사를 드립니다. 선생님 덕분에 많은 기회를 얻어 다양한 연구를 진행할 수 있었고, 이를 바탕으로 다양한 경험을 얻고 졸업 및 새로운 사회생활을 순탄하게 시작할 수 있게 되었습니다. 무엇보다도 선생님이시기 때문에 짧지 않고 쉽지 않은 박사기간 즐겁게, 다른 사람들한테 자신있게 행복하다고 말하면서 대학원 생활을 할 수 있었습니다.
또한 이 연구실에 있으면서 같이 있었던 형, 누나, 친구, 동생들이 없었다면 여기까지 오기 쉽지 않았을 꺼 같습니다. 개인적인 노트는 따로 남기겠지만 제가 석사 때부터 지금까지 거의 모든 프로젝트를 항상 같이 진행해왔던 경민이형, 투박하지만 그래서 더 좋은 한울이형, 고등학교 대학교 대학원 연구실까지 오랜 시간 같이했던 준기형, 그리고 동생들이어서 고마운 형민이 성권이 경현이, 그 외 많은 연구실 사람들 덕분에 여기까지 오는데 큰 도움을 받았기 때문에 감사인사를 드립니다.
항상 저를 믿고 서포트해주시다가 이제 부안에서 아름다운 이야기를 써내려가시고 계시는 아버지, 어머니 그리고 제가 졸업 결심을 하게된 말씀을 해주신 할머니, 항상 일하느라 건강 걱정되는 형, 그런 걱정을 안심시켜주시는 형수님 그 외 친지분들께 드디어 졸업을 하게 되었고 그 동안 저 때문에 고생하셨다는 말씀 드리고 싶습니다.
마지막으로 졸업하는 해, 제 인생에 항상 같이 있기를 약속한 주영이에게 고맙고 사랑한다는 말을 남기겠습니다.
앞으로 학문적으로, 인간적으로 더 나은 김상연이 되도록 항상 노력하겠습니다.