國立政治大學應用數學系

碩士學位論文

利用 SVM 模型判斷股票資料的隨機性成分
Using SVM Model to Classify the Random Components of Stock Data

指導教授：曾正男　博士

研究生：賴彥儒　撰

中 華 民 國 110 年 6 月

DOI:10.6814/NCCU202100699

致謝

時間過得很快，我已經在政大待了 5 年，現在也要為政大的日子劃下一個休止符，前往台大面臨接下來的挑戰。在寫下這段感謝，腦中浮現了許多過去的回憶，利用我的文字表達感謝。

首先感謝我的指導教授：曾正男老師，謝謝他我在大三那一年就可以跟他做研究，大四帶著我一起發了一篇會議論文，並且一直給我鼓勵和支持，才讓我決定在政大應數系完成五年一貫學程。這段時間，他不遺餘力地幫助我，提供我許多的幫助，讓我學到了很多的東西。再來，感謝成大數學系的舒宇辰老師和系上的蔡炎龍老師，在口試時給我的許多建議，此論文方能更加地完整。感謝姜林宗叡學長，在我完成論文的過程中，提供了許多的幫助，讓我論文可以順利完成。

雖然我在研究生室只待了一年，感謝學長和同學們的幫助和鼓勵讓我的研究生生活十分繽紛。感謝這段時間陪伴在我身邊的所有人，給了我很多的幫助和鼓勵，還有感謝我的家人總是默默付出，讓我可以完成這篇論文。

DOI:10.6814/NCCU202100699

中文摘要

該研究的目的是對股票的資料進行分類，以判斷在一段時間內的資料為函數行為或隨機噪音。為了訓練該模型什麼是函數行為和什麼是隨機噪音，我們用三種數學模型對股票資料進行了模擬，並利用訊號處理的技巧從真實股票資料中找出建立數學模型所需要的參數。我們使用支持向量機（SVM）和具有長期短期記憶（LSTM）的深度學習模型進行分類。我們的結果表明，由我們的模擬數據訓練的模型使用在實際數據的預測結果，在顯著水準 $\alpha = 0.05$ 下，我們的分類在統計上有顯著差異。

關鍵字：預測模型、類神經網路、長短期記憶模型、機器學習、支持向量機、總體經驗模態分解

DOI:10.6814/NCCU202100699

Abstract

The purpose of the study was to classify the stock price as functional behavior or random noise in a fixed period. We simulated the data with three kinds of mathematics models to train the model what is functional behavior or random noise. The parameter of mathematics models calculated by the technique of signal processing, such as EEMD. We use the support vector machine(SVM) and the deep learning model with long short-term memory(LSTM) to classification. Our results showed that our model trained by our simulated data used prediction results based on actual data, which are statistically significantly different at the significance level $\alpha = 0.05$ for our classification.

Keywords: Forecasting model, Artificial Neural Network, Long-short term memory, Machine learning, Support vector machine, EEMD

DOI:10.6814/NCCU202100699

Contents

DOI:10.6814/NCCU202100699

DOI:10.6814/NCCU202100699

List of Tables

DOI:10.6814/NCCU202100699

List of Figures

DOI:10.6814/NCCU202100699

Chapter 1

Introduction

In this drastically changing financial market, it is necessary to grasp the rapidly changing trends. Therefore, many participants in the financial market want to find ways to predict the financial market. Machine learning and deep learning approaches achieve unprecedented performance on a broad range of problems. A support vector machine (SVM) is a machine learning model that uses classification algorithms in binary classification. Additionally, the SVM regard as a convex optimization problem, which can find the best hyperplane to classify the data. Sequential deep learning models such as Recurrence Neural Network (RNN) and Long Short Term Memory (LSTM) have proven very powerful for time series data. They have a good performance in the time series classification. In particular, we focus on how to predict the random components in stock price.

According to the Holt and Winter procedure, which describes that the time series is composed of three aspects: level, trend, and seasonality [4], we simulate the data to train the model that can classify random components on the data. Using three types of mathematical models, namely the level, trend, and seasonality, simulate the stock price data. Moreover, we assume that the stock price is a type of signal, then the techniques of signal processing such as EEMD and HHT to decomposition the information of stock and get the seasonal of stock's price.

In this thesis, we build a suitable model to classify the stock's price as function type or random noise in a fixed period. We use SVM and several deep learning models based on LSTM structure. To ensure the model structure, we use grid search and k-folds cross-validation to tune the hyperparameter of the model. To evaluate our model performance, we use our trained model to judge real-world data and use the statistical hypothesis test to prove the classification

DOI:10.6814/NCCU202100699

is efficient on statistics. If we know stock's price is a function type, then we have lots of mathematics model can help us to invest. In contrast, we do not do any operation in this period.

DOI:10.6814/NCCU202100699

Chapter 2

Support Vector Machine

Artificial intelligence(AI) is an approach to make a computer think like a human. The ultimate goal is that the machine is capable of exhibiting intelligence that surpasses the brightest humans. Machine learning(ML) is a subset of AI that uses mathematical analysis and algorithms to solve the problem. It not only process data but also uses data for learning and makes analysis results more accurate. More generally, learning techniques are data-driven methods combining statistics, probability, and optimization. When the model finishes training, it can help us solve the problem. We will discuss Support Vector Machine (SVM), which is the most efficient classification algorithm in the forthcoming sections.

Consider an input space X that is a subspace of $\mathbb{R}^N, N \geq 1$, and the output or target space $\mathscr{Y}$, and let $f : X \to \mathscr{Y}$ be the target function. The training dataset $S = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\} \in (X \times \mathscr{Y})^m$. The learning problem is referred to as a binary classification problem, so $\mathscr{Y} = \{-1, +1\}$. We want to find the hyperplane that separates the training sample into two categories of positively (i.e. $y_i = 1$) and negatively labeled points (i.e. $y_i = -1$). Any hyperplane can be written as

$$\mathbf{w} \cdot \mathbf{x} + b = 0 \tag{2.0.1}$$

where $\mathbf{w} = (w_1, \cdots, w_m) \in \mathbb{R}^m$ is normal vector to the hyperplane, and $\mathbf{x} = (x_1, \cdots, x_m) \in \mathbb{R}^m$ is a vector of data points.

Define the geometric margin $\rho_h(x)$ of a classifier $h : \mathbf{x} \to \mathbf{w} \cdot \mathbf{x} + b$ at a point x is its

DOI:10.6814/NCCU202100699

Euclidean distance to the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$;

$$\rho_h(x) = \frac{|\mathbf{w} \cdot \mathbf{x} + b|}{\|\mathbf{w}\|_2} \tag{2.0.2}$$

The geometric margin ρ_h of a linear classifier h for a sample S is the minimum geometric margin over the points in the sample, $\rho_h = \min_{i=1,\dots,m} \rho_h(x_i)$, that is the distance between the closest sample points and the hyperplane of linear classifier h. The SVM solution is to find the maximum margin hyperplane, which is the safest choice.

2.1 Hard margin

In this case, the training set is linearly separable if all the points can be **perfectly** separates into two groups. The requirement can be stated

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 0, \forall i$$

We derive the equations and optimization problem that define the SVM solution

$$\begin{aligned} & \underset{w,b}{\text{maximize}} \quad \rho_h \\ & \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq \rho_h,\ i = 1, \dots, m. \end{aligned} \tag{2.1.1}$$

Observe that the equations is invariant to multiplication of $(\mathbf{w}, b)$ by a positive scalar. Thus, we can restrict ourselves to pair $(\mathbf{w}, b)$ scaled such that $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1, \forall i$. Furthermore, the optimization problem becomes:

$$\begin{aligned} & \underset{\mathbf{w},b}{\text{maximize}} \quad \frac{1}{\|\mathbf{w}\|} \\ & \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1,\ i = 1, \dots, m. \end{aligned} \tag{2.1.2}$$

Since maximizing $\frac{1}{\|\mathbf{w}\|}$ is equivalent to minimizing $\frac{1}{2}\|\mathbf{w}\|^2$, the pair $(\mathbf{w}, b)$ returned by SVM in the hard margin case is a solution of the following convex optimization problem:

$$\begin{aligned} & \underset{\mathbf{w},b}{\text{minimize}} \quad \frac{1}{2}\|\mathbf{w}\|^2 \\ & \text{subject to} \quad y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1,\ i = 1, \dots, m. \end{aligned} \tag{2.1.3}$$

DOI:10.6814/NCCU202100699

Note that the optimization problem of above promise the solution is exist and unique , because it is convex programming. It is an important property that does not hold for all other learning algorithms.

2.2 Soft margin

In real world, the training data is usually not linearly separable and we relax some constraints, which implies that for any hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$, for some $x_i \in S$:

$$y_i[\mathbf{w} \cdot \mathbf{x}_i + b] \ngeq 1 \tag{2.2.1}$$

Therefore, the constraints in above need to modify. A relaxed version of these constrained is that for some $\xi_i \geq 0, \forall i = 1, \cdots, m$ such that

$$y_i \left[\mathbf{w} \cdot \mathbf{x}_i + b\right] \geq 1 - \xi_i \tag{2.2.2}$$

The variable ξ_i are known as slack variable and are commonly used in optimization problem to define relaxed versions of constraints. If a vector $\mathbf{x}_i$ with $\xi_i \gneq 0$, then $\mathbf{x}_i$ can view as an outlier. If we omit outliers, the training data is linearly separable.

Hence, we can defind the general optimization problem of SVM in the soft margin case where the parameter $C \geq 0$ determined the trade-off between margin-maximization and the minimization of the slack variables $\sum_{i=0}^{m} \xi_i^p$. We also said that C is a scalar regularization hyperparameter.

The soft-margin SVM problem is written as:

$$\begin{gathered} \min_{\mathbf{w}, b, \boldsymbol{\xi}} \frac{1}{2} \|\mathbf{w}\|^2 + C \sum_{i=1}^{m} \xi_i^p \\ \text{subject to } y_i \left(\mathbf{w} \cdot \mathbf{x}_i + b\right) \geq 1 - \xi_i \wedge \xi_i \geq 0, \forall i, \end{gathered} \tag{2.2.3}$$

where $\boldsymbol{\xi} = (\xi_1, \ldots, \xi_m)^T$. The parameter C is called hyper-parameter, that is defined by user not by algorithm, typically determined via n-folds cross validation. This problem is also convex, so it have same good property as hard margin case.

2.3 The Dual Optimization Problem

For simplicity, we discuss hard margin case. We introduce Lagrange variable $\alpha_i \geq 0, i = 1, \ldots, m$, associated to the m constraints. Denote $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^T$. Therefore, our problem can be defined as the following:

$$L(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i [y_i(\mathbf{w} \cdot \boldsymbol{x}_i + b) - 1] \tag{2.3.1}$$

where $\mathbf{w} \in \mathbb{R}^n, b \in \mathbb{R}$, and $\boldsymbol{\alpha} \in \mathbb{R}_+^m$

According to KKT condition, we setting the gradient of the Lagrangian and writting the complementarity conditions:

$$\nabla_{\mathbf{w}} L = \mathbf{w} - \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i = 0 \Rightarrow \mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{2.3.2}$$

$$\nabla_b L = -\sum_{i=1}^{m} \alpha_i y_i = 0 \Rightarrow \sum_{i=1}^{m} \alpha_i y_i = 0 \tag{2.3.3}$$

$$\forall i, \alpha_i [y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1] = 0 \Rightarrow \alpha_i = 0 \vee y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1 \tag{2.3.4}$$

By equation 2.3.2, the weight vector **w** is a linear combination of the training set. By the complementarity conditions 2.3.4, if $\alpha_i \neq 0$, then $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1$ and the corresponding x_i is called support vector. Thus, support vectors lie on the marginal hyperplanes $\mathbf{w} \cdot \mathbf{x}_i + b = \pm 1$

To derive the dual form of the 2.1.3, we plug 2.3.2 and 2.3.4 into 2.3.1. This yield:

$$\begin{gathered} \max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j \left(\mathbf{x}_i \cdot \mathbf{x}_j\right) \\ \text{subject to: } \alpha_i \geq 0 \wedge \sum_{i=1}^{m} \alpha_i y_i = 0, \forall i \in [m] \end{gathered} \tag{2.3.5}$$

In general, the following inequlity always holds:

$$d^* \leq p^*$$

where p^* is the the optimal value of the primal problem and q^* is the optimal value of the dual problem.

However, our problem is satisfies strong duallity property i.e. $d^* = p^*$. Hence, the

DOI:10.6814/NCCU202100699

solution α of the dual problem 2.3.5 can be used directly determined the decision fumction, using equation 2.3.2:

$$h(\mathbf{x}) = sgn(\mathbf{w} \cdot \mathbf{x} + b) = sgn(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot x) + b)$$

2.4 Kernel Method

We extend the SVMs algorithm with **kernel method** to solve the nonlinear problem. In 2.3.5, we also can be write

$$\begin{aligned} &\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} \alpha_i \alpha_j y_i y_j K(\mathbf{x}_i, \mathbf{x}_j) \\ &\text{subject to: } \alpha_i \geq 0 \wedge \sum_{i=1}^{m} \alpha_i y_i = 0, \forall i \in [m] \end{aligned} \tag{2.4.1}$$

,where $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i \cdot \mathbf{x}_j = \langle \mathbf{x}_i, \mathbf{x}_j \rangle$ is the kernel function. In [21], a function $K : X \times X \to \mathbb{R}$ can be defined as below:

$$K(x, x') = \langle \Phi(x), \Phi(x') \rangle, \forall x, x' \in X, \tag{2.4.2}$$

for some mapping $\Phi : X \to \mathbb{H}$ where $\mathbb{H}$ is Hilbert space and it called a feature space. In general, an inner product is commonly using for measuring the similarity between two vectors, because any inner product induces a metric by its norm. K is often a similarity measure between elements of the input space X. The following table is a common kernel in the SVM model [17] [21]:

Name	Formula
Linear kernel	$K(x, y) = \langle x, y \rangle$
Radial basis function (RBF) kernel	$K(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$
Polynomial kernel	$K(x, y) = (x \cdot y + 1)^h$
Laplace RBF kernel	$K(x, y) = \exp\left(-\frac{\|x-y\|}{\sigma}\right)$
Sigmoid kernel	$K(x, y) = \tanh(\alpha x^T y + c)$

Table 2.1: Common kernel in SVM model

An advantage is that we map the input space X to a high-dimensional space. So, we can

DOI:10.6814/NCCU202100699

find a linear classifier/hyperplane that can be perfectly separate in the high-dimensional space $\mathbb{H}$. The different kernel is to take data as transform to different feature space, and at the same time, we may find the linear hyperplane to classify data correctly.

DOI:10.6814/NCCU202100699

Chapter 3

Deep Learning and Neural Networks

As was mentioned in the previous chapter, machine learning(ML) has occupied an important position in computer science. Deep learning is a famous branch of machine learning (ML) methods inspired by the biological neural networks [6]. The Deep learning architecture commonly uses in the real world such as fully-connected neural networks (FCNN), recurrent neural networks(RNN), and convolution neural networks. In facial expression recognition, stock price prediction, and board game programs, deep learning has outstanding performance than other ML models [5] [15]. However, this technology was proposed by David Rumelhart, Geoffrey Hinton, and Ronald J. Williams in the 1980s with back-propagating technology [18], but due to the limitation of computing resources and gradient vanish problem. In 2006, Hinton proposed the Deep Boltzmann Machine (DBM) model to train the multi-layer neural networks successfully [9]. In addition, Nvidia's GPU technology innovation has made deep learning technology regain attention in the world. For example, in 2012, a convolution neural network (CNN) called AlexNet to win first place in the world-class competition of visual, that is ImageNet Large Scale Visual Recognition Challenge(ILSVRC) 2012 recognition [15]. Since then, deep convolutional neural networks (CNNs) have been used in competition. Finally, in 2015, the new structure, ResNet, can have a very deep network of up to 152 layers. This model's top-5 score, namely you check if the target label is one of your top 5 predictions, is beyond human [8]. We introduce the relevant knowledge about the deep learning model below:

DOI:10.6814/NCCU202100699

3.1 Neuron and Neural Networks

Assume the $X = \{x_1, x_2, \cdots, x_N\}$ is a feature space (or input space), $x_i, \forall i$ is called a feature, the target space(or output space) $\mathscr{Y} = \{y_1, \cdots, y_N\}$, and the element in $\mathscr{Y}$ is called a label. For example, let $N = 3$, as shown in Figure 3.1, we give three random independent weights w_1, w_2, w_3 to x_1, x_2, x_3. respectively, which would be adjusted while training in model. For robustness, we add a bias b in the neuron, so we can write in mathematics: $w_1x_1 + w_2x_2 + w_3x_3 + b$. Clearly, it is a linear transformation for x_1, x_2, x_3, but in real world, the problem also is a nonlinear problem. To deal this problem, we use a activation function ϕ to make the output h i.e.

$$h = \phi(\sum_{i=1}^{3} x_i w_i + b)$$

The neural network common structure is comprised of the input layer, hidden layer, and output layer. A picture of the example showed in Figure 3.2. We focus on how to use input data through lots of neural in the hidden layer to get the ‘good’ output (which will define in later). Building the suitable hidden layer is a big topic in computer science, we need to decide how many neurons and layers are suitable, and the activation function. Finally, we use the loss function to evaluate the model performance. We will explain all the detailed steps in the following chapter.

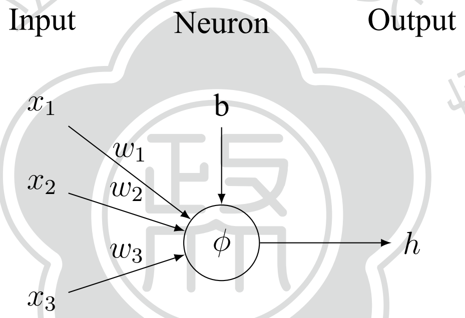


Figure 3.1: A neuron in neural networks

3.2 Activation Function

Activation function takes an important role in neural networks. If we do not use the activation function, the network is only doing lots of linear transformations, and input and output are still linear. In the real world, most situations are not able to predict linearly. The main terminology for nonlinear functions is derivative and monotonic functions. Except for

DOI:10.6814/NCCU202100699

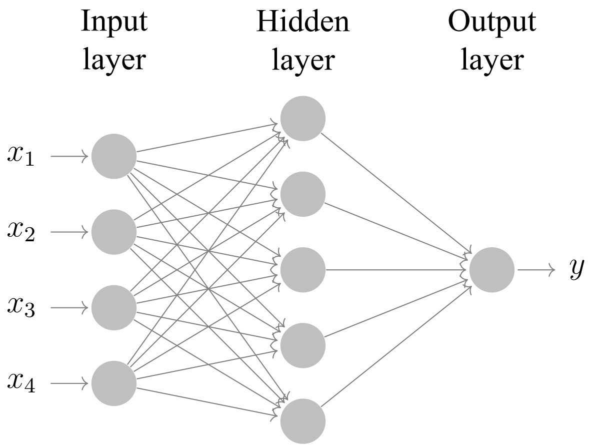


Figure 3.2: The structure of neural networks

the output layer, we select the same activation function for every neuron in hidden layers. The different choices of activation function may cause different results. Here, we show some common activation functions:

1. Sigmoid(or Logistic)

   Equation:

$$\text{Range: } \mathbf{R} \to (0,1), \phi(x) = \frac{1}{1+e^{-x}}$$

   Graph:

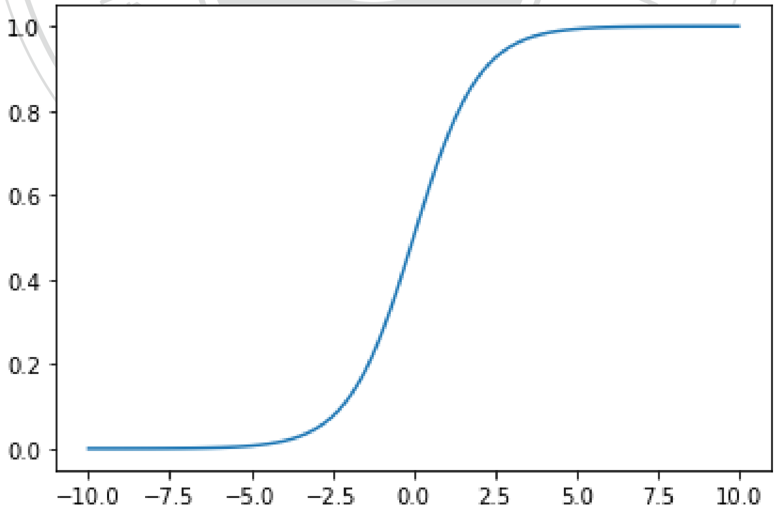


Figure 3.3: Sigmoid function

DOI:10.6814/NCCU202100699

2. Hyperbolic tangent(tanh)

$$\text{Range: } \mathbf{R} \to (-1, 1), \tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} = 2\phi(2x) - 1$$

Graph:

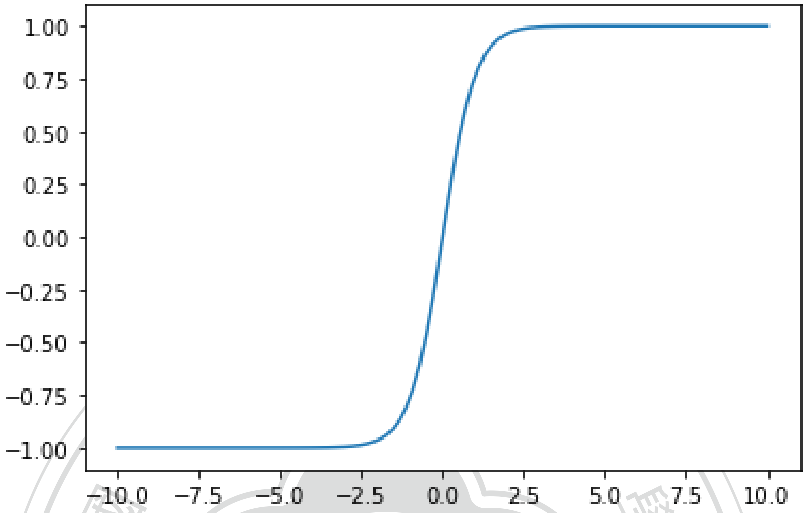


Figure 3.4: Hyperbolic tangent function

3. Rectified Linear Unit(ReLu) [1]

$$\text{Range: } \mathbf{R} \to (0, \infty), relu(x) = \max(0, x)$$

Graph:

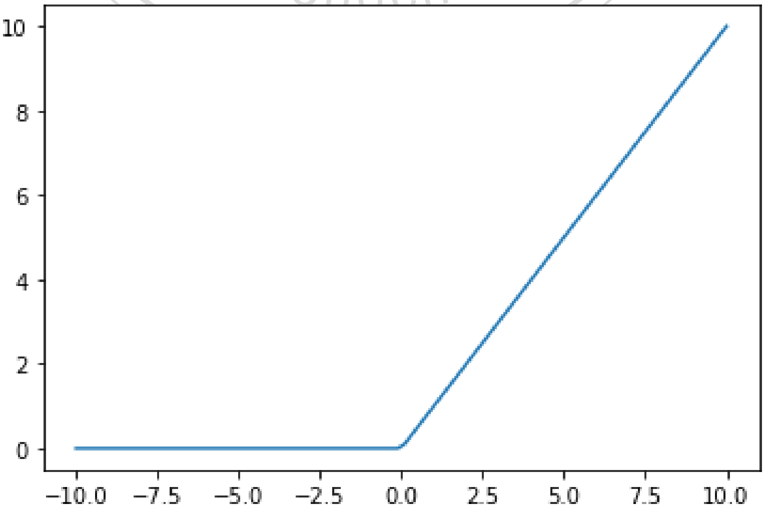


Figure 3.5: ReLu function

DOI:10.6814/NCCU202100699

4. Parametric Rectified Linear Unit(pReLu) [7]

$$\text{Range: } \mathbf{R} \to \mathbf{R}, pReLu(x) = \begin{cases} x, & \text{if } x > 0 \\ ax, & \text{if } x \leq 0 \end{cases}$$

Graph:

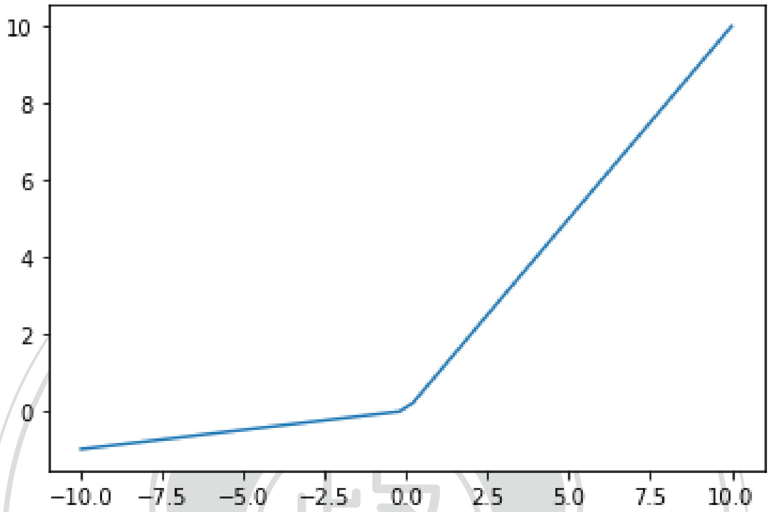


Figure 3.6: pReLu function

3.3 Loss Function

When we train the model, the problem that arises naturally how we can know that the model finishes training. Assume that the set of weight denoted by $\theta^* = \{w_i, \forall i\}$ Our final goal is that find the best set of weight, denoted by θ^* such that the predicted target and real target very 'close' .

Define L is a loss function: $L : \mathscr{Y} \times \mathscr{Y}' \to \mathbb{R}_+$, where $\mathscr{Y}$ is the set of true label and $\mathscr{Y}'$ is the set of predicted label. The function L measures the difference between a true label and a predicted label or evaluates how our algorithms simulate our dataset. Denoted by $f(x_i; \theta)$ means the neural network predicted label with the weights θ and the data x_i. Hence, our training goal is to find the $\theta = \theta^*$ such that $L(\theta^*)$ is the minimum number overall sets. A common example of loss function given below (Suppose we have N data):

DOI:10.6814/NCCU202100699

1. Mean Square Error (MSE)

   Equation:

$$\text{MSE}(\theta) = \frac{1}{N}\sum_{i=1}^{N} \|y_i - f(x_i;\theta)\|^2$$

2. Mean absolute Error (MAE)

   Equation:

$$\text{MAE}(\theta) = \frac{1}{N}\sum_{i=1}^{N} |y_i - f(x_i;\theta)|$$

3. Binary Cross Entropy (BCE)

   Equation:

$$BCE(\theta) = \frac{1}{N}\sum_{i=1}^{N} [y_i(\log(f(x_i;\theta))) + (1-y_i)(1-\log(f(x_i;\theta)))]$$

   where each y_i is binary

4. Categorical Cross Entropy (CCE)

   Equation:

$$CCE(\theta) = \frac{1}{N}\sum_{j=1}^{c}\sum_{i=1}^{N} y_{ji}(\log(f(x_{ji};\theta)))$$

   where each c is the total categories of label and each $y_j i$ is integer, generally.

5. Kullback-Liebler Divergence

   Equation:

$$D_{KL}(P||Q) = -\sum_{i=1}^{N} P(x)\cdot\log\frac{Q(x)}{P(x)} = \sum_{i=1}^{N} P(x)\cdot\log\frac{P(x)}{Q(x)}$$

   where P and Q are probability distribution

3.4 Gradient Descent and Back-propagation

3.4.1 Gradient Descent

For neural networks, we cannot directly calculate the best parameter θ, since the problem is too many variables and complex. In general, the learning problem can regard as an optimization problem. Therefore, we use the gradient descent optimization algorithm to find the local minimum, and weights are updated using the back-propagation of the error algorithm to train a neural network. The idea of gradient descent is that we can find the (local) minimum value of the plane along the opposite direction of the gradient, calculated by a current point. There are three gradient descent algorithms, and the difference between them is how much data we use to calculate the gradient of the objective function. They have a trade-off between the accuracy of the parameter update and the processing time. In the neural networks model, the objective function denoted by $J(\theta)$, θ is a parameter of the model. Let η be the learning rate that determines the size of the steps. The different η helps us to get the better minimum.

First, Batch Gradient descent algorithm is an iterative algorithm, we use full training datasets to calculate the gradient at each step and update the parameter. i.e.

$$\theta^{(t+1)} = \theta^{(t)} - \nabla_\theta J(\theta^{(t)})$$

For this method, if objective function $J(\theta)$ is a non-convex function, this method guarantees to converge to the local minimum. Furthermore, the objective function is convex, it can find the global minimum. However, the convergence rate is very slow in big datasets.

Secondly, Stochastic Gradient Descent (SGD) algorithm:

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta J(\theta; x_i; y_i)$$

Different from to batch gradient descent algorithm, SGD achieves a parameter update at each training example. However, the loss function obtained in each iteration is not in the global optimal direction. For a lot of interactions, it towards the global optimal solution. In particular, the final result is often near the global optimal solution [2] [20]. The benefit is that computation cost is lower than batch gradient descent and convergence rate is better than batch gradient descent. It is worth noting that although this method converges is not the global minimum or local minimum, it is enough in many cases. In fact, for some conditions, the SGD can almost converge in local minimum or global minimum. [3]

Finally, the mini-batch gradient descent algorithms is written as

$$\theta^{(t+1)} = \theta^{(t)} - \eta \cdot \nabla_\theta J(\theta; x_{(i:i+n);y_{(i:i+n)}})$$

This algorithm combines the advantages of the above two methods(Batch normalization and SGD), it updates the parameter for every mini-batch of training example. In the real world, we always use mini-batch gradient descent algorithm to solve the problems. The batch sizes between 50 and 512 are commonly used.

3.4.2 Back-propagation

Now, we can use a gradient descent algorithm to find the minimum objective function. However, calculating the gradient efficiency is a big topic in artificial neural networks. For example, We construct the neural network model (see Figure 3.7)

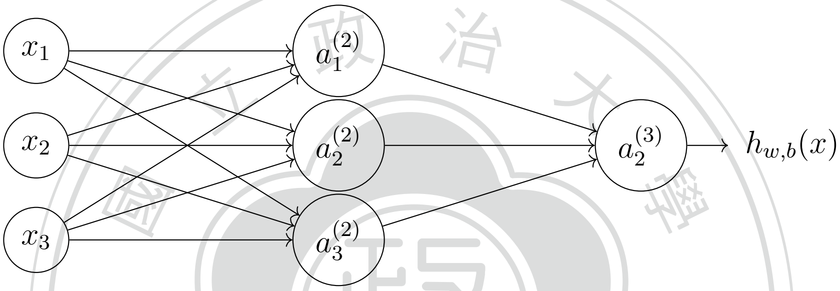


Figure 3.7: The structure of neural networks

We know:

$$
\begin{aligned}
a_1^{(2)} &= \phi\left(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}\right) \\
a_2^{(2)} &= \phi\left(w_{21}^{(1)}x_1 + w_{22}^{(1)}x_2 + w_{23}^{(1)}x_3 + b_2^{(1)}\right) \\
a_3^{(2)} &= \phi\left(w_{31}^{(1)}x_1 + w_{32}^{(1)}x_2 + w_{33}^{(1)}x_3 + b_3^{(1)}\right) \\
h_{W,b}(x) = a_2^{(3)} &= \phi\left(w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + b_1^{(2)}\right)
\end{aligned}
$$

where w_ij is weight between two adjacent layers of neurons and ϕ is an any activation function. For simplicity, our objective function is $J(\theta) = (w_{11}, \cdots, w_{33})$ Now, we use gradient descent algorithm to optimize. We need to find $\nabla J = \frac{\partial J}{\partial w_{11}}\mathbf{e}_{11} + \cdots + \frac{\partial J}{\partial w_{mn}}\mathbf{e}_{mn}$ where e_{ij} is standard orthonormal basis. In our example, $J = (w_{11}, w_{12}, w_{13})$ and $\nabla J = \frac{\partial J}{\partial w_{11}}\mathbf{e}_{11} + \frac{\partial J}{\partial w_{12}}\mathbf{e}_{12} + \frac{\partial J}{\partial w_{13}}\mathbf{e}_{13}$. Calculate directly:

DOI:10.6814/NCCU202100699

$$
\begin{aligned}
\frac{\partial J}{\partial w_{11}} &= \frac{\partial a_2^{(3)}}{\partial w_{11}} \\
&= \frac{\partial \phi\left(w_{11}^{(2)}a_1^{(2)} + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + b_1^{(2)}\right)}{\partial w_{11}} \\
&= \frac{\partial \phi\left(w_{11}^{(2)}\phi\left(w_{11}^{(1)}x_1 + w_{12}^{(1)}x_2 + w_{13}^{(1)}x_3 + b_1^{(1)}\right) + w_{12}^{(2)}a_2^{(2)} + w_{13}^{(2)}a_3^{(2)} + b_1^{(2)}\right)}{\partial w_{11}}
\end{aligned}
$$

However, it is to difficult to calculate. We can use the skill of mathematical, called chain rule i.e. Consider $z = f(u, v)$, where $u = g(t)$, and $v = h(t)$, g and h is differentiate then

$$
\frac{dz}{dt} = \frac{\partial z}{\partial u}\frac{du}{dt} + \frac{\partial z}{\partial v}\frac{dv}{dt}
$$

Using this skill, we can calculate gradient by

$$
\begin{aligned}
\frac{\partial J}{\partial w_{11}} &= \frac{\partial a_2^{(3)}}{\partial w_{11}} \\
&= \frac{\partial a_2^{(3)}}{\partial x_1}\frac{\partial x_1}{\partial w_{11}} + \frac{\partial a_2^{(3)}}{\partial x_2}\frac{\partial x_2}{\partial w_{11}} + \frac{\partial a_2^{(3)}}{\partial x_3}\frac{\partial x_3}{\partial w_{11}}
\end{aligned}
$$

By performing back-propagation, we can get the gradients of the loss function concerning all the inputs and weights of the networks. The benefit is that reduces the computational cost. This algorithm is a powerful and massive speedup of billion times.

3.5 Overfitting, Dropout and Batch Normalization

3.5.1 Overfitting

Whenever we discuss model prediction, it is important to understand prediction errors(bias and variance). It is the best way to understand the problem of overfitting is to express it in terms of bias and variance.

DOI:10.6814/NCCU202100699

Assume that $X = \{X_1, \cdots, X_N\}$ is the data. The labels $Y = \{y_1, \cdots, y_N\}$ are labels are corresponding with X. Given f is some fixed but unknown function for X, then we can write:

$$Y = f(X) + \epsilon$$

where ϵ is random error term with $E(\epsilon) = 0, Var(\epsilon) = \sigma$. Clearly, we can get where So, the expected square error at a point x is The $error(x)$ can be further decomposed as [14]

$$\begin{aligned} Error(x) &= (E[\hat{f}(x)] - f(x))^2 + E\left[(\hat{f}(x) - E[\hat{f}(x)])^2\right] + \sigma^2 && (3.5.1) \\ &= \text{ Bias }^2 + \text{ Variance } + \text{ Irreducible Error} && (3.5.2) \end{aligned}$$

Irreducible error is the error that cannot be improved by a better model. The error that occurs when trying to approximate the behavior of training data, called bias. If you train on a different training, then the variance captures how much your classifier changes set. In Equation 3.5.2, it is a trade-off between bias and variance. The overfitting happens when a lot of noise is captured and fitted in the model. It happens when we train our model much over a noisy dataset, these models have low bias and high variance. In other words, overfitting is means the model is too complex and has a large number of parameters. For example, on Cartesian coordinate system , we have 10 points $(1, 1), (2, 2), \cdots, (10, 10)$. We use the Lagrange interpolation to fit points with a polynomial of order 9. However, we know these points are on $y = x$. The Lagrange interpolation is trying too hard to fit these points and ends up missing the structure of the dataset.

3.5.2 Dropout

To reduce the overfitting, we develop lots of techniques such as regularization and dropout. Regularization methods like Lasso and Ridge reduce overfitting of the objective function. Since we add penalty term in the loss function and the additional term controls the function such the coefficient does not take extreme values. On the other hand, Dropouts modify the network itself. Formally, we express the dropout neural networks model [22].

Suppose a neural network with N hidden layers. Let $z_k^{(l)}$ denote the k-th neuron of inputs at layer l, and $g^{(l)}$ be the vector of outputs at layer l. $W_k^{(l)}$ and $b_k^{(l)}$ denote the k-th neuron of the weights and biases at layer l.

DOI:10.6814/NCCU202100699

We assume the standard neural networks as follows:

$$
\begin{aligned}
z_k^{(l+1)} &= \mathbf{w}_k^{(l+1)}\mathbf{g}^{(l)} + b_k^{(l+1)} \\
g_i^{(l)} &= \phi\left(z_k^{(l)}\right)
\end{aligned}
$$

where ϕ is any activation function With dropout,

$$
\begin{aligned}
r_j^{(l)} &\sim \text{Bernoulli}(p) \\
\widetilde{\mathbf{g}}^{(l)} &= \mathbf{r}^{(l)} * \mathbf{g}^{(l)} \\
z_k^{(l+1)} &= \mathbf{w}_k^{(l+1)}\widetilde{\mathbf{g}}^{l} + b_k^{(l+1)} \\
y_k^{(l+1)} &= \phi\left(z_k^{(l+1)}\right)
\end{aligned}
$$

The dropout is that the probability of each neuron being dropout is p during training in each iteration. When we drop out different neurons, see Figure 3.8, it is equivalent to training a new different neural network. Hence, the dropout procedure is like ensemble learning. The different neural networks will overfit different parts. We train multiple different neural networks and average their weight to reduce the effect of overfitting.

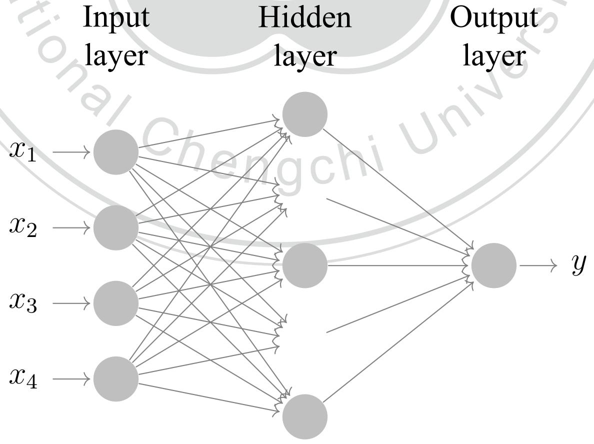


Figure 3.8: The structure of neural networks with dropout

3.5.3 Batch Normalization

We know that normalizing the input data can speed up learning, we consider is that doing the same thing for hidden layers should also get similar effect. Suppose training data and testing data are independent and identically distributed, then the performance of neural networks can be promised. If not, it means the change in the distribution of the input variables present in the training and the test data, called covariate shift. Hence, we use batch normalization to solve the problems [13]. In batch normalization, when we use a stochastic gradient algorithm (SGD) with mini-batch to do forward propagation, we calculate the mean and variance to every batch, and the resulting normalized the mini-batch have mean zero and unit variance.

Suppose values of x over a mini batch $B = \{x_1, \cdots, x_m\}$

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - mu_B)^2$$

$$\hat{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}}$$

$$z_i = \gamma\hat{x}_i + \beta \equiv BN_{\gamma,\beta}(x_i)$$

where $\epsilon > 0$ to avoid denominator is zero. Finally, we use γ and β to do scale and shift, respectively, and they avoid all the data is mean zero and unit variance to the activation function.

DOI:10.6814/NCCU202100699

Chapter 4

Recurrent Neural Networks

As mentioned in the last chapter, deep learning is similar to a function. For the traditional neural networks, we usually assume that all inputs (and output) are independent of each other. However, for lots of cases, this assumption is not suitable. For instance, if you want to predict the stock price, then you better know the data of previous days. In previous cases, the output depends on the previous states. The Recurrence Neural Network (RNN) is a good choice to handle the problem [19]. An RNN model is named recurrence since they do identical work for every element for the sequence. In other words, it also means RNN structure has a **memory** to remember all information. Hence, the RNN model can make a good prediction about sequential data. For example, Given an input sequence $x = (x_1, \cdots, x_N)$ to predict the output sequence $y = (y_1, \cdots, y_M)$. For example, We use ten days of stock data to predict the next ten days of stock data. Then $x_i, y_i \in \mathbb{R}^+, i = 1, 2, \cdots, 10$. The length of the sequence can be any positive integer and N may not equal M.

4.1 Simple Recurrent Neural Network

In recent years, researchers have developed many different types of RNN to solve some defects of the original RNN model. The simplest recurrent neural network, called simple RNN, is the basic of Long-Short Term Memory (LSTM). Simple-RNN has a feedforward layer and a feedback layer like Back-propagation. But Simple-RNN introduces a cycling mechanism based on time (state). The simple RNN structure is shown as Figure 4.1. The hidden layer h has looped back to itself, allowing information to pass from the current time step to the next time step.

DOI:10.6814/NCCU202100699

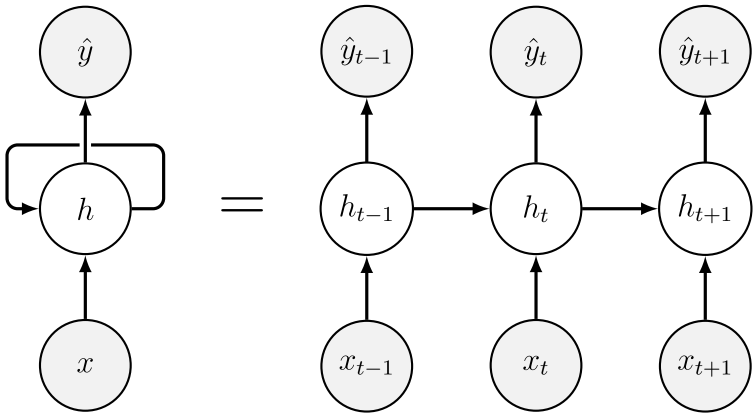


Figure 4.1: Recurrent Neural Network

In mathematics, the following equations define the Simple RNN at time t:

$$y_t = f(h_t;\theta)$$
$$h_t = g(h_{t-1}, x_t;\theta)$$

Variable	Definition
y_t	the output of the Simple RNN at time t
x_t	the input to the Simple RNN at time t
h_t	the state of the hidden layer(s) at time t
θ	the parameters of model (which is the weights and biases for the network).
f, g	an arbitrary activation function such as Sigmoid and ReLu

Table 4.1: The table of variable of RNN structure

The first equation means that given θ, the output at time t depends only on the currently hidden stateh_t, which is the same as a traditional neural network. The second equation means that, given θ,

the current hidden state h_t is depends on the previous hidden states h_{t-1} and the current input x_t.

The equation shows that the Simple RNN can remember the previous information h_{t-1} by past computations to influence the present computations h_t.

The pros and cons of a Simple RNN architecture in the table below:

DOI:10.6814/NCCU202100699

Advantage	Drawbacks
1. Possibility deal with input of any length 2. Computation takes into historical information 3. The dimension of input and output can be different	1. Computing slow 2. Gradient vanishing

Table 4.2: The table of pros and cons

4.2 Long Short Term Memory(LSTM)

As was mentioned in the previous section, the (simple) neural networks can deal with sequential data. However, the model has a big disadvantage: gradient vanish problem, that is for long input-output sequence, RNN has trouble modeling long-term dependencies has less influence on the subsequent decision-making. When the time sequence is more, the influence of the previous information is almost close to zero. The reason is that when we calculate the global gradient that uses the back-propagation algorithm, we need to multiply and summation lots of local gradients with the chain rule. Since each layer and time step are have a gradient, we multiply lots of partial derivatives. When the number of layers of the neural network increases, we need to calculate a lot of gradients. If the values of these gradients are almost less than 1, the gradient calculated by the back-propagation algorithm will become 0. It causes learning to become very slow.

In recent years, the most commonly used structure of RNN is Long Short Term Memory (LSTM). The advantage of LSTM has been able to deal with the gradient vanishing/exploding problem [10].

The important setting to LSTM is the cell state C_t, C_{t-1}, crossing the horizontal line at the top of the Figure 4.2. It extends along the entire chain, with only some minor numerical changes and the information hardly changes. We can remove or add information to the cell state, which structure is called gates. In LSTM, we have three kinds of gate, input gate, forget gate, and output gate. We will show the detail of LSTM in the following.

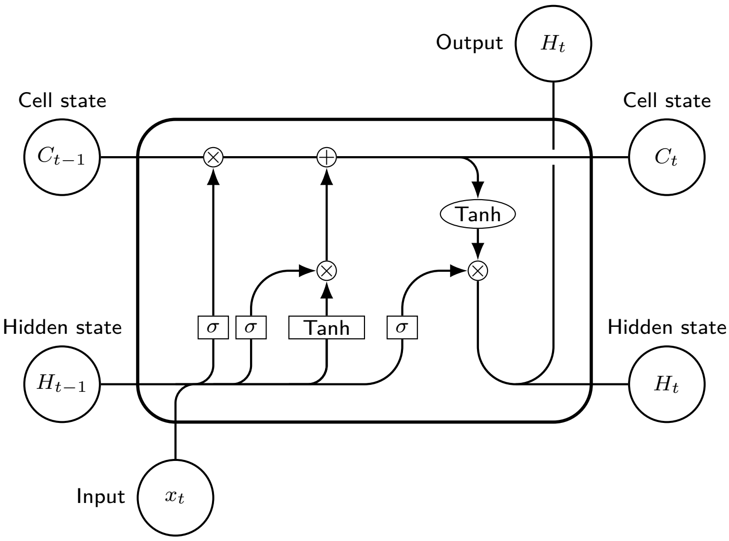


Figure 4.2: The structure of LSTM

Variable	Definition
x_t	Input vector
H_{t-1}	Previous cell output
c_{t-1}	Previous cell state
H_t	Current cell output
c_t	Current cell state
W_f	Weights vector of forget gate
W_c	Weight vector of candidate
W_i	Weight vector of input gate
W_o	Weight vector of output gate
$\sigma(\cdot)$	Sigmoid function

Table 4.3: The Variable in LSTM model

- Forget gate

  In an LSTM model, we use a sigmoid layer, called the forget gate to decide what information we will remember or forget the information from the previous time. We follow the formula:

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

where $\sigma(\cdot)$ is sigmoid function. The value of f_t is between 0 and 1 in the cell state C_{t-1}.

The value of f_t is 1 which represents remember to this while a 0 represents forget to this.

- Input gate

  We decide that new information whether saving in the cell state and divide it into two parts. First, the sigmoid layer, called the input gate, determined which value will update.

$$i_t = \sigma(W_i \cdot [H_{t-1}, x_t] + b_i)$$

  Next, a hyperbolic tangent layer generates a vector of new candidate values, $\tilde{C}_t$, that may update to the cell state. Finally, we combine them to create an update to the cell state.

$$\tilde{c}_t = \tanh(W_C \cdot [H_{t-1}, x_t] + b_c)$$

- Update cell state

  We multiply the old state C_{t-1} by f_t, forgetting the things we determined to forget earlier. Then we add $i_t * \tilde{c}_t$, which is a new candidate value, rescaled by how much we decided to update each state value.

$$C_t = f_t * C_{t-1} + i_t * \tilde{C}_t$$

- Output gate

  The output gate determined the next hidden state and output, and its value based on the previous cell state. First, we put the current input and the previous hidden state into a sigmoid function. It can be represented as:

$$o_t = \sigma(W_o \cdot [H_{t-1}, x_t] + b_0)$$

  Next, we adjust the cell state C_t through hyperbolic tangent function which can scale the value to be between -1 and 1 and multiply it by the output gate value o_t. Hence, the new hidden state is

$$H_t = o_t * \tanh(C_t)$$

Finally, we show that the gradient of the loss function g_k for time step k of the form:

DOI:10.6814/NCCU202100699

$$
\begin{aligned}
\frac{\partial f_k}{\partial W} &= \frac{\partial g_k}{\partial H_k}\frac{\partial H_k}{\partial c_k}\cdots\frac{\partial c_2}{\partial c_1}\frac{\partial c_1}{\partial W} \\
&= \frac{\partial g_k}{\partial H_k}\frac{\partial H_k}{\partial C_k}\left(\prod_{t=2}^{k}\frac{\partial c_t}{\partial c_{t-1}}\right)\frac{\partial c_1}{\partial W}
\end{aligned}
$$

We focus on $\prod_{t=2}^{k}\frac{\partial C_t}{\partial c_{t-1}}$, recall that in LSTM, $c_t = f_t * c_{t-1} + i_t * \tilde{c}_t$. Then

$$
\begin{aligned}
\frac{\partial c_t}{\partial C_{t-1}} &= \frac{\partial}{\partial c_{t-1}}\left[c_{t-1} * f_t + \tilde{c}_t * i_t\right] \\
&= \frac{\partial}{\partial c_{t-1}}\left[c_{t-1} * f_t\right] + \frac{\partial}{\partial c_{t-1}}\left[\tilde{c}_t * i_t\right] \\
&= \frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} + \frac{\partial c_{t-1}}{\partial c_{t-1}} \cdot f_t + \frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t + \frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t \\
&= A_t + B_t + C_t + D_t
\end{aligned}
$$

where A_t, C_t, D_t are the product of activation functions and $B_t = f_t$. (The detail calculation process in Appendix) Hence, using suitable parameter updates of the forget gate, then the gradient contains the forget gate vector of activation function, which makes the network better control the gradients values at each time step. On the other hand, the gradient of cell state consist of function A_t, B_t, C_t ,and D_t. This can better balance gradient values during back-propagation.

DOI:10.6814/NCCU202100699

Chapter 5

Data Simulation

In traditional, the stock's price is treated as the time series data. Many people use the method of time series to predicet the stock's price such as ARIMA model and least squares regression. However, Winter and Holt describe that the time series is composed of three aspects: level, trend, and seasonality [4].

We assume that the stock's price is signal, so we use lots of techniques of signal processing to simulate data and we treat it as a wave. In physics, the wave can be superimposed on the same phase, so it can be used for signal decomposition. Therefore, we assume that the stock price is composed of the functional part and random noise, in particular, the functional part consists of three types of mathematics models, which are seasonal movement model(model 1), exponential model(model 2), and polynomial model(model 3). Model 1 represents seasonality, model 2 represents a trend, and model 3 represents a level.

The training data from parameters obtained from the observed Dow Jones Industrial (DJ) index price from 2019-01-02 to 2020-10-02. The mean value of the DJ index price is 26269.8 and the standard deviation is 1798.336. The functional model is described as:

$$w_1 \cdot \text{Model 1} + w_2 \cdot \text{Model 2} + w_3 \cdot \text{Model 3}$$

, where $w_1 + w_2 + w_3 = 1, w_1, w_2, w_3 \in [0, 1]$.

Then, we mix the functional model and random noise by different weights controlled by α. It is described as:

$$\alpha \cdot \text{Functional model} + (1 - \alpha) \cdot \text{random noise}$$

DOI:10.6814/NCCU202100699

, where $\alpha \in \{0, 0.05, 0.10, \cdots, 0.95, 1\}$. We simulate 10000 replicates, each of which has a length of 30 data points (days). We introduce the individual functional model as follows:

5.1 Seasonal Movement Model

The stock price is a complex time-series data, they have different oscillatory modes and they are non-linear and non-stationary. Hence, we do the signal decomposition to get the characteristic of data. The traditional methods, such as Fourier transformation and discrete wavelet transformation, need to suppose data is linear and stationary. In contrast to traditional methods, namely Hilbert-Huang Transform (HHT) [12], is a famous method that implemented to data that is non-stationary and non-linear, in particular using on the stock price. The main idea is that decompose the signal into lots of components, which complete and nearly orthogonal basis for the original signal, and obtain instantaneous frequency data. These components are called intrinsic mode functions (IMF).

The empirical mode decomposition (EMD) method and Hilbert transformation compose the HHT. The EMD is a procedure to transfer data into a collection of IMFs and the result can be applied to the Hilbert spectral analysis. The main advantage is that we do don assume any assumption on the signals. For example, using discrete wavelet transform, we need to assume the basis for the transformation. An IMF satisfies two properties that mean zero and only one extreme between zero crossings. The method of decomposes the signal into IMFs called the sifting process. The sifting process, that is method of decomposing the signal into IMFs, that describes as below:

1. Define the local maximum and minimum in the test data.

2. To connect all the local maximum, Using cubic spline line as the upper envelope.

3. We repeat the procedure for the local minima to produce the lower envelope.

Now, m_1 refers to the mean of the upper and lower envelopes and data refers to $X(t)$. Define h_1 is written as

$$h_1 := |X(t) - m_1|$$

Next, h_1 is seen as data and m_{11} is the mean of h_1 's upper and lower envelope, in the second

DOI:10.6814/NCCU202100699

	IMF 1	IMF 2	$\cdots$	IMF n
	$X(t)$	$X(t) - c_1 = r_1$	$\cdots$	$r_{n-2} - c_{n-1} = r_{n-1}$
0	$X(t) - m_1 = h_1$	$r_1 - m_2 = h_2$	$\cdots$	$r_{n-1} - m_n = h_n$
1	$h_1 - m_{11} = h_{11}$	$h_2 - m_{21} = h_{21}$	$\cdots$	$h_n - m_{n1} = h_{n1}$
2 $\ldots$	$h_{11} - m_{12} = h_{12}$ $\cdots$	$h_{21} - m_{22} = h_{22}$ $\cdots$	$\cdots$ $\cdots$	$h_{n1} - m_{n2} = h_{n2}$ $\cdots$
k	$h_{1(k-1)} - m_{1k} = h_{1k}$	$h_{2(k-1)} - m_{2k} = h_{2k}$	$\cdots$	$h_{n(k-1)} - m_{nk} = h_{nk}$
IMF	$h_{1k} = c_1$	$h_{2k} = c_2$	$\cdots$	$h_{nk} = c_n$

Table 5.1: The Process of EMD Process

sifting process. Therefore, define h_{11} is written as

$$h_{11} := |h_1 - m_{11}|$$

Repeating the k times until h_{1k} is an IMF, i.e.

$$h_{1(k-1)} - m_{1k} = h_{1k}$$

Hence, h_{1k} is called the first IMF component of the data $c_1 = h_{1k}$. We separate it from the data $X(t) - c_1 = r_1$. The process is repeated for all subsequence r_l and the result is

$$r_{n-1} - c_n = r_n$$

Formally, we represent the EMD process in Table 5.1 (repeat k times).

When the residue, named r_n is a monotonic function from which no IMF can be extracted, the sifting process stops. We can induce that

$$X(t) = \sum_{j=1}^{n} c_j + r_n$$

where r_n is called trend function.

We get k IMF functions. However, they have no meaning. Hence, so we use Hilbert transformation to make it meaningful.

In the EMD process, the mode mixing problem happens commonly. A mode mixing problem in an IMF either comprises signals of widely disparate scales, or the same scale resides in different IMF components. To solve the problem, Wu and Huang present a modified EMD method, called Ensemble EMD (EEMD) [25]. The main idea is that we add white noise of finite amplitude into data before EMD process. Wu and Huang (2005) state ‘White noise is necessary to force the ensemble to exhaust all possible solutions in the sifting process, thus making the different scale signals to collate in the proper intrinsic mode functions (IMF) dictated by the dyadic filter banks’ [25].

The seasonal model is $A_1 \cdot \sin(w_1 t) + A2 \cdot \sin(w_2 t)$, and we apply the EEMD and HHT to estimate the mean period.

Parameter	Values
A_1	1120.882
A_2	856.7009
w_1	0.049
w_2	0.053

Table 5.2: The parameter of Seasonal Movement Model

5.2 Exponential Model

The exponential model represents a slope in long term, written as $B\exp(at)$. Based on theory of stock market and martingale, the data of stock price have heavy autocorrelation. We do the log transformationn before using the ordinary least squares (OLS) to estimate the parameter(See Table 5.3). The log transformation make the increment independent and break its self-correlation to conform to the assumption of regression (residual independence assumption). The exponential model's coefficient B has a half chance of being positive or negative.

	Estimate	Std. error	t-value	$Pr(>\|t\|)$
(Intercept)	1.016e+01	6.685e-03	1520.29	< 2e-16 ***
t	4.885e-05	2.670e-05	1.83	0.0679

Table 5.3: The parameter of Exponential Model

5.3 Polynominal Model

The polynominal model is $X_1 \cdot t^3 + X_2 \cdot t^2 + X_3 \cdot t$. This model represents the higher-order curvature change concept. We use ordinary least squares (OLS) to estimate the coefficients X_1, X_2, X_3 (See Table 5.4). When we simulate data, to increase the data diversity, the polynomial model's coefficient is drawn in a normal distribution with mean of estimate value and a variance of standard error.

	Estimate	Std. error	t-value	$Pr(>\|t\|)$
(Intercept)	2.322e+04	3.098e+02	74.950	< 2e-16 ***
X1	4.565e-04	4.869e-05	9.374	< 2e-16 ***
X2	-3.227e-01	3.244e-02	-9.946	< 2e-16 ***
X3	6.435e+01	6.119e+00	10.517	< 2e-16 ***

Table 5.4: The parameter of Polynominal Model

DOI:10.6814/NCCU202100699

Chapter 6

Experience and Results

In this chapter, we predict the randomness component of stock data with LSTM and SVM models. Our final goal is to predict the rise and fall of stocks at a fixed time. When we know the period is functional, we can use a mathematical model to predict the trend of the sequence. On the contrary, we do not do any operation in this period, when the period is noise.

6.1 Data Transformation

Before we use machine learning and deep learning algorithms, the crucial works is data transformation. Generally, the commonly used techniques are Feature scaling. The main advantage of scaling is to avoid the large number ranges dominating those in smaller number ranges. Another advantage is converged faster when doing gradient descent. [24]. The commonly used techniques of data transformation as follows:

1. Normalization

$$x_{\text{norm}}^{(i)} = \frac{x^{(i)} - x_{\text{min}}}{x_{\text{max}} - x_{\text{min}}}$$

   In general, we scale data to $[0, 1]$ or $[-1, 1]$

2. Z-Score Standardization

$$x_{std}^{(i)} = \frac{x^{(i)} - \mu_x}{\sigma_x}$$

   The mean of the attribute becomes zero and the resultant distribution has a unit standard deviation.

DOI:10.6814/NCCU202100699

Name	LSTM	D/BN	FC	D/BN
A	32	BN	32	BN
B	32	0.25	32	0.25
C	32	BN	16	BN
D	32		16	
E	32			

Table 6.1: Architecture of LSTM model

6.2 Predicted Model

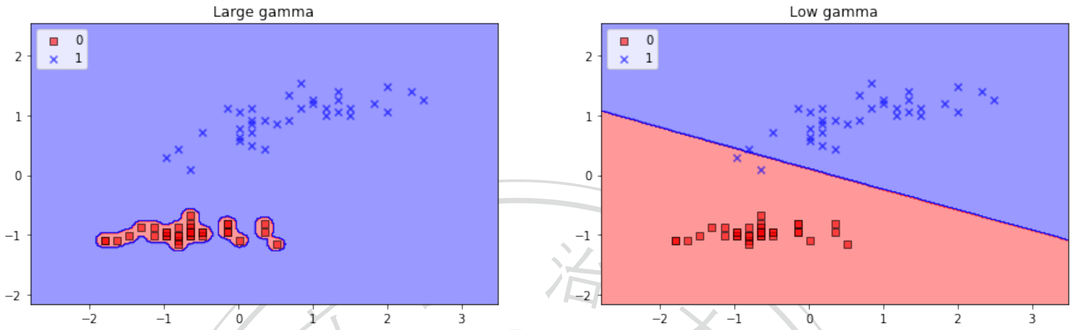
6.2.1 LSTM

In our LSTM model, we design different structure for model(See Table6.1). We add a special layer, batch normalization or dropout layer, between two dense layers to avoid overfitting. Besides, we use an orthogonal matrix as an initial matrix to obtain better performance. The orthogonal matrix has a good mathematical property such that efficiency reduces the gradient vanish and gradient exploding problem [8] [23].

In the Table 6.1, the LSTM column means the units of LSTM. In D/BN columns, the number represents dropout rate and BN means we use batch normalization instead of dropout. In all fully connected layers, we use pRelu as the activation function. The loss function that uses cross-entropy with the optimizer is SGD. We set the hyperparameters of the SGD - learning rate as 0.1, weight decay is 10^{-6}, and momentum 0.9. We set the batch size for 128 and trained 1000 epochs.

6.2.2 SVM

We follow the step in this paper [11]. We use soft-margin SVM model with RBF kernel $K(x, y) = e^{-\gamma\|x-y\|}$ where $\gamma = \frac{1}{2\sigma^2}$, which is a Gaussian function and linear kernel $K(x, y) = \langle x_i, x_j \rangle$. We determine the penalty coefficient C and the coefficient of RBF kernel γ. C is the parameter for the soft margin objective function. Small C ignores the outliers in the training data. The model chooses a large margin separating hyperplane. Since the penalty will cause the smaller-margin hyperplane like the hard-margin case, large C minimizes the number of misclassified samples. The penalty is different for all misclassified examples. It is proportional to the distance from the decision boundary. The gamma parameter is the inverse of the standard deviation of the RBF kernel, i.e. Gaussian function, which measures the similarity between two points. When gamma is very small, i.e. a large variance of Gaussian function, then the similarity radius is large. It means the model can not capture the complexity or ‘shape’ of the data. But, for gamma is large i.e. a low variance of Gaussian function, the points are very close to each other.



(a) Large gamma (b) Low gamma

Figure 6.1: Different efficient on gamma

Using grid-search and k-folds cross-validation to tune the best hyperparameters, C and γ. The parameter of model is shown as Table 6.2. Finally, we ensure the hyperparameters, using whole data sets to train the model.

6.3 Model Performance

We use the simulated data (in chapter 6) to train our model, and we predict test data in one period(we choose 30 days to be one period) is random data or functional data. The model prediction is +1, which means that this sequence is functional. On the contrary, the model prediction is 0, which means that this sequence is random noise. Therefore, the model should evaluate its accuracy. The formula to calculate accuracy described below:

$$\text{accuracy} = \frac{\text{tp} + \text{tn}}{\text{tp} + \text{tn} + \text{fp} + \text{fn}},$$

where

tp = Number of true positive values

tn = Number of true negative values

DOI:10.6814/NCCU202100699

Name	kernel	C	γ
1	rbf	1	10
2	rdf	1	1
3	rbf	1	0.1
4	rbf	10	10
5	rbf	10	1
6	rdf	10	0.1
7	rbf	100	10
8	rdf	100	1
9	rbf	100	0.1
10	linear	1	
11	linear	10	
12	linear	100	

Table 6.2: Architecture of SVM model

fp = Number of false positive values

fn = Number of false negative values

Accuracy is a measure of correctly predicted total observations. We use cross-validation to calculate the average accuracy and standard deviation. The result is provided in the tables below (Table 6.3 and 6.4):

Model	Accuracy	Standard Deviation
Model A	84.49%	6.14%
Model B	82.54%	15.35%
Model C	80.66%	9.91%
Model D	71.29%	21.44%
Model E	71.70%	21.58%

Table 6.3: Results of LSTM for our simulation data

DOI:10.6814/NCCU202100699

Model	Accuracy	Standard Deviation
Model 1	98.2%	0.25%
Model 2	97.7%	0.25%
Model 3	95.8%	0.5%
Model 4	98.7%	0.20%
Model 5	98.5%	0.20%
Model 6	97.4%	0.30%
Model 7	98.6%	0.25%
Model 8	98.5%	0.25%
Model 9	98.3%	0.25%
Model 10	96.2%	0.4%
Model 11	96.8%	0.4%
Model 12	96.8%	0.4%

Table 6.4: Results of SVM for our simulation data

In Table 6.3, the performance of the batch normalization layer is stabler than the dropout layer (see model A and model B). The number of neurons in a fully connected layer does not affect the accuracy and standard deviation(see Model A and Model C). The fully connected layer after the LSTM layer is necessary(see Model A and Model E) because the accuracy has dropped significantly, besides the standard deviation has also become larger. However, the standard deviations of all models are too large. This phenomenon indicates that the model is not stable, and it is easy to encounter problems in practical applications.

In Table 6.4, the performance of the RBF kernel is better than the linear kernel. The C and γ in Model $1 \sim 10$ do not affect the accuracy and standard deviation. In particular, the standard deviation of all models is smaller than 1 percent, which represents the models are stable.

Additionaly, we focus on the relationship between the stock length of time price and accuracy. Hence, we use the different time period to train the SVM model, the result is provided below:

As shown in Figure 6.2, the accuracy decreases with the length of the period of training data. It is also worth noting that we use seven days to train models and predict, and the accuracy is 78%. However, the training days are less than five days, and the accuracy dramatically

DOI:10.6814/NCCU202100699

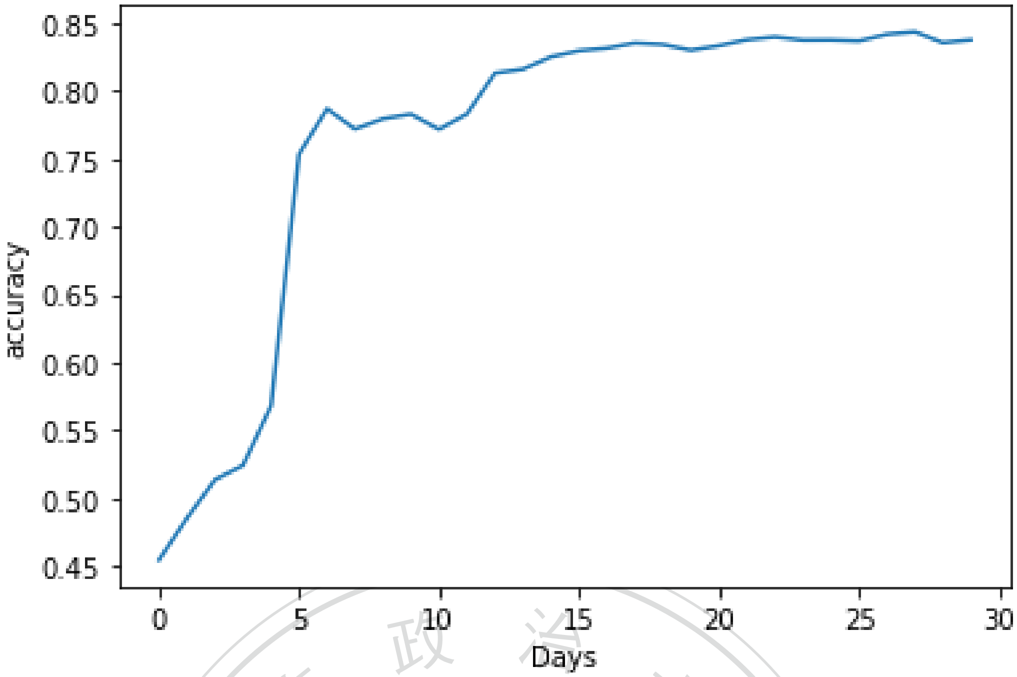


Figure 6.2: Relationship between days and accuracy

decreases. From our perspective, the figure means that we need at least one weak data and find the behavior for the stock market.

6.4 Test on real world data

Finally, we use the closing price of the Dow Jones Industrial Average(DWJ) between 2001.01.02 to 2020.04.30 to be testing data. We split the data every 30 days (non-overlapping). We classify each data as the functional type or noise by our model; to verify whether our classification is reasonable, we use the moving average to eliminate the noise and make the function more apparently [16]. To find the distance of original data and the extracted data from the moving average, we calculate the moving average of each data (windows size = 5) and then calculate the original data and the absolute error of the moving average. Finally, the mean of the absolute error of the function type (μ_1) and the absolute error of the random noise type (μ_2). Doing a paired sample t-test, the statistical hypothesis is

DOI:10.6814/NCCU202100699

$$H_0 : \mu_1 - \mu_2 = 0$$

$$H_1 : \mu_1 - \mu_2 \neq 0$$

The p-value is less than 0.001, so we reject null hypothesis (H_0) at a confidence level of 95%. Therefore, the classification of SVM model is statistic significant.

DOI:10.6814/NCCU202100699

Chapter 7

Conclusion and Discussion

In our assumption, we hope the stock's data is a signal, use the techniques of signal processing to simulate the new dataset. Based on the simulated dataset, we build two models, LSTM and SVM. In LSTM models, the batch normalization layer is a better choice than the dropout layer. The models have high accuracy but not stable (high standard deviation). Though the LSTM can reduce the overfitting and gradient vanish/exploding problem, the LSTM structure is too complex for this problem.

On the other hand, the SVM model has higher accuracy than LSTM, and the standard deviation is much low. Since the SVM algorithms are a convex programming problem, we can find the global minimum implies the model is stable. Hence, we use the SVM model to observe the relationship between the stock length of time price and accuracy.

Furthermore, we use the DWJ data to validate our classification is whether statistic significant. The result shows that our classification model can classify correctly. On the other hand, this result said that the stock's price can regard as a signal. Therefore, we use techniques of data processing is make sense.

Based on our model, we can devise a new strategy for doing stock portfolio management, trading stock, and develop a mathematical model to predict the stock trend when the period is functional. In future work, we can conduct simulated trading on the period, which is a function type, calculate the return on investment, and compare it with the other trading strategies.

DOI:10.6814/NCCU202100699

Appendix A

The derivatives of the $\frac{\partial c_t}{\partial c_{t-1}}$ in 4.2

$$
\begin{aligned}
\frac{\partial f_t}{\partial c_{t-1}} \cdot c_{t-1} &= \frac{\partial}{\partial c_{t-1}} \left[\sigma\left(W_f \cdot [H_{t-1}, x_t]\right)\right] \cdot c_{t-1} \\
&= \sigma'\left(W_f \cdot [h_{t-1}, x_t]\right) \cdot W_f \cdot \frac{\partial H_t}{\partial c_{t-1}} \cdot c_{t-1} \\
&= \sigma'\left(W_f \cdot [H_{t-1}, x_t]\right) \cdot W_f \cdot o_{t-1} \tanh'\left(c_{t-1}\right) \cdot c_{t-1} \\
\frac{\partial i_t}{\partial c_{t-1}} \cdot \tilde{c}_t &= \frac{\partial}{\partial c_{t-1}} \left[\sigma\left(W_i \cdot [H_{t-1}, x_t]\right)\right] \cdot \tilde{c}_t \\
&= \sigma'\left(W_c \cdot [H_{t-1}, x_t]\right) \cdot W_c \cdot \frac{\partial H_t}{\partial c_{t-1}} \cdot \tilde{c}_t \\
&= \sigma'\left(W_c \cdot [H_{t-1}, x_t]\right) \cdot W_c \cdot o_{t-1} \tanh'\left(c_{t-1}\right) \cdot \tilde{c}_t \\
\frac{\partial \tilde{c}_t}{\partial c_{t-1}} \cdot i_t =& \frac{\partial}{\partial c_{t-1}} \left[\sigma\left(W_c \cdot [H_{t-1}, x_t]\right)\right] \cdot i_t = \\
&= \sigma'\left(W_c \cdot [h_{t-1}, x_t]\right) \cdot W_c \cdot \frac{\partial H_t}{\partial c_{t-1}} \cdot i_t = \\
&= \sigma'\left(W_c \cdot [h_{t-1}, x_t]\right) \cdot W_c \cdot o_{t-1} \tanh'\left(c_{t-1}\right) \cdot i_t
\end{aligned}
$$

Then,

$$
\begin{aligned}
\frac{\partial c_t}{\partial c_{t-1}} &= \sigma'\left(W_f \cdot [H_{t-1}, x_t]\right) \cdot W_f \cdot o_{t-1} \tanh'\left(c_{t-1}\right) \cdot c_{t-1} \\
&\quad + f_t \\
&\quad + \sigma'\left(W_i \cdot [H_{t-1}, x_t]\right) \cdot W_i \cdot o_{t-1} \tanh'\left(c_{t-1}\right) \cdot \tilde{c}_t \\
&\quad + \sigma'\left(W_c \cdot [H_{t-1}, x_t]\right) \cdot W_c \cdot o_{t-1} \tanh'\left(c_{t-1}\right) \cdot i_t
\end{aligned}
$$

DOI:10.6814/NCCU202100699

Bibliography

[1] Abien Fred Agarap. Deep learning using rectified linear units (relu). *arXiv preprint arXiv:1803.08375*, 2018.

[2] Léon Bottou. Stochastic gradient descent tricks. In *Neural networks: Tricks of the trade*, pages 421–436. Springer, 2012.

[3] Léon Bottou, Frank E Curtis, and Jorge Nocedal. Optimization methods for large-scale machine learning. *Siam Review*, 60(2):223–311, 2018.

[4] Chris Chatfield and Mohammad Yar. Holt-winters forecasting: some practical issues. *Journal of the Royal Statistical Society: Series D (The Statistician)*, 37(2):129–140, 1988.

[5] J. X. Chen. The evolution of computing: Alphago. *Computing in Science Engineering*, 18(4):4–7, 2016.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.

[7] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification, 2015.

[8] Mikael Henaff, Arthur Szlam, and Yann LeCun. Recurrent orthogonal networks and long-memory tasks. *arXiv preprint arXiv:1602.06662*, 2016.

[9] Geoffrey E Hinton, Simon Osindero, and Yee-Whye Teh. A fast learning algorithm for deep belief nets. *Neural computation*, 18(7):1527–1554, 2006.

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

DOI:10.6814/NCCU202100699

[11] Chih-Wei Hsu, Chih-Chung Chang, Chih-Jen Lin, et al. A practical guide to support vector classification, 2003.

[12] Norden Eh Huang. *Hilbert-Huang transform and its applications*, volume 16. World Scientific, 2014.

[13] Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International conference on machine learning*, pages 448–456. PMLR, 2015.

[14] Gareth James, Daniela Witten, Trevor Hastie, and Robert Tibshirani. *An introduction to statistical learning*, volume 112. Springer, 2013.

[15] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. *Advances in neural information processing systems*, 25:1097–1105, 2012.

[16] Guohui Li, Zhichao Yang, and Hong Yang. Noise reduction method of underwater acoustic signals based on uniform phase empirical mode decomposition, amplitude-aware permutation entropy, and pearson correlation coefficient. *Entropy*, 20(12), 2018.

[17] K-R Muller, Sebastian Mika, Gunnar Ratsch, Koji Tsuda, and Bernhard Scholkopf. An introduction to kernel-based learning algorithms. *IEEE transactions on neural networks*, 12(2):181–201, 2001.

[18] David E Rumelhart, Geoffrey E Hinton, and Ronald J Williams. Learning representations by back-propagating errors. *nature*, 323(6088):533–536, 1986.

[19] Mike Schuster and Kuldip K Paliwal. Bidirectional recurrent neural networks. *IEEE transactions on Signal Processing*, 45(11):2673–2681, 1997.

[20] Ohad Shamir and Tong Zhang. Stochastic gradient descent for non-smooth optimization: Convergence results and optimal averaging schemes. In *International conference on machine learning*, pages 71–79. PMLR, 2013.

[21] Alex J Smola and Bernhard Schölkopf. A tutorial on support vector regression. *Statistics and computing*, 14(3):199–222, 2004.

DOI:10.6814/NCCU202100699

[22] 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.

[23] Eugene Vorontsov, Chiheb Trabelsi, Samuel Kadoury, and Chris Pal. On orthogonality and learning recurrent networks with long term dependencies. In *International Conference on Machine Learning*, pages 3570–3578. PMLR, 2017.

[24] Xing Wan. Influence of feature scaling on convergence of gradient iterative algorithm. In *Journal of Physics: Conference Series*, volume 1213, page 032021. IOP Publishing, 2019.

[25] Zhaohua Wu and Norden E Huang. Ensemble empirical mode decomposition: a noise-assisted data analysis method. *Advances in adaptive data analysis*, 1(01):1–41, 2009.

DOI:10.6814/NCCU202100699