國立政治大學資訊管理學研究所

碩士學位論文

優化資料清理與機器學習的機制

The refined mechanism for data cleaning and machine learning

指導教授:蔡瑞煌 博士

研究生:余艾玨 撰

中華民國一〇七年七月

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

謝辭

碩士班的這兩年裡，很感謝得到許多老師和同學的指導與幫助，讓我的畢業論文可以順利的完成，過程中雖然艱辛，但是能夠有如此大的收穫，真的是非常值得。

首先要感謝我的論文指導教授蔡瑞煌老師，他不斷地協助我，引導我來完成論文，從應用的領域到實驗設計，除了教我專業知識，還常常花時間和我討論如何改善演算法，讓程式可以順利的執行，因為他辛苦和耐心的指導，我才能順利的完成實驗和論文。也很感謝我的家人，爸爸和媽媽在我成長的過程中，不斷地在背後支持我、鼓勵我，有你們的支持，我才可以專心讀完碩士學位和完成論文。

畢業論文展現了我在碩士班兩年裡的學習過程和成果，如何將所學加以應用並解決問題是最重要的部分，遇到困難的時候非常感謝有老師的指導、家人的支持、還有同學們的協助，論文雖然還有需要改進的地方，但也是我努力後的成果。

余艾玨 敬筆

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

優化資料清理與機器學習的機制

摘要

近年來人工智慧在機器學習的應用扮演重要的角色，而相較於大數據分析的統計方法，ANN 成為最有用方法中的其中一個，為了處理動態環境中的時間序列資料和離群值，Wu (2017)提出一個資料清理和機器學習的機制，實驗結果顯示提出的機制在資料清理和機器學習方面是很有效的，Wu (2017)已經透過單一隱藏層倒傳遞神經網路實作 RLEM，這個研究將使用兩個方法優化此機制，一個是在 RLEM 的損失函數(loss function)加上正規化項來避免過度擬合(overfitting)的問題，另一個是修改 RLEM 並透過新版的 Tensorflow 實作來達成目標。

關鍵字: 人工神經網路、正規化、單一隱藏層倒傳遞神經網路、RLEM

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

The refined mechanism for data cleaning and machine learning

Abstract

In recent years, artificial intelligence (AI) has become an important part in the application of machine learning, and the artificial neural networks (ANN) serves as one of the most useful methods compared to statistical methods for the purpose of big data analytics. To cope with the time series data that may have concept-drifting phenomenon and outliers, Wu (2017) had derived a mechanism for effective data cleaning and machine learning. The experiment results had shown that the proposed mechanism is promising in effective data cleaning and machine learning. Wu (2017) had implemented the resistant learning with envelope module (RLEM) via the adaptive single-hidden layer feed-forward neural networks (SLFN). This research will add the regularization term to loss function to prevent overfitting and will refine RLEM to improve the accuracy of the predicted return of carry trade. The refined mechanism will be implemented via the updated version of Tensorflow.

Keyword: artificial neural networks, regularization, single-hidden layer feed-forward neural networks, resistant learning with envelope module

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Index

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Figure Index

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Table Index

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Chapter 1 Introduction

1.1 Background

In recent years, artificial intelligence (AI) has become an important part in the application of machine learning that interacts with or imitates humans in a convincingly intelligent way. Instead of building a traditional program comprised of logical statements and decision trees, a neural network is built specifically for training and learning. The network is based on a parallel network of neurons, each set up for a specific purpose (Android Authority, 2018). It is often used in the field of healthcare, business, education, finance, law and manufacturing.

ANN serves as one of the most useful methods compared to traditional statistical methods for virtual classification especially in financial predictions. It is a highly accurate model and yet limited by many assumptions. The process of ANN is also very hard to be realized that makes it difficult to be applied despite the advantages. It will be more convenient for the users to interpret the results if the process can be more easily understood (Monthly Archives, 2016).

Additionally, ANN offers a number of advantages, including requiring less formal statistical training, ability to implicitly detect complex nonlinear relationships between dependent and independent variables, ability to detect all possible interactions between predictor variables, and the availability of multiple training algorithms. Disadvantages include its black box nature, greater computational burden, proneness to overfitting, and the empirical nature of model development (Tu, 1886). Among these advantages, overfitting is a problem that many people encountered. Training a network may be done in many different ways that all involve a brute force iterative approach. This approach is to maximize output accuracy and train the optimum paths through the networks (Android Authority, 2018).

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Wu (2017) derive a mechanism to cope with the time series data that may have concept-drifting phenomenon and outliers. The proposed mechanism is implemented via TensorFlow and GPU for effective data cleaning and machine learning. The ANN mechanism corresponds to the U.K. against the U.S. from January 1990 to December 2016 (324 months) with the data set of nominal exchange rates, LIBOR, CPI, money supply, and GDP for the U.S., and any of the other counterparty countries. Due to the random initial weights, the total amount of adopted hidden nodes and training time needed depending on each ε value (ε = 0.75, 0.5, 0.25, 0.1). There are two stages in the experiment. In the first stage, the resistant learning algorithm is used with the envelope module to learn the training block and then remove the detected outlier candidates. In the second stage, the SLFN obtained in stage one is applied to each data in the testing block. As a result, the total number of adopted hidden nodes and training time spent increase as the ε value decreases. The smaller ε value is, the greater effectiveness will be. In addition, the movement of excess returns of carry trade can be well predicted with this mechanism.

1.2 Motivation

As ANN becomes more and more popular, a problem in machine learning rises as how to make an algorithm perform well not only on the training data but also on new inputs. Many strategies are designed to reduce the test error, probably at the expense of increased training error (Goodfellow et al., 2016). Rather than a black box problem in the deep learning process, it is more important to know what drives the performance and get good results in a more systematic way. Therefore, developing more effective strategies has become one of the significant goals in this field.

Compared to CPU, graphic processing unit (GPU) has a massively parallel architecture consisting of thousands of smaller and more efficient cores designed for handling multiple tasks simultaneously (NVIDIA, 2018). NVIDIA has released lots of

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

libraries that implement common computational primitives. These primitives are highly optimized for GPUs needed in deep learning. As Tensorflow is able to compute all the gradients, it is more convenient that we do not have to write backward pass. The other advantage about Tensorflow is that we can switch all this computation between CPU and GPU (Li et al., 2017).

Our goal is to refine the learning process of a model and get higher accuracy rate. Considering that GPU is more efficient and yet more expensive, we decided to improve mechanism with different aspects. Among the ANN algorithms, the most common method is backpropagation, which will update the variables until it achieves the training goal. Even though optimization has been proposed and shown its advantages on some models, we are not sure it will suit our needs in training process. Therefore, we adopt several strategies such as regularization to resolve the situation of overfitting while training complicated ANN models, refined backpropagation and refined cramming mechanism to improve the RLEM that mentioned above.

1.3 Objective

To achieve the goal, this research refines the mechanism for data cleaning and machine learning in the concept drifting environment (Wu, 2017) to increase the accurate of carry trade prediction with different strategies. In this research, we will add the regularization term to the loss function to prevent overfittting and will refine the backpropagation and cramming mechanism. As the mechanism contains a great number of matrix multiplications, the refined mechanism will be implemented via the updated version of Tensorflow.

This is an experimental research that combines AI field and financial field using refined mechanism to improve the results of predicting the return of carry trade. I am looking forward to contributing both AI and financial fields in the future.

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Chapter 2 Literature Review

This chapter is a brief introduction containing five parts of literature review. The first section is the regularization that avoids the problem of overfitting. The second section is about how to combine backpropagation and gradient descent to adjust the parameters in the neural network. The third section is GPU and the Tensorflow, which is one of the major deep learning frameworks. The fourth section is the mechanism of coping with outliers in the concept drifting environment. And the last section is the mechanism for data cleaning and machine learning.

2.1 Regularization

Overfitting is a common issue in machine learning, which occurs when we build a model that not only captures the signal but also the noise in a dataset. In order to avoid overfitting, we can create models that generalize and perform well on different data points to correct overfitting with regularization. It is the concept of adding an additional term to the loss function. In addition to the data loss, it should fit the training data. Regularization is just anything that you do to model, rather than explicitly trying to fit the training data, that sort of penalizes somehow the complexity of the model (Li et al., 2017).

"Regularization is any modification we make to a learning algorithm that is intended to reduce its generalization error but not training error" (Goodfellow et al., 2016, page 221). Regularization is a very important technique in machine learning to prevent overfitting. There are lots of regularization strategies. For instance, extra constraints are put on a machine learning model, such as adding restrictions on the parameter values. Extra terms can also be put in the objective function that may be thought of as corresponding to a soft constraint on the parameter values. The performance can be improved by these extra constraints and penalties on the test set if we choose correctly. Regularization of an estimator works by trading increased bias for reduced variance. We can realize that an

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

effective regularizer makes a profitable trade, reducing variance significantly while the bias is not overly increasing (Goodfellow et al, 2016).

One of the most common regularization methods is L2 regularization (also called Ridge), that only the weights are regularized n different layers differently strong instead of biases. The L2 regularization is the sum of the square of the weights and its computational efficient due to having analytical solutions and it forced the parameters to be relatively small, the bigger the penalization, the smaller the coefficients will be (Enhance Data Science, 2017). As follows:

L2 regularization adds a penalty equal to the sum of the squared value of the coefficients:

$$R(\mathrm{w}) = \sum_i |w_i{}^2| \tag{1}$$

2.2 Gradient descent optimization algorithms

In order to minimize or maximize a loss function, we need optimization algorithm that is simply a mathematical function dependent on the Model's internal learnable parameters which are used in computing the target values from the set of predictors used in the model. The weight and bias values in the neural networks are internal learnable parameters which are used in computing the output values and are learned and updated in the direction of optimal solution such as minimize the loss function and also play an important part in the training process of the neural network model (Towards Data Science, 2017).

There are two major types of optimization algorithm including first order optimization algorithms and second order optimization algorithms. The former algorithms are used to minimize or maximize a loss function using its gradient values with respect to the parameters like gradient descent. The first order derivative tells us whether the function is increasing or decreasing at a particular point and gives us a line which is tangential to a point on its error surface. The latter is used to minimize or maximize the loss function and tells us whether the first derivative is increasing or decreasing which hints at the function's

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

curvature. It provides us with a quadratic surface which touches the curvature of the error surface. So we can choose different optimization strategy according to these characteristics, the first order optimization techniques are easy to compute and less time consuming especially on large data sets. Unless the second order derivative is known, the second order techniques are always slower and costly to compute in terms of both memory and time (Towards Data Science, 2017).

Backpropagation

To optimize the neural network, backpropagation is a technique to calculate derivatives quickly. It is a way that we can apply an activation function to those sums of products after we propagate forward calculating the dot product of Inputs signals and their corresponding weights. And then we use gradient descent to propagate backwards in the network carrying error terms and update weight values (Towards Data Science, 2017). The backpropagation learning algorithm can be shown as:

The learning is an optimization problem that minimizes $E(Z)$.

Step 0.1: Generate and store the initial values of $\boldsymbol{Z}$.

Step 0.2: Input all training data {($\mathbf{X}_1$, $\mathbf{d}_1$), …, ($\mathbf{X}_N$, $\mathbf{d}_N$)}, with $\mathbf{d}_c$ being the desired response vector corresponding to the c^{th} stimulus vector $\mathbf{X}_c$.

Step 1: Execute the forward operation of SLFN regarding all training data: calculate $h(\boldsymbol{X}_c, \boldsymbol{w}_i^H) \forall\ i\ \forall$ c and store them, then calculate $y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W^H}) \forall\ l\ \forall$ c and store them.

Step 2: Based upon $y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W^H})$ and d_{cl} values, calculate the $E(Z)$ value and store it.

Step 3: If $E(Z)$ is less than the predetermined value (says, ε_1), then STOP.

Step 4: Based upon values of $h(\boldsymbol{X}_c, \boldsymbol{w}_i^H) \forall\ i\ \forall$ c and $y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W^H}) \forall\ l\ \forall$ c, execute the following backward operations:

Step 4.1: calculate the values of $\frac{\partial E(Z)}{\partial w_{l0}^O} \equiv \sum_{c=1}^N (y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W^H}) - d_{cl})\ \ \forall\ l$

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

and store them.

Step 4.2: calculate the values of

$\frac{\partial E(Z)}{\partial w_{li}^{O}} \equiv \sum_{c=1}^{N}(y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W^H}) - d_{cl})h(\boldsymbol{B}_c, \boldsymbol{X}_i)\ \ \forall\, l\, \forall\, i$ and store them.

Step 4.3: calculate the values of

$\frac{\partial E(Z)}{\partial w_{i0}^{H}} \equiv \sum_{c=1}^{N}\sum_{l=1}^{q}(y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W^H}) - d_{cl})(1-(h(\boldsymbol{X}_c, \boldsymbol{w}_i^H))^2)w_{li}^{O}\ \ \forall\ i$ and store them.

Step 4.4: calculate the values of

$\frac{\partial E(Z)}{\partial w_{ij}^{H}} \equiv \sum_{c=1}^{N}\sum_{l=1}^{q}(y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W^H}) - d_{cl})(1-(h(\boldsymbol{X}_c, \boldsymbol{w}_i^H))^2)w_{li}^{O}\, x_{cj}\ \ \forall\, i$

$\forall\, j$ and store them.

We are given some function $\boldsymbol{f}(\boldsymbol{x})$ where $\boldsymbol{x}$ is a vector of inputs and we are computing the gradient of $\boldsymbol{f}$ at $\boldsymbol{x}$ (i.e.$\boldsymbol{\nabla f}(\boldsymbol{x})$). The backward-flowing gradient can be interpreted on an intuitive level, for instance, the most common used in neural networks that back propagation performs during the backward pass are add gate and multiply gate. The add gate takes the gradient on its output and distributes it equally to all of its inputs, regardless of what their values were during the forward pass. While the multiply gate takes the input activations, swaps them and multiplies by its gradient (Li et al., 2017).

Many concepts extend in a straight-forward manner to matrix and vector operations which are the matrix-matrix multiplication multiply operations. In addition, the gradient with respect to a variable should have the same shape as the variable (Li et al., 2017). The matrix multiplication on the left composed of a bunch of rows matrix multiplies that on the right by another matrix composed of a bunch of columns matrix. This produces a final matrix where each element in the output matrix is a dot product between one of the rows and one of the columns of the two input matrices. These dot products are all independent, it can be split up completely and have each of those different elements of the output matrix

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

all being computed in parallel. Besides, they all sort of are running the same computation which is taking a dot product of these two vectors (Li et al., 2017).

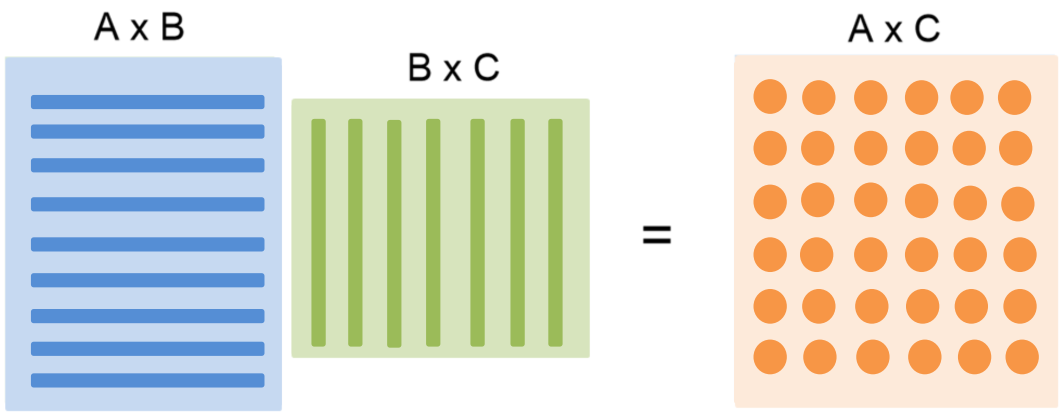


Figure 1: Matrix Multiplication
(Source: Li et al., 2017)

2.3 GPU and Tensorflow

The graphic card also called Graphics Processing Unit (GPU) is really developed. Most common talking about GPUs in deep learning are NVIDIA GPUs. Both GPUs and CPUs are kind of a general purpose computing machine where they can execute programs and do sort of arbitrary instructions, but they're qualitatively pretty different. The difference between a GPU and a CPU is that consumer GPUs have thousands of cores, each of those cores runs at a much slower clock speed. Instead of operating independently, the GPU cores have to work together and sort of paralyze one task across many cores rather than each core totally doing its own thing. While these days desktop CPU has four or six even up to 10 cores. This means hardware can physically run, like maybe 8 or up to 20 threads concurrently. So the CPU can maybe do 20 things in parallel at once. It is not only a gigantic number, but those threads for a CPU are pretty powerful. Every CPU can actually do a lot of things fast and independently. Another thing to show between CPUs and GPUs is memory. CPUs have some cache, but that's relatively small and the majority of the memory for CPU is pulling from system memory. GPUs also have their own caching

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

system similar to the caching hierarchy in a CPU where there are sort of multiple hierarchies of caching between the 12 gigabytes of GPU memory and the actual GPU cores. The GPUs typically have their own relatively large block of memory within the card itself. To summarize, GPUs are specialized for highly paralyzable algorithms. So the prototypical algorithm of something that works really well and is like perfectly suited to a GPU is matrix multiplication. The CPUs are good for general purpose processing and they can do a lot of different things (Li et al., 2017).

When the data are read from different places in the two input matrices, a GPU can have all of this elements of the output matrix all computed in parallel and that could make this thing compute very fast. This is kind of the prototypical type of problem that a CPU might have to go in and step through sequentially and compute

Each of these elements comes one by one. CPUs these days have multiple cores, so they can do vectorized instructions as well. Nevertheless, GPUs tend to have much better throughput for massively parallel problems especially when these matrices get really big (Li et al., 2017).

Tensorflow is probably a main deep learning framework from Google as a better choice for a lot of research type problems these days. In Tensorflow, we can divide computation into two major stages. The first stage is to define computational graph and then run the graph over and over again and actually feed data into the graph to perform whatever computation wanted to perform. So this is the common pattern while using Tensorflow (Li et al., 2017).

2.4 The Resistant learning with envelope module (RLEM)

Tsaih and Cheng (2009) present a resistant learning procedure single-hidden layer feed-forward neural network (SLFN) to detect outliers. The SLFN fitting function is defined as:

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

$$a_i(x) \equiv \tanh(\, w_{i0}^{H} \;+\; \textstyle\sum_{j=1}^{m} w_{il}^{H} x_i \,), \tag{2}$$

$$f(x) \;\equiv w_0^{O} + \; \textstyle\sum_{i=1}^{p} w_i^{O} \tanh(\, w_{i0}^{H} \;+\; \sum_{j=1}^{m} w_{il}^{H} x_i \,) \tag{3}$$

where $\tanh(x) \equiv \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}}$, m is the number of explanatory variables (x_j) ; $\boldsymbol{x} \equiv (\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_m})^{\mathrm{T}}$; p is the adaptive number of adopted hidden nodes; w_i^{o} is the bias value of the i^{th} hidden node; the superscript H throughout the paper refers to quantities related to the hidden layer; w_{ij}^{H} is the weight between the j^{th} explanatory variable x_j and the i^{th} hidden node; w_0^{o} is the bias value of the output node; the superscript o throughout the paper refers to quantities related to the output layer; and w_i^{o} is the weight between the i^{th} hidden node and the output node. In their study, a character in bold represents a column vector, a matrix, or a set, and the superscript *T* indicates the transposition.

Through this SLFN tool, the coming information x is first transformed into$\equiv \left(a_1, a_2, \dots, a_p\right)^{T}$, and the corresponding value of *f* is generated by a rather than x. Namely, given the observation, all the corresponding values of hidden nodes are first calculated with $a_i \equiv \tanh\left(w_{i0}^{H} \;+ \sum_{j=1}^{m} w_{ij}^{H} x_j\right)$ for all I, and the corresponding value $f(\mathrm{x})$ is then calculated as $f(\mathrm{x}) = \; g(a) \equiv \; w_0^{o} + \; \sum_{i=1}^{p} w_i^{o}\, a_i$.

In the learning stage, a set of *N* training cases $\{\left(\boldsymbol{X}_1, \boldsymbol{d_1}\right), \left(\boldsymbol{X}_2, \boldsymbol{d_2}\right), \;\dots,\; \left(\boldsymbol{X}_N, \boldsymbol{d_N}\right)\}$ is given. Regarding all training cases, the learning goal is to seek a $\boldsymbol{Z}$ where, for all $l\left(output\ nodes\right)$ and $all\ c\ (training\ cases)$,
$|d_{cl}\text{-}y\left(\boldsymbol{X}_c, \boldsymbol{w}_l^{O},\, \boldsymbol{W^{H}}\right)| < \varepsilon \;\; \forall\, c \;\forall\, l,\ where\ \varepsilon\, is\ tiny$. The error function is defined as:

$$E(\boldsymbol{Z}) \equiv \; \tfrac{1}{N} \textstyle\sum_{c=1}^{N} \sum_{l=1}^{q} \left(dcl \; - y\left(\boldsymbol{X}_c, \boldsymbol{w}_l^{O},\, \boldsymbol{W^{H}}\right)\right)^2 \tag{4}$$

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

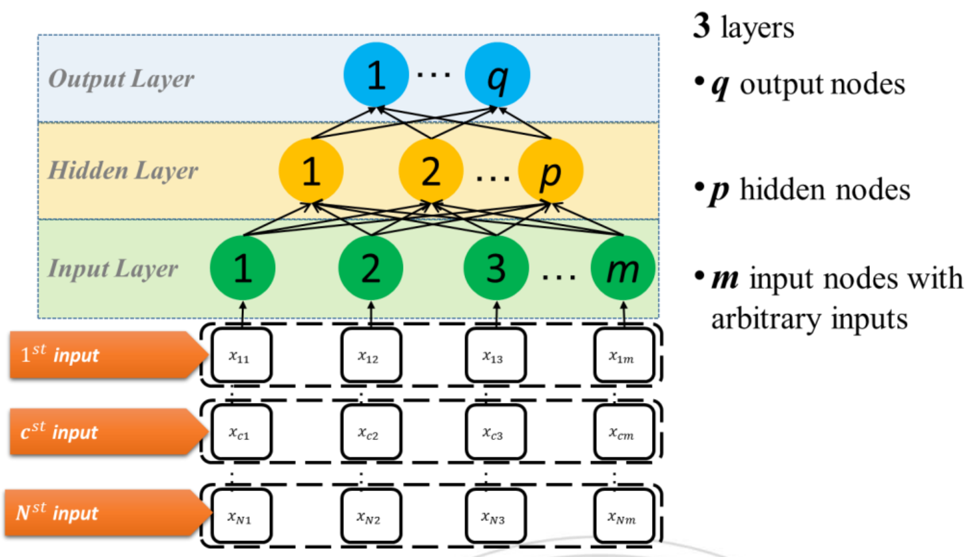
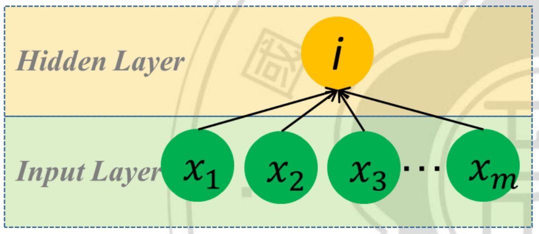


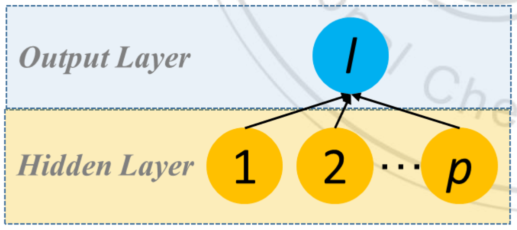
Figure 2: The Network Structure of The Single-hidden Layer Feed-forward Neural Networks (SLFN) with multiple output nodes



The activation value of i^{th}hidden node :

$$h(\boldsymbol{X}_c, \boldsymbol{w}_i^H) \equiv \tanh\left(w_{i0}^H + \sum_{j=1}^{m} w_{ij}^H x_{cj}\right)$$

Figure 3: The Activation Value of Hidden Node



The activation value of l^{th}output node :

$$y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W}^{\boldsymbol{H}}) \equiv w_{l0}^O + \sum_{i=1}^{p} w_{li}^O h(\boldsymbol{X}_c, \boldsymbol{w}_i^H)$$

Figure 4: The Activation Value of Output Node

Tsaih and Cheng (2009) proposed a resistant learning mechanism (RL) with the SLFN and a tiny pre-specified ε value 10^{-6} to deduce a function form. The RL takes the training examples one by one. The RL implements both robustness analysis and deletion diagnostics to exclude potential outliers at the early stage, thus prevent SLFN from learning them. Hereafter, the outlier is the observations far away from the fitting function deduced from a

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

subset of the given observations. The idea of robustness analysis contains features for deriving an (initial) subset of m+1 reference observations to fit the linear regression model, ordering the residuals of all N observations at each stage and then augmenting the reference subset gradually, based upon the smallest trimmed sum of the squared residuals principle. The deletion diagnostics employs with the diagnostic quantity being the number of pruned hidden nodes when one observation is excluded from the reference pool. The RL also dynamically adapts the number of adopted hidden nodes and the associated weights of SLFN during the training process.

To sum up, the RL implements the weight-tuning mechanism, the recruiting mechanism, and the reasoning mechanism to allow the SLFN to evolve dynamically during the learning process and to explore an acceptable nonlinear relationship between explanatory variables and the responses in the presence of outliers.

Huang et al. revised RL into the RLEM to speed up the process of identifying the potential outlier shown in Table 1. The envelope width of the RLEM is 2ε. The ε value 0.05 proposed in RLEM is larger than a tiny ε value (10^{-6}) proposed in RL. The parameter k in Step 2 refers to the percentage of potential outlier, which means at least $N(1\text{-}k)$ reference observations will be wrapped into the envelope. For instance, assume we suspect that there are approximately at least 95 percent non-outliers and at most 5 percent outliers, then we can set k as 5% and the RLEM will result in a fitting function with an envelope that contains 95 percent reference observations into consideration while building the SLFN estimate. A SLFN estimate is acceptable at the n^{th} stage if all of the smallest n squared residuals are less than the pre-specified ε.

Table 1: The resistant learning with envelope module (adapted from Huang et al., 2014). *N* is the total amount of training data, *m* is the amount of input nodes, *k* is the percentage of potential outlier, and the envelope width is 2*ε*.

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

```
Step 1: Use the first m+1 reference observations in the training data set to set up an
        acceptable SLFN estimate with one hidden node. Set n = m+2.
Step 2: If n > N*(1-k), STOP.
Step 3.1: Use the obtained SLFN to calculate the squared residuals regarding all N
          training data.
Step 3.2: Present the n reference observations that are the ones with the smallest n
          squared residuals among the current squared residuals of all N training
          data.
Step 4: If all of the smallest n squared residuals are less than ε (the envelope width),
        then go to Step 7; otherwise, there is one and only one squared residual that
        is larger than ε.
Step 5: Set $\tilde{w} = w$.
Step 6: Apply the gradient descent mechanism to adjust weights w of SLFN. Use the
        obtained SLFN to calculate the squared residuals regarding all training data.
        Then, either one of the following two cases occurs:
        If the envelope of obtained SLFN does contain at least n observations, then
        go to Step 7.
        If the envelope of obtained SLFN does not contain at least n observations,
        then set $w = \tilde{w}$ and apply the augmenting mechanism to add extra hidden
        nodes to obtain an acceptable SLFN estimate.
Step 7: Implement the pruning mechanism to delete all of the potentially irrelevant
        hidden nodes; n + 1 → n; go to Step 2.
```

2.5 The mechanism for data cleaning and machine learning

Carry trade is a trading strategy which borrows at a low interest rate and invests in an asset that provides a higher rate of return. It is based on borrowing in a low interest rate

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

currency and converting the borrowed amount into another currency, with these proceeds either placed on deposit in the second currency if it offers a higher rate of interest, or deployed into assets – such as stocks, commodities, bonds, or real estate that are denominated in the second currency (Investopedia, 2018).

The behavior of exchange rates and the professional is the only unknown in a carry trade has mostly settled into a belief that fundamental explanations of the exchange rate have no predictive value in the short-run. Jordà et al. (2012) knit together basic no-arbitrage conditions in international economics to derive a predictive frame-work that nests commonly used carry trade strategies. According to Jordà et al. (2012) the ex-post nominal excess returns to a carry trade are:

$$y_{t+1} = \Delta e_{t+1} + (i_t^* - i_t) \tag{5}$$

Where e_{t+1} is the log nominal exchange rate in U.S. dollars per foreign currency, and i_t and i_t^* are nominal interest rates home (U.S.) and abroad for a riskless deposit with a one-period maturity. Given that the interest rates in are observed at time *t*, the major task of grasping a positive carry trade return hinges on developing a good forecasting model for e_{t+1}. It is well known that economic fundamentals do not have good predictive power for the exchange rate, especially in the short-term. Nevertheless, some evidence shows that these macroeconomic fundamental might be useful in generating positive return from carry trade. Therefore, the mechanism of (Wu, 2017) proposed is used to forecast the return of carry trade one year later and predicted the exchange rate using macro fundamentals. i.e., they forecast *h* period ahead exchange rates as:

$$e_{t+h} = \alpha_0 + \beta_1(m_t - m_t^*) + \beta_2(g_t - g_t^*) + \beta_3(r_t - r_t^*) + \beta_4(\pi_{t+1} - \pi_{t+1}^*) + \varepsilon_{t+h} \tag{6}$$

where m_t is log money supply, g_t is log GDP, r_t is nominal interest rate, and π_{t+1} is the expected next period rate of inflation from the home country. Mark of * represents the values from foreign country. To achieve the estimation purpose, researchers may use the long-term government bond rate as a proxy for the expected inflation rate. The investor can

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

make use of eq. (5) to determine the trade orientation with the prediction of the exchange rate (Wu, 2017).

For data cleaning and machine learning, this study adapts the works of Tsaih and Cheng (2009) and Huang et al. (2016) to present a mechanism that implements the moving window and RLEM. M is the index of the current window, N is the sample size of the training block, B is the sample size of the testing block, s is the standard deviation of training data in the training block, and ε represents the maximal deviation between actual and predicted outputs (Wu, 2017). The proposed mechanism is shown as Table 2:

Table 2: The proposed ANN mechanism of Wu (2017)

Step 0: Set *M* as 1.

Step 1.1: Apply the RLEM stated in Table 1 (with envelope width = 2σ) to the *N* training examples $\{(\mathbf{x}^{(M-1)B+1}, y^{(M-1)B+1+h}), (\mathbf{x}^{(M-1)B+2}, y^{(M-1)B+2+h}), \ldots, (\mathbf{x}^{(M-1)B+N}, y^{(M-1)B+N+h})\}$ to filter out *Nk* potential outliers and obtain an acceptable SLFN.

Step 1.2: Remove the outlier candidates, and then use the SLFN obtained in Step 1.1 and the RLEM stated in Table 1 (with envelope width = 2ε) again to learn the remained $N(1-k)$ training examples. (start from n = 1)

Step 2: Apply the SLFN obtained in Step 1.2 to the *B* testing examples $\{(\mathbf{x}^{(M-1)B+N+1}, y^{(M-1)B+N+1+h}), (\mathbf{x}^{(M-1)B+N+2}, y^{(M-1)B+N+2+h}), \ldots, (\mathbf{x}^{MB+N}, y^{MB+N+h})\}$.

Step 3: For more data, $M \leftarrow M+1$ and GOTO Step 1.1; otherwise, STOP.

In Step 1.1, we apply the RLEM stated in Table 1 (with envelope width = 2σ) to *N* training examples in the current training block to result in an acceptable SLFN whose envelope contains at least $N(1-k)$ examples. σ is the standard deviation of training examples in the current training block. Furthermore, we also obtain the order information and deviance information regarding all training examples in the current training block. Here,

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

the “order information” is the order of the data sequence learning by the SLFN and the “deviance information” is the distance between actual and predicted outputs of SLFN. The RLEM guarantees that the *N*(1-*k*) data in the training block are wrapped into the envelope. The last *Nk* data are the potential outliers, which may not be wrapped into the envelope. Namely, Step 1.1 adopts both the deviance information and the order information to identify *Nk* examples in the current training block that will be excluded in Step 1.2. In Step 1.2, remove the outlier candidates, and then use the SLFN obtained in Step 1.1 and the RLEM stated in Table 1 (with envelope width = 2ε) again to learn the remained *N*(1-*k*) training examples. ε represents the maximal deviation between actual and predicted outputs of SLFN regarding all *N*(1-*k*) training examples.

In Step 2, we apply the obtained SLFN to the testing examples. Step 3 examines the stopping criteria. If there are more examples, this mechanism slides the window further and goes back to Step 1.1. If there are no more examples, the mechanism stops.

According to the statements of (Huang et al., 2014), Step 1.1 can act as an effective outlier-filter. According to the statements of (Tsaih and Cheng, 2009), Step 1.2 can result in an effective learning of the patterns embedded in the data resulted from Step 1.1 (Wu, 2017).

Based on the aforementioned theories of carry trade, the variables of input datasets for a month interval (end-of-month) include (1) nominal exchange rates, (2) LIBOR (one-month London interbank offered rates), (3) consumer price indices (CPI), (4) money supplies, and (5) GDPs, as shown in Table 3 (Wu, 2017).

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Table 3: The definition of all variables in the input vector (Wu, 2017)

$\mathbf{x}^t \equiv (x_1^t, x_2^t, x_3^t, x_4^t, x_5^t, x_6^t, x_7^t, x_8^t, x_9^t)^T$ at time epoch t

Variable	**Definition**
x_1^t	nominal exchange rates between U.K. and U.S.
x_2^t	12-month LIBOR for the U.S.
x_3^t	CPI for the U.S.
x_4^t	money supplies for the U.S.
x_5^t	GDPs for the U.S.
x_6^t	12-month LIBOR for the U.K.
x_7^t	CPI for the U.K.
x_8^t	money supplies for the U.K.
x_9^t	GDPs for the U.K.

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Chapter 3 Experiment Design

This research refines the mechanism for data cleaning and machine learning in the concept drifting environment based on the one proposed mechanism (Wu, 2017) to predict the return of carry trade. The experiment is designed to optimize the efficiency of resistant learning mechanism and increase the accuracy. This chapter is a brief introduction containing two parts of experiment design. The first section is the daata description according to the U.S. and U.K. data from January 1990 to December 2016. The second section is about the experiment design, and the experimental environment is shown as Table 4:

Table 4: The computer infrastructure used in the experiment

OS	Ubuntu 16.04 LTS
GPU	ASUS ROG STRIX GeForce® GTX 1080-O8G
CPU	Intel® Core^{TM} i7-6900K
RAM	DDR4-2133 64G
Language	Python 3.5
API	Tensorflow-gpu r1.3
CUDA	CUDA 8.0.27

3.1 Data description

In this research, the variables are shown in Table 3 and dataset is from (Wu, 2017). We use the data (102 months) of the U.K. against the U.S. observed at a monthly frequency. In this experiment, we use two moving windows with the training block and the testing instance to illustrate the experiment result. According to the training and testing block in Table3, when M = 1, the first window training block consists of 1st to 100th instances and the testing block is the 101st instance. When M = 2, the training block consists of 2nd to

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

101st instances and the testing block is the 102nd instance. The arrangement of moving window is shown as Figure 5.

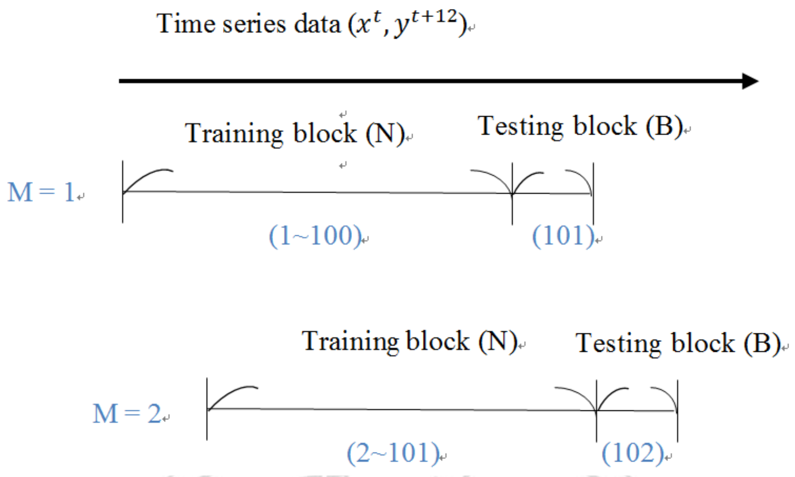


Figure 5: The proposed arrangement of moving window.

3.2 Experiment design

The refined mechanism includes two stages, the first stage will filter out the 5% potential outliers and the second stage will learn the remained data to predict the outputs of the carry trade. In the learning stage, a set of N training cases $\{(\boldsymbol{X}_1, \boldsymbol{d_1}), (\boldsymbol{X}_2, \boldsymbol{d_2}), \ldots, (\boldsymbol{X}_N, \boldsymbol{d_N})\}$ is given. Regarding all training cases, the learning goal is to seek a $\boldsymbol{Z}$ where, for all l (output nodes) and all c (training cases), $|d_{cl}\text{-}y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W}^H)| < \varepsilon \ \forall c \ \forall l$, where ε is 0.75. The training optimization algorithm consists of two terms: the first one called loss term, which measures how well the model fits the data. And the second term we added to penalize the model complexity using the L2 regularization term, which is the sum of the squared weights. λ (also called the regularization rate) is a hyper-parameter regularization strength to represent a scalar value that controls how weights are balanced. The error function is defined as:

$$E(\boldsymbol{Z}) \equiv \frac{1}{N}\sum_{c=1}^{N}\sum_{l=1}^{q}\left(dcl - y(\boldsymbol{X}_c, \boldsymbol{w}_l^O, \boldsymbol{W}^H)\right)^2 + \lambda\sum_{c=1}^{N} w_c{}^2 \quad (7)$$

The learning becomes an optimization problem:

$$\min_{\boldsymbol{Z}} E(\boldsymbol{Z}) \quad (8)$$

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

The flowchart of refined ANN mechanism is shown in Figure 6 which is modified based on the mechanism for data cleaning and machine learning in the concept drifting environment (Wu, 2017). There are two portions of the mechanism we refined including cramming mechanism and weight tuning mechanism, furthermore, the refined mechanism will be implemented via the updated version of Tensorflow. The weight tuning mechanism is shown in Figure 7 to adjust weights and minimize the loss function where ε_1, ε_2 and ε_3 are given tiny numbers, $e^c = dc - y(\boldsymbol{X}_c, \boldsymbol{w}^{\boldsymbol{O}}, \boldsymbol{W}^{\boldsymbol{H}})$.

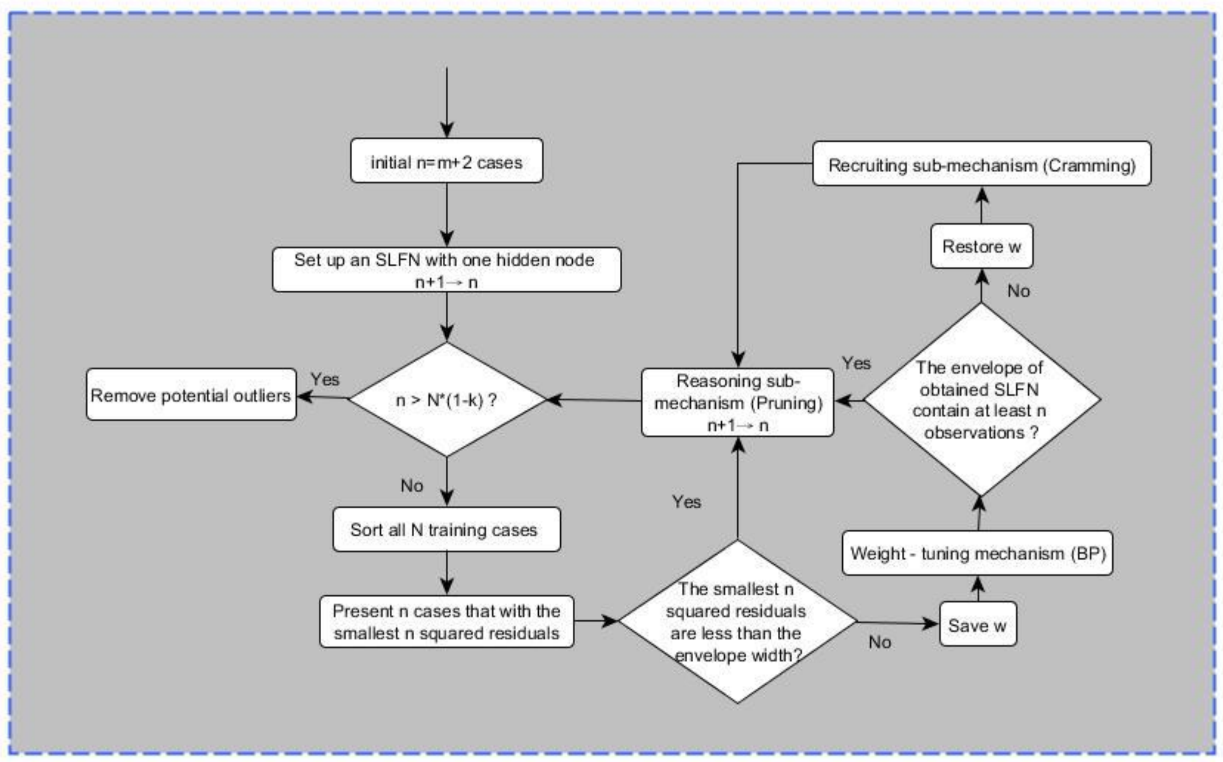

initial n=m+2 cases
Set up an SLFN with one hidden node n+1→ n
n > N*(1-k) ?
Yes
Remove potential outliers
No
Sort all N training cases
Present n cases that with the smallest n squared residuals
The smallest n squared residuals are less than the envelope width?
Yes
No
Save w
Weight - tuning mechanism (BP)
The envelope of obtained SLFN contain at least n observations ?
Yes
No
Restore w
Recruiting sub-mechanism (Cramming)
Reasoning sub-mechanism (Pruning) n+1→ n

Figure 6: The flowchart of proposed algorithm

Table 5 shows the refined resistant learning with envelope module (Adapted from Huang et al., 2014). The $\bar{\lambda}$ in cramming mechanism is calculated in step 6(c) instead of an assigned number. *N* is the total amount of training data, *m* is the amount of input nodes, *k* is the percentage of potential outlier, and the envelope width is 2*ε*. (code reference: https://github.com/rlem). Figure 7 shows the refined flowchart of backpropagation.

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Table5: The refined resistant learning with envelope module (RLEM) for Stage 1 and Stage 2

Stage 1	Step 1: Use the first m+1 reference observations in the training data set to set up an acceptable SLFN estimate with one hidden node. Set $n = m + 2$.
Stage 2	Step 1: $n = 1$
Stage 1 and Stage 2	Step 2: If $n > N(1 - k)$, STOP. Step 3.1: Use the obtained SLFN to calculate the squared residuals regarding all N training data. Step 3.2: Present the n reference observations (x^c, y^c) that are the ones with the smallest n squared residuals among the current squared residuals of all N training data. Let $\mathrm{I}(n)$ be the set of indices of these observations. Step 4: If $(e^c)^2 \leq \varepsilon^2$ $\forall\, c \in \mathrm{I}(n)$, go to Step 7; otherwise, there is one and only one squared residual that is larger than ε^2. Assume $\kappa \in \mathrm{I}(n)$, $(e^\kappa)^2 > \varepsilon^2$ $\forall\, c \in \mathrm{I}(n)$, and $(e^c)^2 \leq \varepsilon^2$ $\forall\, c \in \mathrm{I}(n)$-$\{\kappa\}$ $\forall\, c \in \mathrm{I}(n)$. Step 5: Set $\tilde{\mathrm{w}} = \mathrm{w}$. Step 6: Apply the gradient descent mechanism to adjust weights w of SLFN until either one of the following two cases occurs: (1) If the envelope of obtained SLFN does contain at least n observations, then go to Step 7. (2) If the envelope of obtained SLFN does not contain at least n observations, then apply the following augmenting mechanism to add two extra hidden nodes to obtain an acceptable SLFN estimate:

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

	(a) Set $\mathrm{w} = \widetilde{\mathrm{w}}$ and find an m-vector α of length one such that $\alpha^{\mathrm{T}}(\mathrm{x}^{c} - \mathrm{x}^{\kappa}) \neq 0 \quad \forall\, c \in \mathrm{I}(n)\text{-}\{\kappa\}$. (b) For every $c \in \mathrm{I}(n)\text{-}\{\kappa\}$, compute $\alpha^{\mathrm{T}}(\mathrm{x}^{c} - \mathrm{x}^{\kappa})$. Let $\bar{c}$ be the smallest index c yielding $min_{c\in \mathrm{I}(n)-\{\kappa\}} \lvert\alpha^{\mathrm{T}}(\mathrm{x}^{c} - \mathrm{x}^{\kappa})\rvert$. (c) Set $\bar{\lambda} = \left\lvert\frac{tanh^{-1}\left(\sqrt{\frac{tanh^{2}(\zeta)+1-\frac{\Phi}{tanh(2\zeta)}}{1-tanh^{2}(\zeta)\left(\frac{\Phi}{tanh(2\zeta)}-1\right)}}\right)}{\alpha^{T}(\mathrm{x}^{\bar{c}}-\mathrm{x}^{\kappa})}\right\rvert$, where ζ is a given small number and $\Phi \equiv \frac{2tanh(\zeta)\varepsilon}{(N-1)\lvert y^{\kappa}-w_0^{o}-\sum_{i=1}^{p} w_i^{o} a_i^{\kappa}\rvert}$. Step 6(2)(d): Let p+2 → p and add new hidden nodes p-1 and p to the existing SLFN with $w_{p-1,0}^{H} = \zeta\text{-}\bar{\lambda}\alpha^{\mathrm{T}}\mathrm{x}^{\kappa}$, $w_{p-1}^{H} = \bar{\lambda}\alpha$, $w_{p,0}^{H} = \zeta\text{+}\bar{\lambda}\alpha^{\mathrm{T}}\mathrm{x}^{\kappa}$, $w_{p}^{H} = \text{-}\bar{\lambda}\alpha$, and $w_{p-1}^{o} = w_{p}^{o} = \frac{y^{\kappa}-w_0^{o}-\sum_{i=1}^{p-2} w_i^{o} a_i^{\kappa}}{2tanh(\zeta)}$. Step 7: Implement the pruning mechanism to delete all of the potentially irrelevant hidden nodes; $n + 1 \rightarrow n$; go to Step 2.

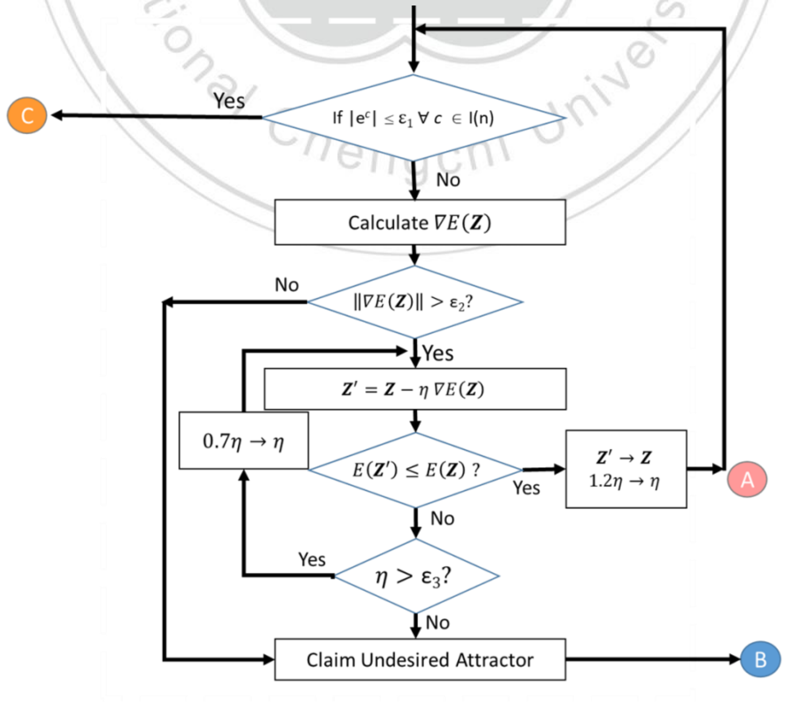

If $\lvert e^c\rvert \le \varepsilon_1 \forall c \in I(n)$
Yes
C
No
Calculate $\nabla E(\mathbf{Z})$
No
$\lVert\nabla E(\mathbf{Z})\rVert > \varepsilon_2$?
Yes
$\mathbf{Z}' = \mathbf{Z} - \eta\,\nabla E(\mathbf{Z})$
$0.7\eta \rightarrow \eta$
$E(\mathbf{Z}') \le E(\mathbf{Z})$?
Yes
$\mathbf{Z}' \rightarrow \mathbf{Z}$
$1.2\eta \rightarrow \eta$
A
No
Yes
$\eta > \varepsilon_3$?
No
Claim Undesired Attractor
B

Figure 7: The weight tuning mechanism (BP)

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Chapter 4 Experiment Result

During the experiment, the training block are {(x^{1}, y^{13}),…,(x^{100}, y^{112})}, {(x^{2}, y^{14}),…,(x^{101}, y^{113}) and the testing instances are {(x^{101}, y^{113}) and (x^{102}, y^{114})}. In the Step 1.1 of Table 2, we get the half envelope width σ, the deviation of 100 training data in the training block is approximately 1.5409 in the first moving window. And in the Step 1.2 of Table 2, we set the half envelope width ε as 0.75. Table 6 indicates the total number of adopted hidden nodes and training time of whole learning process. We get more adopted hidden nodes and it takes more time because of complicated backpropagation.

Table 6: Total number of adopted hidden nodes and training time

		Step 1.1	Step 1.2	Total
The 1st window (σ =1.5409, ε = 0.75)	Total number of adopted hidden nodes	37	89	89
	Time spent (hh:mm:ss)	03:06:11	12:29:17	15:35:28
The 2nd window (σ =1.5214, ε = 0.75)	Total number of adopted hidden nodes	41	92	92
	Time spent (hh:mm:ss)	03:13:07	12:35:29	15:48:36

Figure 8 shows the envelope module of the step 1.1, and No. 31, 32, 34, 35 and 36 instances are identified as potential outliers. The remaining inliers in are all wrapped in the envelope width in both of two steps. Figure 9 shows the actual and predicted return of carry

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

trade in step 1.2 with the testing instances 1.3918. We reduce the error of the prediction of carry trade but we get more adopted hidden nodes.

Figure 8 and Figure 9 show that besides outliers in red-cross notes, inliers in blue dashed line presents the actual return of carry trade. The envelope module in gray dotted line wrapped the inlier instances in the envelope width, the orange solid line represents the predicted return of carry trade.

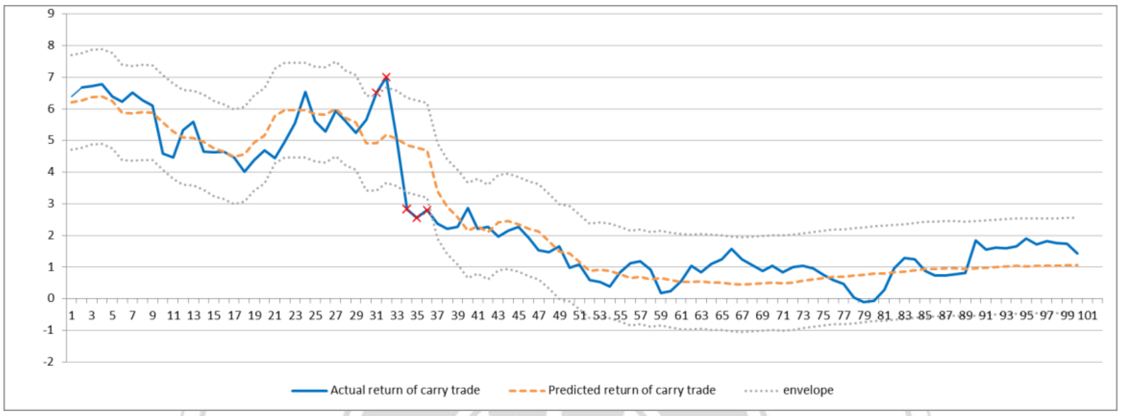


Figure 8: The envelope module of the Step 1.1 to identify the outliers with σ = 1.5409

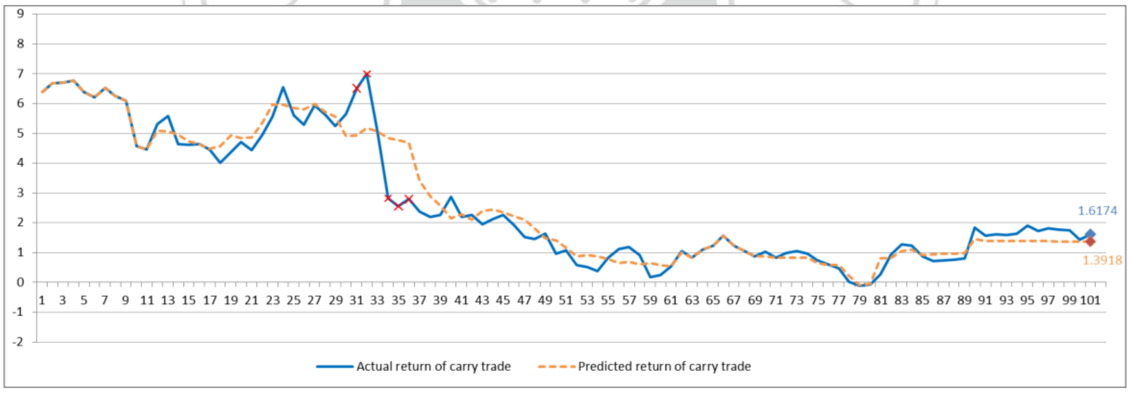


Figure 9: Actual and predicted return of carry trade with ε = 0.75

Figure 10 shows the envelope module of the step 1.1, and No. 30, 31, 33, 34 and 35 instances are identified as potential outliers. The remaining inliers are all wrapped in the envelope width in both of two steps. Figure 11 shows the actual and predicted return of carry trade in step 1.2 with the testing instances 1.5941. Figure 10 and Figure 11 show that

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

besides outliers in red-cross notes, inliers in blue dashed line presents the actual return of carry trade. The envelope module in gray dotted line wrapped the inlier instances in the envelope, the orange solid line represents the predicted return of carry trade.

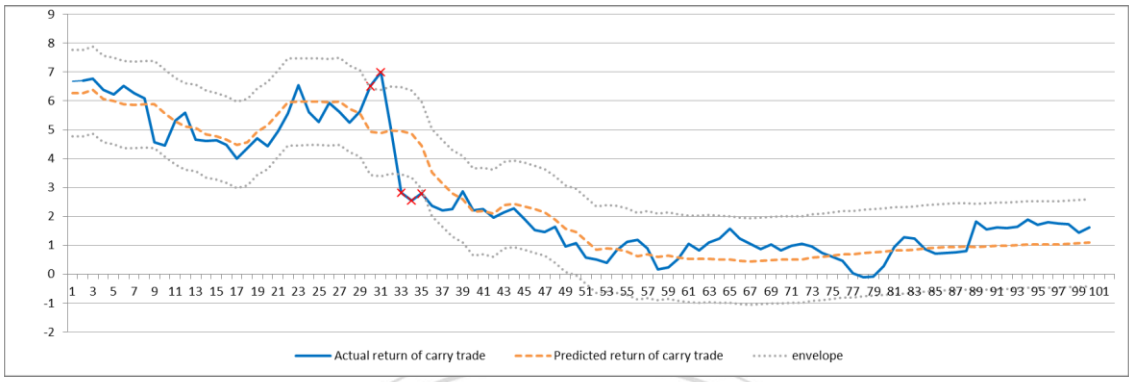


Figure 10: The envelope module of the Step 1.1 to identify the outliers with $\sigma = 1.5214$

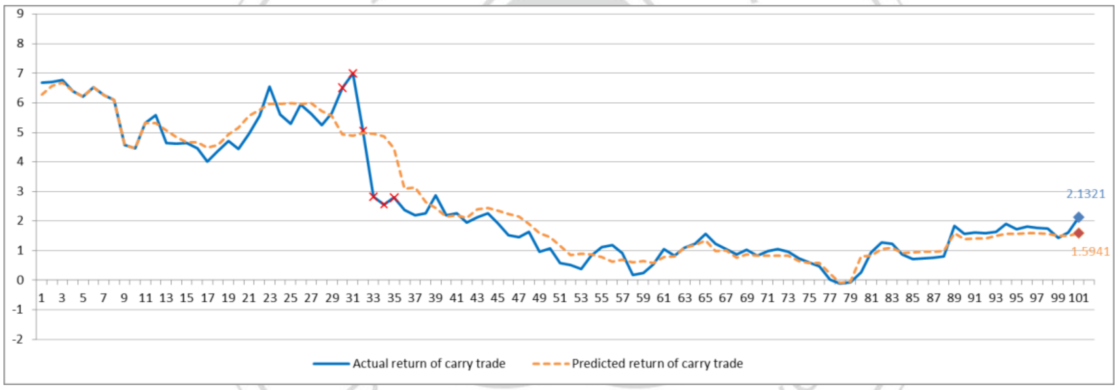


Figure 11: Actual and predicted return of carry trade with $\varepsilon = 0.75$

Table 7 shows the deviation between actual and predicted return of carry trade in different σ for two moving windows. The mean and variance of inlier training instances are both smaller than $\sigma = 1.5409$ (Step 1.1) and $\varepsilon = 0.75$ (Step 1.2) for the first moving window. In the second moving window, the mean and variance of inlier training instances are also both smaller than $\sigma = 1.5214$ (Step 1.1) and $\varepsilon = 0.75$ (Step 1.2).

In Table 8, it shows the accuracy of movement prediction for the Step 1.1 and Step 1.2. Step 1.1 can recognize positive movement but can't recognize negative movement of the training instances. After Step 1.2, both of positive and negative movement can be

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

recognized correctly for the two moving windows. The smaller variance of deviation indicates that we had overcome overfitting and reduced deviation.

Table 7: Deviation between actual and predicted return of carry trade

Moving window	RLEM	Step 1.1			Step 1.2	
		training instance (inlier:95)	training instance (all:100)	1 testing instance	training instance (inlier:95)	1 testing instance
1	Mean of Deviation	0.4371	0.5107	0.5493	0.2275	0.2446
	Variance of Deviation	0.0784	0.1807	/	0.0478	/
2	Mean of Deviation	0.5233	0.4477	0.5492	0.2395	0.5380
	Variance of Deviation	0.1855	0.0755	/	0.0440	/

Table 8 shows accuracy of movement prediction for the Step 1.1 and Step 1.2

M	Actual movement	Step 1.1						Step 1.2			
		training instance (inlier:95)		training instance (all:100)		1 testing instance		training instance (inlier:95)		1 testing instance	
		+	-	+	-	+	-	+	-	+	-
1	+	93/93	N/A	98/98	N/A	1/1	N/A	93/93	N/A	1/1	N/A
	-	2/2	N/A	2/2	N/A	N/A	N/A	N/A	2/2	N/A	N/A
2	+	93/93	N/A	98/98	N/A	1/1	N/A	93/93	N/A	1/1	N/A
	-	2/2	N/A	2/2	N/A	N/A	N/A	N/A	2/2	N/A	N/A

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Chapter 5 Conclusion and Future Work

This research refined a mechanism for data cleaning and machine learning to cope with outliers and predict the return of carry trade. The refined mechanism is implemented via the Tensowflow and GPU. We add the regularization to the loss function to prevent the problem of overfitting, and refine the cramming mechanism and weight tuning mechanism of RLEM to improve the accuracy of the predicted return of carry trade.

After the experiment, we find out that the regularization term reduce the overfitting, we not only get less error in the training data but also less error in testing data. The testing result is getting closer to the actual predicted return of carry trade but it takes more time in learning with the refined backpropagation of RLEM.

The experiment result indicates that the refined mechanism can predict movement of the return of carry trade better. We find out some valuable issues in the experiment:

1. Issue on technology field: Because of the less knowledge and new software library of machine learning, the version of Tensorflow is still being updated and improved, we have to try constantly and get more information about it. To improve the training models, we must need to choose a suitable way via knowing the reason of the error between the desired and actual return of carry trade.
2. Issue on application field: Although we had refined the ANN mechanism, it still needs to concentrate on finding other ways to achieve real-world solution such as getting more feature as input which can affect the consquence of output results especially for the finance field. It is also significant for us to assign parameters and moving windows properly of the experiment model.
3. Issue on hardware: Besides using the Tensorflow, it is also important to use TensorFlow running faster on the latest GPUs and scales well across GPUs to train the models in hours instead of days.

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

There are several research limitations that need to be improved and future goals as the following:

1. We prevent overfitting but the testing data is not precisely to the actual return of carry trade at all, so it might need more data for the learning process. Try to make all of the moving windows finish and get more information of the entire experiment.
2. It takes a lot of time for learning, so we can improve the hardware or adopt multiple GPUs with different machine to reduce the learning time and finish the moving windows.

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

Reference

1. Android Authority (2018) “Artificial intelligence vs machine learning : what’s the difference?”, available at https://www.androidauthority.com/artificial-intelligence-vs-machine-learning-832331/ (accessed 5 March 2018)
2. J. Cao, Y. Pang, X. Li, J. Liang (2018) “Randomly translational activation inspired by the input distributions of ReLU,” Neurocomputing (275), pp:859-868
3. D.A. Clevert, T. Unterthiner, S. Hochreiter (2016) “Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs),” Published as a conference paper at ICLR
4. Educational Research Techniques (2016) “Black Box Method-Artificial Neural Networks”, available at https://educationalresearchtechniques.com/2016/07/06/black-box-method-artificial-neural-networks/ (accessed 5 March 2018)
5. Enhance Data Science (2017) “Machine Learning Explained: Regularization”, available at http://enhancedatascience.com/2017/07/04/machine-learning-explained-regularization/ (accessed 5 March 2018)
6. I. Goodfellow , Y. Bengio, A. Courville (2016), “Deep Learning,” The MIT Press
7. S. Y. Huang, J. W. Lin, and R. H. Tsaih (2106), “Outlier Detection in the Concept Drifting Environment,” In: Proceedings of the International Joint Conference on Neural Networks (IJCNN), pp:31-37
8. S. Y. Huang, F. Yu, R. H. Tsaih, and Y. Huang (2104), “Resistant learning on the envelope bulk for identifying anomalous patterns,” In: Proceedings of the 2014 International Joint Conference on Neural Networks (IJCNN), pp:3303-3310

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

9. Investopedia "Carry Tradde" available at https://www.investopedia.com/terms/c/carry-trade.asp-0 (accessed 20 March 2018)

10. Ò . Jordà, and A. M. Taylor (2012), "The carry trade and fundamentals: Nothing to fear but FEER itself," Journal of International Economics, vol. 88, pp:74-90

11. F. F. Li, J. Johnson, S. Yeung (2017), "Convolutional Neural Networks for Visual Recognition, Stanford University School of Engineering," available at http://cs231n.stanford.edu/ (accessed 5 March 2018)

12. J. D. Olden, M. K. Joy, R. G. Death (2004), "An accurate comparison of methods for quantifying variable importance in artificial neural networks using simulated data," Ecological Modeling (178:3), pp:389-397

13. Quora (2013), "Differences between L1 and L2 as Loss Function and Regularization", available at http://www.chioka.in/differences-between-l1-and-l2-as-loss-function-and-regularization/ (accessed 5 March 2018)

14. S. Ruder (2016), "An overview of gradient descent optimization algorithms", available at http://ruder.io/optimizing-gradient-descent/index.html#adam (accessed 5 March 2018)

15. N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov (2014) "Dropout: A Simple Way to Prevent Neural Networks from Overfitting," Journal of Machine Learning Research (15) 2014, pp:1929-1958

16. The Theory of Everything (2017), "Understanding Activation Functions in Neural Networks", available at https://medium.com/the-theory-of-everything/understanding-activation-functions-in-neural-networks-9491262884e0 (accessed 5 March 2018).

17. Towards Data Science (2017), "Types of Optimization Algorithms used in Neural Networks and Ways to Optimize Gradient Descent", available at

DOI:10.6814/THE.NCCU.MIS.011.2018.A0

https://towardsdatascience.com/types-of-optimization-algorithms-used-in-neural-networks-and-ways-to-optimize-gradient-95ae5d39529f (accessed 5 March 2018).

18. R. H. Tsaih, T. C. Cheng (2009), "A resistant learning procedure for coping with outliers," Annals of Mathematics and Artificial Intelligence (57:2), pp:161-180
19. J. V. Tu (1996), "Advantages and disadvantages of using artificial neural networks versus logistic regression for predicting medical outcomes" Journal of Clinical Epidemiology 49(11), pp:1225-1231.
20. F. Y. Tzeng, K. L. Ma (2005), "Opening the Black Box — Data Driven Visualization of Neural Networks", Visualization, IEEE
21. L. Wan, M. Zeiler, S. Zhang, Y. L. Cun, R. Fergus (2013), "Regularization of Neural Networks using DropConnect" Proceedings of the 30th International Conference on Machine Learning, PMLR (28:3), pp:1058-1066
22. J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, Y. Ma (2009), "Robust face recognition via sparse representation," IEEE Transactions (31:1), pp:210-227
23. J. Wu. (2017), "Application of Machine Learning to Predicting the Returns of Carry Trade. Unpubliched Master Thesis," National Chengchi University, Taipei
24. S. N. Zeng, J. P. Gou, L. M. Deng (2017), "An antinoise sparse representation method for robust face recognition via joint l1 and l2 regularization," Expert Systems with Applications (82), pp:1-9

DOI:10.6814/THE.NCCU.MIS.011.2018.A0