Application of the Selective Desensitization Neural Network to Concept Drift Problems

Tomoki Ichiba¹, Kazumasa Horie², Shoichi Someno¹, Tomohiro Aki¹ and Masahiko Morita¹

¹ University of Tsukuba
1–1–1 Tennodai, Tsukuba Ibaraki 305–8573 Japan
Phone/FAX:+81-29-853-6554
E-mail: ichiba@bcl.esys.tsukuba.ac.jp
someno@bcl.esys.tsukuba.ac.jp
aki@bcl.esys.tsukuba.ac.jp
mor@bcl.esys.tsukuba.ac.jp

² University of Tsukuba
1–1–1 Tennodai, Tsukuba Ibaraki 305–8577 Japan
E-mail: horie@bipl-sdnn.org

Abstract

Previous studies have shown that a selective desensitization neural network (SDNN) has a high function approximation ability, low hyper-parameter dependency, and suitability for online incremental learning. These properties suggest that an SDNN can deal well with temporal changes in the characteristics of data, or concept drift, although this has not been verified. In this study, we conducted experiments on online learning using an artificial dataset generated using a time-varying function and a real-world dataset on a stock prices index, and evaluated the effectiveness of an SDNN for concept drift problems. The results show that an SDNN exhibited a superior performance than existing methods for both datasets, suggesting that an SDNN is highly suitable for certain types of concept drift problems.

1. Introduction

Concept drift is a problem in which the data characteristics, such as the input-output relation and the input distribution, change over time. Examples of data with concept drift are stock prices, which have often been used as real-world datasets in previous studies [1, 2]. We focus on online prediction of concept drift problems using machine learning.

Conventional learning algorithms have a certain difficulty in dealing with concept drift, in that they require hyper-parameter optimization, or are unsuitable for online incremental learning. Recent studies have shown that a selective desensitization neural network (SDNN) may be suitable because it has high expression and generalization abilities and low hyper-parameter dependency, and can fit additional data without significantly disrupting previously learned information. However, such suitability has yet to be verified.

In the present study, we conducted experiments on online prediction and evaluated the effectiveness of an SDNN for concept drift.

2. Selective desensitization neural network

The SDNN function approximator is constructed by applying manipulations of pattern coding and selective desensitization to a parallel perceptron (PP), which is the only part that an SDNN trains. The structure of an SDNN is shown in Figure 1. Details of pattern coding, selective desensitization, and PP are described in this section.

2.1 Pattern coding

Pattern coding converts each analog value into binary (±1) n-dimensional vectors as code patterns. In pattern coding, Q code patterns are created, which correspond to an input space divided into Q. There are several methods used to create code patterns, and herein we explain the method used in the present study.

First, we create P₁, the first code pattern, by selecting +1 and −1 randomly for each element, such that half of the elements take positive values, and the rest take negative values.
Second, we create \( P_2 \) by inverting the sign of the \( r \) elements of \(+1\) and the \( r \) elements of \(-1\) of \( P_1 \), which are selected randomly. Subsequently, we repeat this process for creating \( P_k \) based on \( P_{k-1} \) until \( P_1 \) to \( P_Q \) are created.

The results of this method ensure that the correlation between two consecutive patterns is high, and that the correlation between two patterns decreases as the two patterns are separated.

### 2.2 Selective desensitization

Selective desensitization integrates two binary code patterns into a single ternary \((-1, 0, 1)\) pattern by modifying one pattern with the other.

As an example, we consider the case of desensitizing the pattern \( S = (s_1, ..., s_n) \) with the pattern \( C = (c_1, ..., c_n) \). In desensitization, it is assumed that there is a random one-to-one correspondence between \( S \) and \( C \) elements. When correspondence exists between \( s_i \) and \( c_j \), \( s_i \) is desensitized through the following expression.

\[
s'_i = \frac{s_i(1 + c_j)}{2} \quad (1)
\]

When \( c_j = -1 \), \( s_i \) is desensitized, and \( s'_i \) becomes a neutral value \((= 0)\). When \( c_j = 1 \), \( s'_i \) becomes a value of \( s_i \). As a result, half of the desensitized pattern elements are 0, and the rest are 1 or –1.

### 2.3 Parallel perceptron

A PP consists of \( m \) simple perceptrons (SPs) with a Heaviside function as an activation function. The output of the PP is obtained from the number of SPs that output a value of 1. The p-delta method \([3]\) is used for learning.

As an example, we consider a case in which only \( k \) SPs output 1, although \( l > k \) SPs should output 1 for the input. At this time, we apply error correction learning for \( l - k \) SPs, which are selected from the \( m - k \) SPs that output 0 in the order of internal potential closer to the threshold value \( (= 0)\). In the case of \( k > l \), the same error correction learning is conducted for \( k - l \) SPs selected from the \( m - k \) SPs that output 1.

### 3. Experiment with an artificial dataset

We compared an SDNN and other methods using an artificial dataset. Concept drift is categorized into several patterns, such as sudden drift, incremental drift, and reoccurring drift \([1]\). In this experiment, we focus on both sudden and incremental drift.

#### 3.1 Methods

An artificial dataset was generated by sampling an artificial time-varying function \( f(x, y; t) \).

\[
k(x, y) = \frac{1 + x}{4} \cos(2\pi x y^2) + \frac{1}{2}
\]

\[
l(x, y) = \begin{cases} 
0.5 & (0.7 \leq x \leq 0.95 \land 0.1 \leq y \leq 0.9) \\
1 & ((x - 0.25)^2 + (y - 0.75)^2 < 0.04) \\
\frac{1 + x}{4} \cos(5\pi x y^2) + \frac{1}{2} & \text{(otherwise)}
\end{cases}
\]

\[
g(x, y; t) = \begin{cases} 
\frac{1}{2000} & (t \leq 1500) \\
\frac{2000 - t}{2000} & (1500 \leq t \leq 2500) \\
\frac{t - 2000}{2500} & (2500 \leq t < 3000) \\
\frac{t - 3000}{t} & (t \geq 3000)
\end{cases}
\]

\[
f(x, y; t) = \begin{cases} 
1 & (g(x, y; t) > 1) \\
0 & (g(x, y; t) < 0) \\
g(x, y; t) & \text{(otherwise)}
\end{cases}
\]

The value of this function gradually decreases during \( 1000 < t \leq 1500 \), increases during \( 1500 < t \leq 2500 \), and decreases during \( 2500 < t \leq 2999 \) to the initial value (Figures 2a–2e), which can be regarded as incremental drift. It then changes abruptly at \( t = 3000 \) (sudden drift); the spatial frequency of the function increases and discontinuous domains appear (Figure 2f).

At each time step, one sample was obtained randomly from the lattice points of 0.01 intervals within the input domain \( \{ (x, y) \mid x, y \in [0, 1] \} \), which was given to each function
approximator only once for training (and discarded at the next time step). The mean absolute error was then calculated from the approximation error at all (101 \times 101) lattice points. The total number of time steps was 7,000. For comparison, we tested other function approximators: a multi-layer perceptron (MLP), a radial basis function network (RBFN) [4], and an incremental normalized Gaussian network (INGnet) [5].

**MLP** One hidden layer of 50 units was applied between the input and output layers. For the hidden layer, a hyperbolic tangent (tanh) was used as the activation function, whereas the linear function was used for the output layer. Training was conducted using a backpropagation algorithm. We repeated the above experiment ten times using different initial synaptic weights.

**RBFN** We set the lattice points of \( s \) intervals in the input domain, which were the center of the basis functions. We used a Gaussian function as the basis function. Intervals \( s \) greatly affect the expression and generalization abilities of an RBFN. In this experiment, we used two types of RBFN. One is termed RBFN\(_{441}\), whose parameter are \( s = 0.05 \), a standard deviation of \( \sigma = 0.05 \), and 441 basis functions. The other is termed RBFN\(_{81}\), whose parameters are \( s = 0.125 \), a standard deviation of \( \sigma = 0.125 \), and 81 basis functions. Training was conducted using a gradient descent.

**INGnet** We used a normalized Gaussian function as the basis function, which was additionally set during the learning phase. We added a basis function when the absolute error of the training was larger than the threshold value \( e_{\text{max}} \), and when the value of all existing basis functions was smaller than the threshold value \( e_{\text{min}} \). The parameters were \( e_{\text{max}} = 0.05 \), \( e_{\text{min}} = 0.5 \), and \( \sigma = 0.05 \). Training was conducted using a gradient descent.

**SDNN** In this experiment, we used LIBSDNN\(^1\). The pattern coding parameters were \( n = 400 \), \( q = 201 \), and \( r = 2 \). The number of SPs \( m \) was set to 280, and the output of the SDNN was calculated using \( 0.005k - 0.2 \), where \( k \) is the number of SPs with outputs of 1.

Training was repeated until the absolute training error reached below 0.01 for each function approximator.

### 3.2 Results

Table 1 shows the average approximation error over all time steps, in which the error was lowest for the SDNN. Figure 3 shows the temporal changes in the mean absolute error for each function approximator.

MLP exhibited the largest error at any time steps, indicating that MLP is not good at online incremental learning. The error of INGnet did not decrease during \( 1000 < t < 3000 \), probably because the number of bases increased with a decrease in generalization ability and thus given samples were insufficient for INGnet to follow the changes of the target function. Here, RBFN\(_{81}\) reduced the error during the early time steps and adapted to the gradual change at \( 1000 \leq t < 3000 \), but not at \( t \geq 3000 \), indicating that it is unable to represent a complex function with 81 basis functions. However, RBFN\(_{441}\) was able to fit the complex function but reduced the error only slowly for \( t \leq 1000 \) and could not adapt to a gradual change during \( 1000 < t \leq 3000 \), indicating that it requires more samples. Thus, RBFN cannot adapt to both sudden and incremental drifts.

In contrast, the SDNN was able to adapt to both drifts, indicating that it can fit a simple target function with a small number of samples, and can represent a complex function.

### 4. Experiment using real-time dataset

We also applied an SDNN to the prediction of future closing prices of Nikkei225 on the Tokyo Stock Exchange\(^2\) from previous closing prices. This price usually changes gradually, but occasionally does so extremely rapidly, and thus has been used in previous studies on concept drift [1, 2].

#### 4.1 Methods

We used the same data (from May 19, 1979 to May 15, 2017) and procedure as in [1].

A simple method used to handle streaming data with concept drift is to use a sliding window that keeps \( k \) newest samples. These \( k \) samples are normalized and input into an SDNN trained to predict the value \( N \) steps ahead (\( N = \)

\(^1\)https://github.com/BIPL-HORIE/LIBSDNN

\(^2\)https://finance.yahoo.com/quote/%5EN225/
This indicates that an SDNN is adaptable to changes in the characteristics of the data. Figure 4 shows the predicted value of ARWin and an SDNN for \( N = 3 \). ARWin exhibits large prediction errors, particularly when significantly changing from a rising to falling trend or vice versa, but SDNN does not.

It should be noted that most of the error values of the SDNN were smaller than those of the OPOSSAM, which exhibited the best performance in a previous study \([1]\), although we did not test this method ourselves. We also conducted the same experiment using data on other terms (from May 21, 1979 to May 19, 1997), and confirmed that the SDNN exhibited almost the same performance, indicating the generality of the results.

5. Conclusions

We showed that an SDNN has an excellent capability regarding online incremental learning for a time-variant function approximation. We also applied the SDNN to a real-world time-series prediction task and obtained a superior performance over existing methods. These results suggest that an SDNN is highly suitable for certain types of concept drift problems. In future research, we will apply an SDNN to other concept drift problems to confirm its effectiveness.

Acknowledgment

This work was supported partly by JSPS KAKENHI, Grant No. JP18H03304.

References


