Journal of Signal Processing, Vol.23, No.4, pp.145-149, July 2019

SELECTED PAPER AT NCSP'19

Application of a Selective Desensitization Neural Network to Concept Drift Problems

Tomoki Ichiba[1], Kazumasa Horie[2], Shoichi Someno[1], Tomohiro Aki[1] and Masahiko Morita[3]

[1] Graduate School of
Systems and Information Engineering,
University of Tsukuba
1–1–1 Tennodai, Tsukuba,
Ibaraki 305–8573, Japan
E-mail: ichiba@bcl.esys.tsukuba.ac.jp
someno@bcl.esys.tsukuba.ac.jp
aki@bcl.esys.tsukuba.ac.jp

[2] Center for Computational Sciences,
University of Tsukuba
1–1–1 Tennodai, Tsukuba,
Ibaraki 305–8577, Japan
E-mail: horie@bipl-sdnn.org

[3] Faculty of Engineering,
Information and Systems,
University of Tsukuba
1–1–1 Tennodai, Tsukuba,
Ibaraki 305–8573, Japan
E-mail: mor@bcl.esys.tsukuba.ac.jp

Abstract

A selective desensitization neural network (SDNN) has a high function-approximation ability, low hyperparameter dependence, 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 of a stock prices index, and evaluated the effectiveness of an SDNN to solve concept drift problems. The results show that using the SDNN exhibited superior performance over the 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 hyperparameter optimization, or are unsuitable for online incremental learning. Recent studies [3, 4] have shown that a selective desensitization neural network (SDNN) has high expression and generalization abilities and low hyperparameter dependence. In addition, we found that an SDNN can fit additional data without significantly disrupting previously learned information, indicating that it is suitable for online incremental learning, and that it may deal well with concept drift. However, an SDNN has not been applied to concept drift problems to test its performance.

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

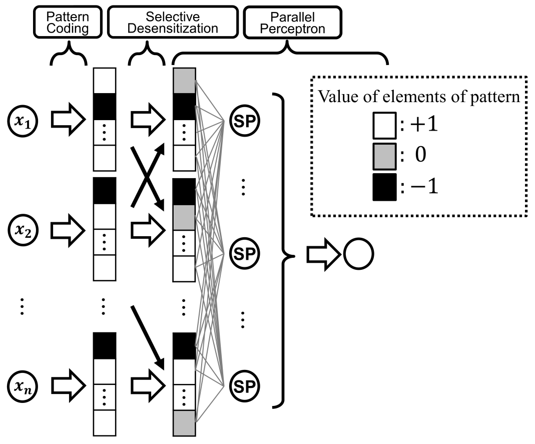

Figure 1: Structure of SDNN

2. Selective Desensitization Neural Network

An 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 Fig. 1. Details of pattern coding, selective desensitization, and a 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, which correspond to an input space divided into Q, are created. There are several methods of creating code patterns. We explain our method below.

First, we create P_1, 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 r elements of $+1$ and r elements of -1 of P_1, which are selected randomly. Subsequently, we repeat this process to create 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 increasingly 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 the elements of S and C. When correspondence exists between s_i and c_j, s_i is desensitized through the following expression.

$$s'_i = \frac{s_i(1 + c_j)}{2} \tag{1}$$

When $c_j = -1$, s_i is desensitized, and s'_i becomes a neutral value $(= 0)$. When $c_j = 1$, s'_i becomes the 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 [5] 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 increasing closeness of the internal potential 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 the 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.

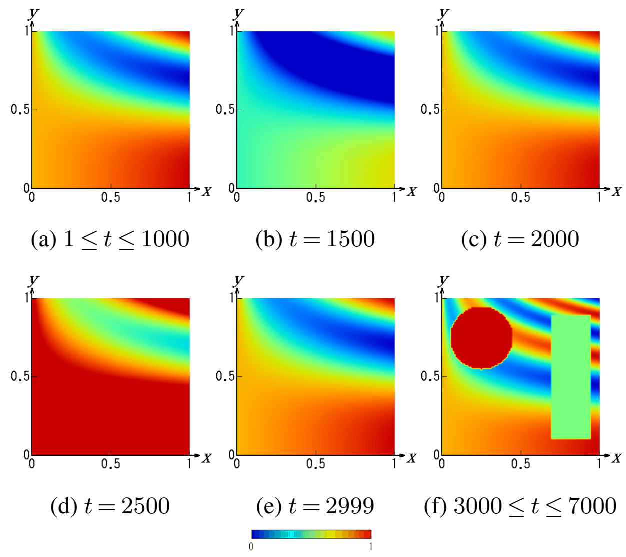

Figure 2: Target function at a certain time step t

3.1 Methods

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

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

$$g(x,y;t) = \begin{cases} k(x,y) & (t < 1000) \\ k(x,y) - \frac{t-1000}{1500} & (1000 \le t < 1500) \\ k(x,y) - \frac{2000-t}{1500} & (1500 \le t < 2500) \\ k(x,y) - \frac{t-3000}{1500} & (2500 \le t < 3000) \\ l(x,y) & (t \ge 3000) \end{cases}$$

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

$$k(x,y) = \frac{1+x}{4}\cos(2\pi\sqrt{x}y^2) + \frac{1}{2} \tag{2}$$

The value of this function gradually decreases during $1000 < t \le 1500$, increases during $1500 < t \le 2500$, and again decreases during $2500 < t \le 2999$ to the initial value (Fig. 2(a)–2(e)), 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 (Fig. 2(f)).

At each time step, one sample was obtained randomly from the lattice points at 0.01 intervals within the input domain

Table 1: Average approximation error over all time steps

MLP	RBFN_441	RBFN_81	INGnet	SDNN
0.220	0.064	0.071	0.055	0.047

$\{(x, y) \mid x, y \in [0, 1]\}$ and 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 (101×101) lattice points. The total number of time steps was 7,000. For comparison, we tested other function approximators: a multilayer perceptron (MLP), a radial basis function network (RBFN) [6], and an incremental normalized Gaussian network (INGnet) [7].

MLP One hidden layer of 50 units was applied between the input and output layers. For the hidden layer, the hyperbolic tangent (tanh) was used as the activation function, whereas a 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 at s intervals in the input domain, which were the center of the basis functions. We used a Gaussian function as the basis function. The interval s greatly affects the expression and generalization abilities of an RBFN. In this experiment, we used two types of RBFN. One is termed RBFN_441, whose parameters 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 $e_{\max}$ and when the value of all existing basis functions was smaller than the threshold value $a_{\min}$. The parameters were $e_{\max} = 0.05$, $a_{\min} = 0.5$, 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. Fig. 3 shows the temporal changes in the mean absolute error for each function approximator.

[1]https://github.com/BIPL-HORIE/LIBSDNN

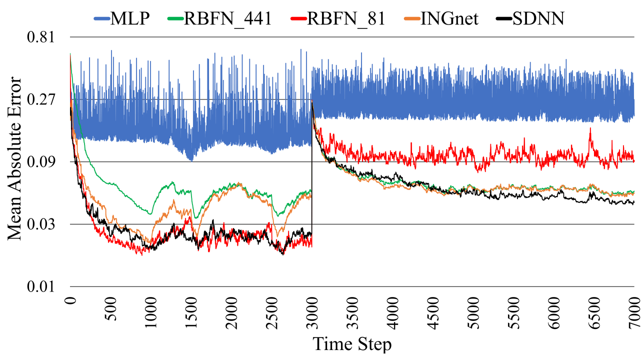

Figure 3: Temporal changes in approximation error

MLP exhibited the largest error at all time steps, indicating that it 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, the given samples were insufficient for INGnet to follow the changes in 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 to the abrupt change 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 the 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 also represent a complex function.

4. Experiment Using Real-Time Dataset

We also applied an SDNN to the prediction of future closing prices of the 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 of handling streaming data with concept drift is to use a sliding window that keeps the k newest samples. These k samples are normalized and input into an SDNN trained to predict the value N steps ahead ($N = 1, 3, 5$). Different SDNNs are used for different N. The dataset consists of 4,906 samples $d_1, \ldots, d_{4906}$. At time step

[2]https://finance.yahoo.com/quote/% 5EN225/

Table 2: Prediction errors for each method

	$N = 1$		$N = 3$		$N = 5$	
	RMSE	MAE	RMSE	MAE	RMSE	MAE
ARWin	299.45	222.12	387.99	290.04	467.19	351.94
SDNN	191.63	136.98	327.36	242.11	425.53	317.20
OPOSSAM (for reference)	200.35	142.90	347.73	252.50	444.58	331.09

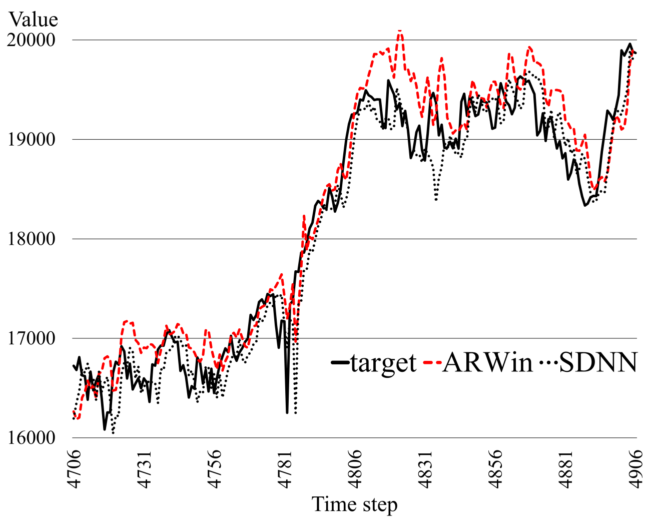

Figure 4: Stock price predicted by each approximator for the last 200 time steps for $N = 3$

t, the SDNN receives k samples $d_{t-N}, \ldots, d_{t-k-N+1}$, and is trained to predict d_t. At $t > 1000$, it also predicts $\hat{d}_{t+N}$ from $x_t, \ldots, x_{t-k+1}$, and the prediction error between d_{t+N} and $\hat{d}_{t+N}$ is calculated (we use both RMSE and MAE as metrics of the prediction accuracy).

We used ARWin [2] for comparison.

ARWin ARWin has a sliding window of k in size, and the sliding window induces k subwindows. Each subwindow has the i newest samples ($i = 1, \ldots, M$), which are used to calculate a predicted value based on linear regression. The output of ARWin is determined using the weighted sum of these values, where the weights are updated through learning. In this experiment, the window size k was set to $3, 4, \ldots, 50$, according to [1].

SDNN We used LIBSDNN with the parameters $n = 1250$, $q = 1251$, $r = 1$, and $m = 700$. The window size k was fixed to 50 for the SDNN. The 200-day moving average $\bar{d}_t = (d_t + \cdots + d_{t-199})/200$ was used as the baseline, that is, the input variable was given by $x_i = d_{t-i+1}/\bar{d}_t$ ($i = 1, \ldots, 50$), and $\hat{d}_{t+N}$ was obtained by multiplying the output value by $\bar{d}_t$.

4.2 Results

Table 2 shows the prediction errors, which indicate that the SDNN can predict future prices much better than ARWin. This indicates that the SDNN is adaptable to changes in the characteristics of the data. Fig. 4 shows the values predicted by ARWin and the SDNN for $N = 3$. ARWin exhibits large prediction errors, particularly when significantly changing from a rising to falling trend or vice versa, but the SDNN does not.

It should be noted that most of the error values of the SDNN were smaller than those of OPOSSAM, which exhibited the best performance in a previous study [1], although we did not test that method ourselves. We also conducted the same experiment using other data (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 for online incremental learning for a time-variant function approximation. We also applied an 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

[1] A. Yamaguchi, S. Maya, T. Inagi and K. Ueno: OPOSSAM: Online prediction of stream data using self-adaptive memory, IEEE International Conference on Big Data, 2018.

[2] S. Yoshida, K. Hatano, E. Takimoto and M. Takeda: Adaptive online prediction using weighted windows, The IEICE Transactions on Information and Systems, Vol. E-94-D, No. 10, pp. 1917-1923, 2011.

[3] K. Nonaka, F. Tanaka and M. Morita: Empirical comparison of feedforward neural networks on two-variable function approximation, The IEICE Transactions on Information and Systems (Japanese Edition), Vol. J-94-D-II, No. 12, pp. 2114-2125, 2011.

[4] K. Horie, A. Suemitsu, T. Tanno and M. Morita: Direct estimation of wrist joint angular velocities from surface EMGs by using an SDNN function approximator, Proceedings of the 23rd International Conference on Neural Information Processing (ICONIP), 2016.

[5] P. Auer, H. Burgsteiner and W. Maass: A learning rule for very simple universal approximators consisting of a single layer of perceptrons, Neural Networks, Vol. 21, pp. 786-795, 2008.

[6] D. S. Broomhead and D. Lowe: Radial basis functions, multi-variable functional interpolation and adaptive networks, Complex Systems, Vol. 2, No. 3, pp. 321-355, 1988.

[7] J. Morimoto and K. Doya: Learning dynamic motor sequence in high-dimensional state space by reinforcement learning: Learning to stand up, The IEICE Transactions on Information and Systems (Japanese Edition), Vol. J-82-D, No. 11, pp. 2118-2131, 1999.