



Hybridní rostoucí samoorganizující mapa Hybrid Growing Self-Organizing Map

Libor Horák

Abstrakt: Software se schopností se adaptovat najde uplatnění v mnoha oblastech například v kontrolních systémech, v herním průmyslu, simulacích nebo v systémech zaměřených na predikci. Umělé neuronové sítě mohou být dobrým základem těchto systémů. Hybridní rostoucí samoorganizující mapa představená v tomto článku se dokáže přizpůsobovat novým situacím v průběhu svého „života“. Tento online trénink je navrhnut tak, aby nedocházelo k zapomínání již naučených věcí. Tohoto chování je dosaženo spojením rostoucí samoorganizující mapy a dvouvrstvé neuronové sítě.

Klíčová slova: hybridní rostoucí samoorganizující mapa, umělá inteligence, katastrofické zapomínání, neuronové sítě, dilema plasticity a stability, klasifikace

Abstract: Software with adaptation ability find their application in many areas for example in control systems, game industry, simulations or in some prediction systems. The artificial intelligence neural networks can be a good base for this type of software. The hybrid growing self-organizing map architecture introduced in this article can adapt for new situations. It also allows an additional online training without a catastrophic forgetting. These abilities are achieved combining the growing self-organizing map and two layer feed-forward network.

Keywords: hybrid growing self-organizing map, artificial intelligence, catastrophic forgetting, neural networks, plasticity-stability dilemma, classification

JEL Classification: C45

Background

Software with adaptation ability find their application in many areas for example in control systems, game industry, simulations or in some prediction systems. The artificial intelligence neural networks can serve as a good base for this type of software.

Neural networks have several abilities.

- 1) Generalization
- 2) Classification
- 3) Filtration
- 4) Association
- 5) Optimization

Every network is being taught for some type of inputs in training (learning) process. In this process networks adapt their weights based on training patterns, which were prepared before. Most types of networks can react to untrained inputs too. This ability is called generalization, because the reaction of a network is created as a generalization based on its knowledge. This kind of adaptation to new inputs is great for time series predictions. But if we have completely different situation with new class of inputs, the generalization will not be enough to solve this situation and the network will need to learn again.

1. Catastrophic Forgetting and Stability-plasticity Dilemma

Human brain has a special ability to learn a new information without forgetting something. The most of computer programs and neural networks cannot do this. Some network types are losing their memory during the learning process and others even cannot learn at all and they can solve only one kind of problem – the one they were designed to.

The knowledge of the network is stored in weights. The outputs are a function of weights and inputs, hence if we change weights, the outputs and the generalization ability will be affected.

$$\text{outputs} = f(w_1, w_2, \dots, w_n, I_1, I_2, \dots, I_n)$$

If we want to adapt our network for a new situation with unknown inputs, we will have to train network again or we will have to change network's architecture. If we use first way and train network with new input patterns, we will change weights and for the same inputs we will get different outputs.

$$\text{outputs}_{\text{new}} = f(w_{1,\text{new}}, w_{2,\text{new}}, \dots, w_{n,\text{new}}, I_1, I_2, \dots, I_n)$$

$$\text{outputs} \neq \text{outputs}_{\text{new}}$$

After this additional training network is prepared to handle new situation, but change of weights has a negative effect to network's reaction to the old inputs. This behaviour is called a catastrophic forgetting, because network can easily lose all stored information in memory. This is a general problem for most of neural networks for example back-propagation networks or self-organizing maps.

The catastrophic forgetting is defined as a complete forgetting of previous learned information by a neural network exposed to new information. (Mermillod, Bugaiska, and Bonin, 2013, p.1)

Looking for a balance between learning new information and losing learned information is called a stability-plasticity dilemma. The stability-plasticity dilemma was first introduced by Stephen Grossberg (Grossber, 1980; Grossber, 2000; French, 1999).

Stability of the network means that a pattern should not oscillate among different cluster units at different stages of training.

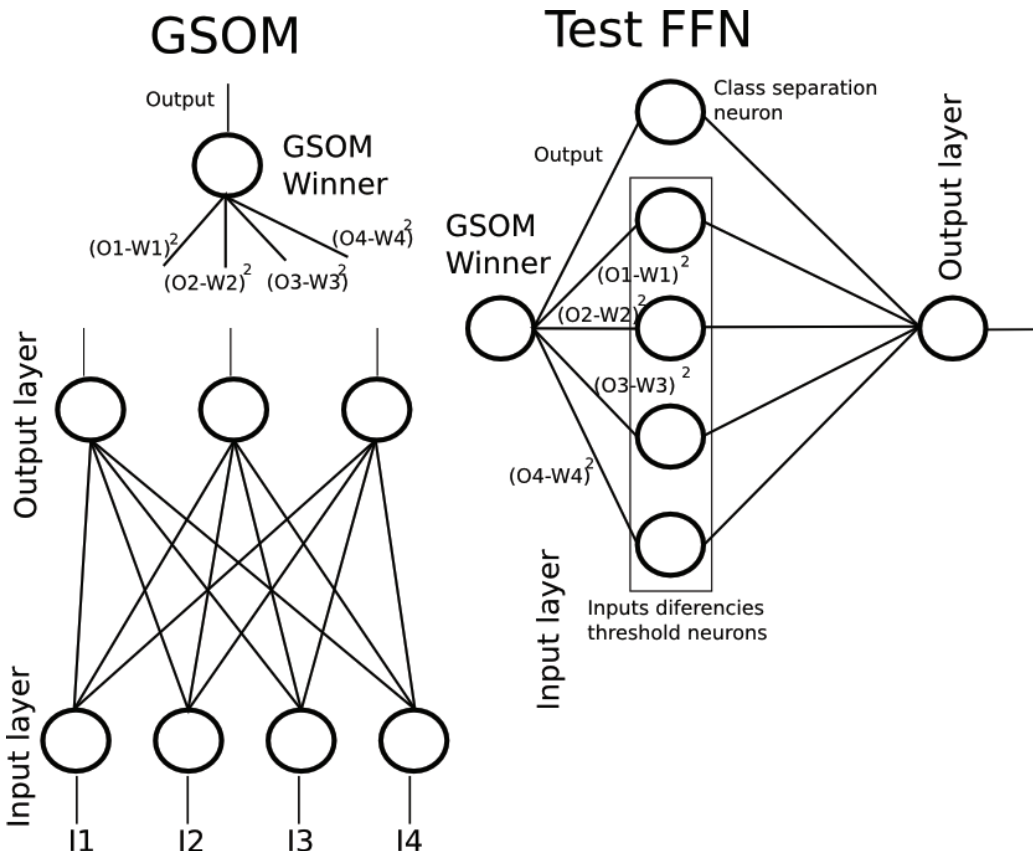
Plasticity is the ability of the net to respond to learn new pattern equally at any stage of learning. (Rajasekaran and Vijayalakshmi Pai, 2011, p.125)

The stability-plasticity dilemma must be solved by every brain system that needs to rapidly and adaptively respond to the flood of signals. (Grossberg 2000, p.2)

2. Growing Self-Organizing Map with Test Feed-forward Network

This paper shows hybrid system consisting of a growing self-organizing map (Villman, Bauer, 1998) (GSOM) and a two layer feed-forward neural network (FFN), which satisfies both the stability and the plasticity requirement and it is prevented from the catastrophic forgetting. The plasticity is achieved by changes of the GSOM architecture in the training process and the stability is guaranteed by the Test FFN. The whole system serves for inputs classification. Example of the system architecture is on figure 2.0.

Figure 2.0 - Hybrid self-organizing map architecture



As most of neural networks this system works in two modes.

- Calculation mode
- Training mode

The GSOM part creates system result in both modes and the FFN part handles results monitoring in the calculation mode and a results testing in the training mode.

2.1 Growing Self-Organizing Map Architecture (GSOM)

The growing self-organizing map has two layers called an input and an output. GSOM function is to classify an input vector \vec{I} to one class C_i from a group of learned classes C . C_i

$$C \in \{C_1, C_2, \dots, C_M\}$$

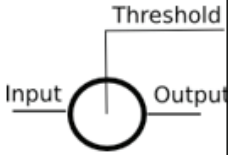
$$\vec{I} \rightarrow C_i$$

Its architecture and function is very similar as a common self-organizing map for example Kohonen network (Rajasekaran and Vijayalakshmi Pai, 2011; Kohonen 1990), but GSOM output layer can be extended by new neurons in the training mode. This way the network adapts itself to a new situation.

Input Layer

The input layer serves as an interface to the whole system and prepares the input vector \vec{I} for the output layer. It consists of input neurons. Number of input neurons N depends on the size of the input vector \vec{I} . The input vector \vec{I} is a situation description. Each value I_j represents some important situation attribute.

$$\vec{I} = \{I_1, I_2, \dots, I_N\}$$

GSOM Input Neuron	
	1
Inputs	$inputs \in (-\infty; \infty)$
Activation function	$f(x) = x$
Threshold	$threshold \in (-\infty; \infty)$
Weight	1
Output	$Output = f(W * I - threshold)$

Output Layer

Each output neuron in this layer symbolises one class C_i from the group C . GSOM output layer has an adaptation ability. It can be extended by a new output neuron in the training mode, so we can easily add new classes to our classification C . It is done by adding new neurons to the output layer. This principle follows the plasticity rule, because there is no change of weights in the other output neurons.

$$C_{new} \in \{C_{m+1}, C_{m+2}, \dots, C_{m+k}\}$$

$$C = C \cup C_{new}$$

Every output neuron has a connections to each input neurons. For each connection has a weight w_j . All w_j create weight vector \vec{W} and it represents an average representative \vec{R}_i from class C_i .

$$\vec{W} = \vec{R} = \{W_1, W_2, \dots, W_N\}$$

GSOM Output Neuron		
Number of inputs	N	
Threshold	0	
Weights	Variable	
Output	$Output = \sum_{i=0}^N (I_i - W_i)^2$	

The output value of output neuron represents a distance between average class representative \vec{R}_i and the input vector \vec{I} . In other words how much is the input vector \vec{I} different from average class representative \vec{R}_i .

Finding Winning Neuron

The Winning neuron is defined as an output neuron with the minimal value of the output O_{win} , hence its average class representative \vec{R}_i is the nearest to the input vector \vec{I} and the winner represents the most suitable class C_{win} for the input vector \vec{I} .

$$O_{win} = \min (O_1, O_2, \dots, O_{num.of.outputneurons})$$

Output Quality

The quality of an output is defined as a function of inputs differences $Id_{win,i}$ between single attributes I_i of the input vector \vec{I} and single winning neuron weights $W_{win,i}$. The last quality parameter is called class difference Cd_{win} and it is equal to winner's output value O_{win} .

$$Id_{win,i} = (I_i - W_{win,i})^2$$

$$Cd_{win} = O_{win} = \sum_{i=1}^N (Id_{win,i})$$

$$Quality = f(Cd_{win}, Id_{win,1}, Id_{win,2}, \dots, Id_{win,N})$$

This extended approach to the output quality dealing with every input attribute I_i individually allows to separate inputs to defined classes more accurately.

2.2 Test Feed-forward Network (FNN) Architecture

The Test feed-forward network function is testing quality of the output. If quality test fails in the training mode, new output neuron to the GSOM output layer will be added and the system will adapt to a new class of inputs.

We can test quality of the output in the calculation mode too. There won't be any effects on the system architecture, if test fails, but we can log this failure as an information about new unknown situation in the system environment. Based on this information we can create new training patterns and train the network again.

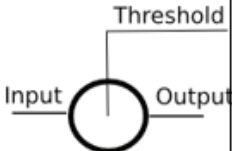
Input Layer

The input layer is used to compare input differences $Id_{win,i}$ and the class difference Cd_{win} with input precisions Ip_i and a class precision Cp . Precisions define borders between classes C_i in our classification C . Higher precision values create bigger classes otherwise lower values make classification more sensitive and sort inputs to more smaller classes.

FNN input vector $\overrightarrow{I_{test}}$ is created by all differences $Id_{win,i}$ and Cd_{win} hence the input layer has $N + 1$ neurons. Where N is number of GSOM input neurons.

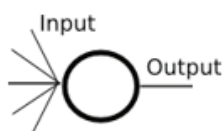
$$\overrightarrow{I_{test}} = \{O_{win}, (I_1 - W_{win,1})^2, (I_2 - W_{win,2})^2, \dots, (I_N - W_{win,N})^2\}$$

Each input neuron is one little precision test so the class precision Cp is set as a threshold for first input neuron and all input precisions Ip_i are thresholds for remaining input neurons. If some difference is higher than precision, input neuron will be activated and this part of precision test will fail.

FNN Input Neuron	
Number of inputs 	1
Activation function	$f(x) = \begin{cases} x > 0 \rightarrow 1 \\ x \leq 0 \rightarrow 0 \end{cases}$
Threshold	Variable
Weight	1
Output	$Output = f(W * I - threshold)$

Output Layer

The output layer has only one output neuron, its function is to decide if the output quality test fails or not. This decision is based on outputs from all input layer neurons. If one of them is activated, the output neuron will be activated too and the quality test will fail, otherwise the output neuron will remain inactive and the quality test will pass.

FNN Output Neuron	
	
Number of inputs	1
Activation function	$f(x) = \begin{cases} x > 0 \rightarrow 1 \\ x \leq 0 \rightarrow 0 \end{cases}$
Threshold	0
Weight	1
Output	$Output = f\left(\sum_{i=0}^{N+1} W_i I_i\right)$

3. Fast training process

In the training mode (figure 3.0) GSOM weights and architecture are adapted. Test FNN has only testing function and there are not any weights or architecture changes. The fast training process has only one training epoch, so the network learns each training pattern only once.

Training of the Untrained Network

The untrained network starts with only one neuron in the GSOM output layer. This neuron takes the attributes of the first training pattern as its weights.

$$\overrightarrow{R}_{first} = \overrightarrow{W}_{first} = \overrightarrow{P}_{first} = \{W_1, W_2, \dots, W_N\} = \{P_1, P_2, \dots, P_N\}$$

First class C_{first} and its average class representative R_{first} are created. There won't be any changes of weights for this neuron in the training process anymore. This "no change of weights" principle is applied to all future output neurons. When a neuron is created, its weights are set to constant values. This approach makes training process liable on the training patterns order. This can be a limitation in some situations.

Training process in steps:

- 1) Calculate output in the GSOM input layer for the first training pattern
- 2) Set first output neuron weights
- 3) While is there some untrained pattern do
- 4) Calculate output in the GSOM input layer
- 5) Calculate output in the GSOM output layer
- 6) Find winning neuron in GSOM
- 7) Test quality of the GSOM output in Test FNN
- 8) If test failed add new neuron
- 9) Set new neuron weights
- 10) Go to step 2
- 11) End of training

Adding new neuron

If FNN precision test failed, a new neuron will be added to the GSOM output layer. This neuron represents new class C_{m+1} . The new neuron weights will be equal to individual attributes P_i of the training pattern \vec{P} .

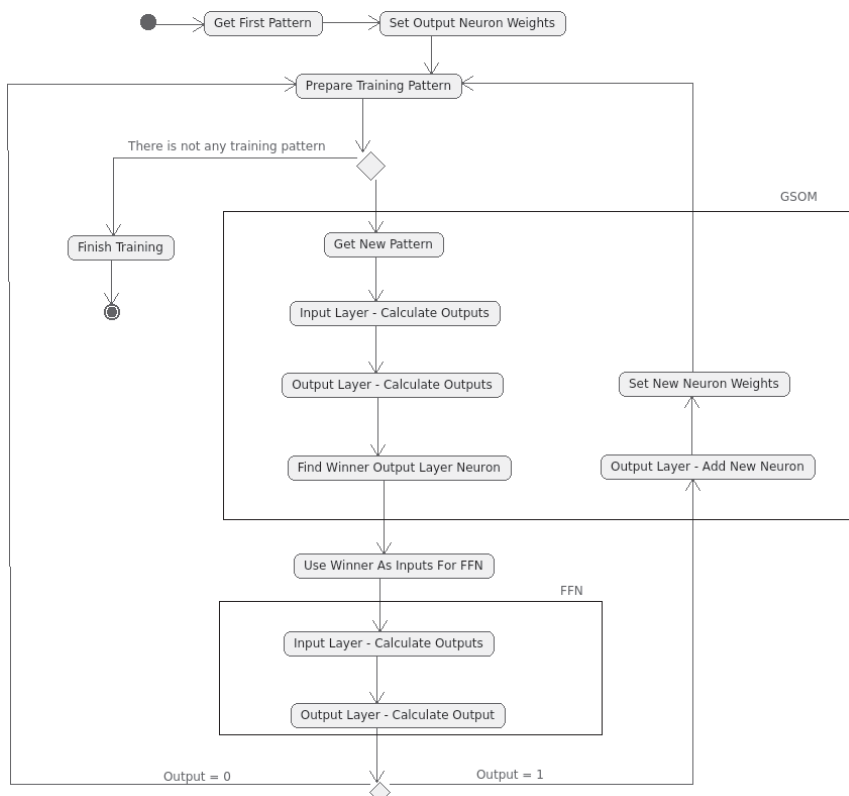
$$\vec{R}_{new} = \vec{W}_{new} = \vec{P} = \{W_1, W_2, \dots, W_N\} = \{P_1, P_2, \dots, P_N\}$$

There are no changes in other neurons, hence plasticity principle is satisfied and the catastrophic forgetting is prevented.

Additional Online Training

Thanks this ability we can switch to the training mode after some time in the calculation mode. I call it an “additional online training”, because the network is adapting during runtime. The additional online training process starts at the step 3 in training process steps. After the additional training network handles known inputs with same output and for unknown inputs creates a new better output.

Figure 3.0 - Fast training process



Conclusion

The hybrid growing self-organizing map architecture introduced in previous chapters can adapt for new situations. It also allows the additional online training without the catastrophic forgetting. These abilities combined with the fast training process without weight changing make this architecture suitable for situations, when we have some partial classification prepared from previous analyses and we want to extend this classification by new classes. Previous analyses can be for example a trained Kohonen map or some growing self-organizing map.

Growing Self-Organizing Map with Test Feed-forward Network is suitable for:

- Controlling and monitoring systems which work in unpredictable environment
- Testing systems when we want to test if our classification meets precision for all the inputs
- Interactive simulations systems e.g. building evacuation
- Games

Finding proper settings (GSOM class precision and inputs precisions) of the Test feed-forward network without a previous analyse can be difficult. This limitation makes described system unsuitable for analysing unknown situations.

Reference

- [1] Rajasekaran, S., Vijayalakshmi Pai, G.A. Neural Networks, Fuzzy Logic, and Genetic Algorithms Sythesis and Applications, PHI Learning Private Limited, New Dehli 2011
- [2] French, R. M. Catastrophic forgetting in connectionist networks, Trends in Cognitive Science. 1999, no. 4, vol. 3.
- [3] Grossberg, S. How Does a Brain Build a Cognitive Code?. Psychological Review. 1980, no. 1., vol. 87.
- [4] Grossberg, S. Adaptive Resonance Theory. Technical Report CAS/CNS-2000-024. Boston 2000.
- [5] Kohonen, T. The Self-Organization Map, Proceedings of the IEEE. 1990, no. 9., vol. 78.

- [6] Mermillod, M., Bugaiska, A., Bonin P. The stability-plasticity dilemma: investigating the continuum from catastrophic forgetting to age-limited learning effects. *Frontiers in Psychology*. 2013, vol. 4.
- [7] Villman, Th., Bauer, H.-U. Applications of the growing self-organization map, *Neurocomputing*, 1998, vol. 21

Ing. Libor Horák, České Vysoké učení technické v Praze, Thákurova 7, 166 29 Praha 6, email: libor.horak1@gmail.com