Method for training a spiking neuron to associate input-output spike trains

Ammar Mohemmed, Stefan Schliebs, *Satoshi Matsuda, and Nikola Kasabov

Knowledge Engineering Discovery Research Institute,
350 Queen Street, Auckland 1010, New Zealand
http://www.kedri.info
*Department of Mathematical Information Engineering,
Nihon University, Japan
{amohemme,sschlieb,nkasabov}@aut.ac.nz,matsuda.satoshi@nihon-u.ac.jp

Abstract. We propose a novel supervised learning rule allowing the training of a precise input-output behavior to a spiking neuron. A single neuron can be trained to associate (map) different output spike trains to different multiple input spike trains. Spike trains are transformed into continuous functions through appropriate kernels and then Delta rule is applied. The main advantage of the method is its algorithmic simplicity promoting its straightforward application to building spiking neural networks (SNN) for engineering problems. We experimentally demonstrate on a synthetic benchmark problem the suitability of the method for spatio-temporal classification. The obtained results show promising efficiency and precision of the proposed method.

Keywords: Spiking Neural Networks, Supervised Learning, Spatio-temporal patterns

1 Introduction

The desire to better understand the remarkable information processing capabilities of the mammalian brain has recently led to the development of more complex and biologically plausible connectionist models, namely spiking neural networks (SNN). See *e.g.* [5] for a comprehensive text on the material. These models use trains of spikes as internal information representation rather than continuous-value variables. Many studies investigate how to use SNN for practical applications, some of them demonstrating very promising results in solving complex real world problems. However, due to the increased complexity of SNN compared to traditional artificial neural networks, the training of these networks in a supervised fashion has proved to be difficult. See [8] for a review on supervised learning methods for SNN.

Only few algorithms have been proposed that are able to impose a precise input-output mapping of spike trains to a SNN. One of the first supervised learning methods for SNN is SpikeProb [3]. It is a gradient descent approach that adjusts the synaptic weights in order to emit a single spike at specified

time. The timing of the output spike encodes a specific information, *e.g.* the class label of the presented input sample. Using SpikeProp, the SNN can not be trained to emit a desired train of spikes that has more than one spike.

An interesting learning rule for spatio-temporal pattern recognition has been suggested in [7]. The so-called Tempotron enables a neuron to learn whether to fire or not to fire in response to a specific input stimulus. Consequently, the method allows the processing of binary classification problems. However, the neuron is not intended to learn a precise target output spike train, but instead whether to spike or not to spike in response to an input stimulus.

A Hebbian based supervised learning algorithm called Remote Supervised Method (ReSuMe) was proposed in [11] and further studied in [12, 13]. ReSuMe, similar to spike time dependent plasticity (STDP) [1, 2], is based on a learning window concept. Using a teaching signal specific desired output is imposed to the output neuron. With this method, a neuron is able to produce a spike train precisely matching a desired spike train.

Recently, a method called Chronotron was proposed in [4]. It is based on minimizing the error between the desired spike pattern and the actual one. The error is measured using the Victor-Purpura spike distance metric [15]. This metric produces discontinuities in the error landscape that must be overcome through approximation. E-Learning is compared with ReSuMe on some temporal classification tasks and its better performance in terms of the number of spike patterns that can be classified is shown.

In this study, we propose a new learning rule allowing a neuron to learn efficiently associations of input-output spike trains. Although being principally simpler [1] compared to the methods mentioned above, we demonstrate its efficiency on a synthetic benchmark problem. The method is based on an extended Widrow-Hoff or Delta rule. In order to define a suitable error metric between desired and actual output spike trains, we convolve each spike sequence with a kernel function. Minimizing the error allows the efficient training of an output neuron to respond to specific input spikes with a desired target spike train.

We first present the new learning method. In the experiment section, we demonstrate the method using arbitrary input-output spike sequences for training an output neuron. Furthermore, we apply the learning method on a synthetic spatio-temporal classification problem and discuss the results.

2 Learning method

In this section, we describe the neural and synaptic model used in this study, followed by a detailed description of the proposed learning algorithm.

2.1 Neural and synaptic model

The Leaky Integrate-and-Fire (LIF) neuron is arguably the best known model for simulating spiking networks. It is based on the idea of an electrical circuit

[1] For example the method does not require the definition of learning windows.

containing a capacitor with capacitance C and a resistor with a resistance R, where both C and R are assumed to be constant. The dynamics of a neuron i are then described by the following differential equation:

$$\tau_m \frac{du_i}{dt} = -u_i(t) + R\ I_i^{\text{syn}}(t) \tag{1}$$

The constant $\tau_m = RC$ is called the membrane time constant of the neuron. Whenever the membrane potential u_i crosses a threshold ϑ from below, the neuron fires a spike and its potential is reset to a reset potential u_r. Following [5], we define

$$t_i^{(f)} : u_i(t^{(f)}) = \vartheta, f \in \{0, \ldots, n-1\} \tag{2}$$

as the firing times of neuron i where n is the number of spikes emitted by neuron i. It is noteworthy that the shape of the spike itself is not explicitly described in the traditional LIF model. Only the firing times are considered to be relevant.

The synaptic current I_i^{syn} of neuron i is modeled using an α-kernel:

$$I_i^{\text{syn}}(t) = \sum_j w_{ij} \sum_f \alpha(t - t_j^{(f)}) \tag{3}$$

where $w_{ij} \in \mathbb{R}$ is the synaptic weight describing the strength of the connection between neuron i and its pre-synaptic neuron j. The α-kernel itself is defined as

$$\alpha(t) = e\ \tau_s^{-1}\ t\ e^{-t/\tau_s} \Theta(t) \tag{4}$$

where $\Theta(t)$ refers to the Heaviside function and parameter τ_s is the synaptic time constant. We describe the parametrization of the model equations in the experiments section of this paper.

2.2 Learning

Similar to other supervised training algorithms, the synaptic weights of the network are adjusted iteratively in order to impose a desired input/output mapping to the SNN. We start with the common Widrow-Hoff rule for modifying the weight of a synapse i:

$$\Delta w_i^{\text{WH}} = \lambda x_i\ (y_d - y_{\text{out}}) \tag{5}$$

where $\lambda \in \mathbb{R}$ is a real-valued positive learning rate, x_i is the input transferred through synapse i, and y_d and y_{out} refer to the desired and the actual neural output, respectively. This rule was introduced for traditional neural networks with linear neurons. For these models, the input and output corresponds to real-valued vectors.

In SNN however, trains of spikes are passed between neurons and we have to define how to implement Equation 5 for this form of information exchange. In order to define the distance between spike trains, we introduce here an extended Delta rule. We transform each spike sequence into a continuous function using

a kernel function. This is similar to the binless distance metric used to compare spike trains [14]. In this study, we use an α-kernel, however many other kernels appear suitable in this context. The convolved input x_i is defined as

$$x_i(t) = \sum_f \alpha(t - t_i^{(f)}) \tag{6}$$

where $\alpha(t)$ refers to the α-function defined in Equation 4. Representing spikes as a continuous function allows us to define the difference between spike sequences as the difference between their representing functions. Similar to the neural input, we define the desired and actual outputs of a neuron:

$$y_d(t) = \sum_f \alpha(t - t_d^{(f)}) \tag{7}$$

$$y_{\text{out}}(t) = \sum_f \alpha(t - t_{\text{out}}^{(f)}) \tag{8}$$

As a consequence of the spike representation defined above, Δw_i^{WH} itself is a function over time. By integrating Δw_i^{WH} we obtain a scalar Δw_i that is used to update the weight of synapse i:

$$\Delta w_i = \lambda \int x_i(t)\ (y_d(t) - y_{\text{out}}(t))\ dt \tag{9}$$

Weights are updated in an iterative process. In each iteration (or epoch), all training samples are presented sequentially to the system. For each sample the Δw_i are computed and accumulated. After the presentation of all samples, the weights are updated to $w_i(e+1) = w_i(e) + \Delta w_i$, where e is the current epoch of the learning process.

We note that the algorithm is capable of training the weights of a single neural layer only. Related methods such as ReSuMe [11] and the Chronotron [4] exhibit similar restrictions. Therefore, a combination with the well-known Liquid State Machine (LSM) approach [9] was suggested in these studies. By transforming the input into a higher-dimensional space, the output of the LSM can potentially be mapped to any desired spike train.

Figure 1 illustrates the functioning of the learning method. An output neuron is connected to three input neurons through three excitatory synapses with randomly initialized weights. For the sake of simplicity, each input sequence consists of a single spike only. However, the learning method can also deal with more than one spike per input neuron. The inputs $t_i^{(f)}$ are visualized in Figure 1A. In this example, we intend to train the output neuron to emit a single spike at a pre-defined time $t_d^{(0)}$.

Assume that, as shown in Figure 1B, the presented stimulus causes the excitation of the output neuron resulting in the generation of two output spikes at times $t_{\text{out}}^{(0)}$ and $t_{\text{out}}^{(1)}$, respectively, neither of them equals the desired spike time $t_d^{(0)}$. The evolution of the membrane potential $u(t)$ measured in the output

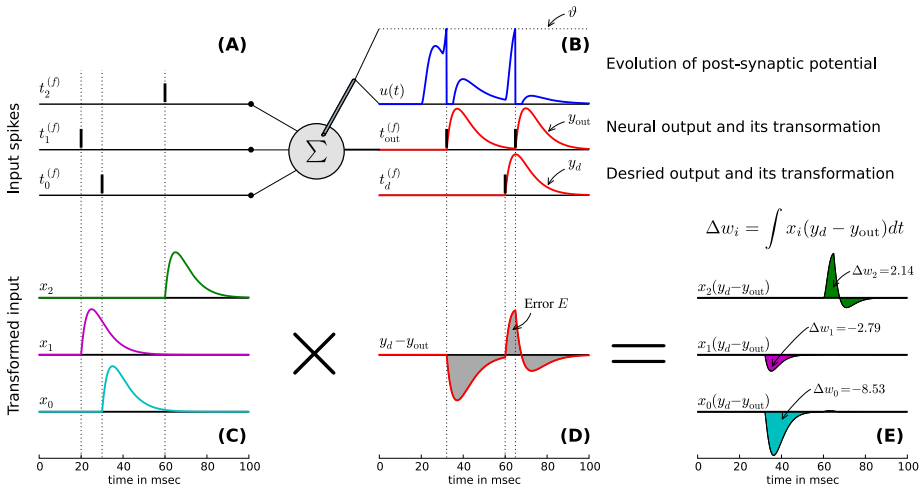


Fig. 1: Illustration of the proposed training algorithm. See text for a detailed explanation of the figure.

neuron is shown in middle top diagram of the figure above the actual and the desired spike trains, *cf.* Figure 1B.

The lower part in the figure (Figure 1C,D,E) depicts a graphical illustration of Equation 9. Using Equations 6 and 7, the input, actual and desired spikes trains are convolved with the α-kernel (Figure 1B and C). We define the area under the curve of the difference $y_d(t) - y_{\text{out}}(t)$ as an error between actual and desired output:

$$E = \int |y_d(t) - y_{\text{out}}(t)| \, dt \tag{10}$$

Although this error is not used in the computation of the weight updates Δw_i, this metric is an informative measure of the achieved training status of the output neuron. Figure 1E shows the weight updates Δw_i. We especially note the large decrease of weight w_0. The spike train $t_0^{(0)}$ of the first input neuron causes an undesired spike at $t_{\text{out}}^{(0)}$ and lowering the corresponding synaptic efficacy potentially suppresses this behavior. On the other hand, the synaptic weight w_2 is increased promoting the triggering of spike $t_{\text{out}}^{(1)}$ at an earlier time.

We note that, unlike related methods such as ReSuMe [11], the defined learning rule does not employ any learning windows making the method easy to comprehend and to implement.

3 Experimental results

Two experiments are conducted to demonstrate the functioning and the efficiency of the proposed learning algorithm. The first experiment intends to demonstrate the concept of the proposed learning algorithm, while the second

Neural Model	
Type	Leaky integrate-and-fire (LIF) neuron
Description	Dynamics of membrane potential $u(t)$: – Spike times: $t^{(f)} : u(t^{(f)}) = \vartheta$ – Sub-threshold dynamics: $\tau_m \frac{du}{dt} = -u(t) + R\ I^{\text{syn}}(t)$ – Reset & refractoriness: $u(t) = u_r \forall f : t \in (t^{(f)}, t^{(f)} + \tau_{\text{ref}})$ – exact integration with temporal resolution dt
Parameters	Membrane time constant $\tau_m = 10$ms Membrane resistance $R = 333.33\text{M}\Omega$ Spike threshold $\vartheta = 20$mV, reset potential $u_r = 0$mV Refractory period $\tau_{\text{ref}} = 3$ms Time resolution $dt = 0.1$ms, simulation time $T = 200$ms
Synaptic Model	
Type	Current synapses with $\alpha-$function shaped post-synaptic currents (PSCs)
Description	Synaptic input current $I^{\text{syn}}(t) = \sum w \sum_f \alpha(t - t^{(f)})$ $\alpha(t) = \begin{cases} e\ \tau_s^{-1}\ t\ e^{-t/\tau_s}, & \text{if } t > 0 \\ 0, & \text{otherwise} \end{cases}$
Parameters	Synaptic weight $w \in \mathbb{R}$, uniformly randomly initialized in $[0, 25]$ Synaptic time constant $\tau_s = 5$ms
Input Model	
Type	Random input
Details	Population of 200 input neurons each firing a single spike at a randomly chosen time in the period $(0, T)$

Table 1: Tabular description of the experimental setup as suggested in [10].

experiment implements the more practical scenario of a spatio-temporal pattern classification problem.

We follow the initiative recently proposed in [10] for promoting reproducible descriptions of neural network models and experiments. The initiative suggests the use of specifically formatted tables explaining neural and synaptic models along with their parametrization, *cf.* Table 1. For both experiments, we use 200 input neurons that stimulate the synapses of a single output neuron, *cf.* Figure 2. The spike trains for each input neuron are sampled from a uniform random distribution in the interval $[0, 200]$ms. For the sake of simplicity of this demonstration, we allow only a single spike for each input neuron (multiple spikes in a spike train are processed in the same way). A similar experimental setup was used in [4]. The single output neuron is fully connected to all 200 input neurons with randomly initialized connection weights. All of our experiments employ the SNN simulator NEST [6].

3.1 Demonstration of the learning rule

The purpose of the first experiment is to demonstrate the concept of the proposed learning method. The task is to learn a mapping from a random input spike

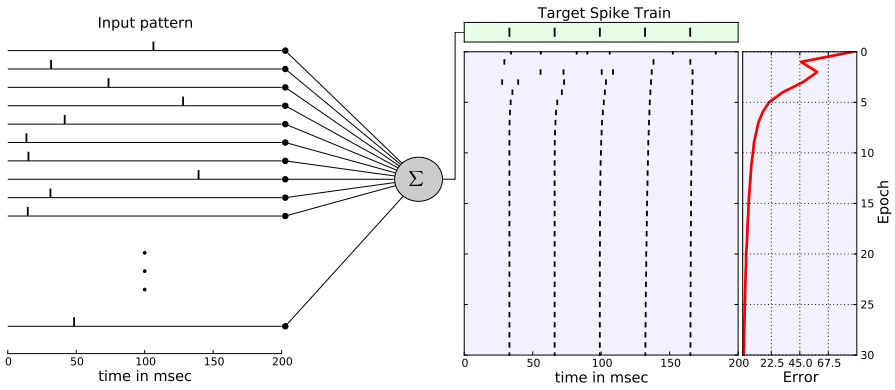


Fig. 2: A single output neuron is trained to respond with a temporally precise output spike train to a specific spatio-temporal input. The organization of the figure is inspired by [4].

pattern to specific target output spike train. This target consists of five spikes occurring at the times $t_d^{(0)} = 33$, $t_d^{(1)} = 66$, $t_d^{(2)} = 99$, $t_d^{(3)} = 132$ and $t_d^{(4)} = 165$ms. Initially, the synaptic weights are randomly generated uniformly in the range $(0, 25\text{pA})$. In 100 epochs, we apply the new learning rule to allow the output neuron to adjust its connection weights in order to produce the desired output spike train. The experiment is repeated for 100 runs each of them initialized with different random weights in order to guarantee statistical significance.

In Figure 2, the experimental setup of a typical run is illustrated. The left side of the diagram shows the network architecture as defined in the experimental setup above. The right side shows the desired target spike train (top) along with the produced spike trains by the output neuron over a number of learning epochs (bottom). We note that the output spike trains in early epochs are very different from the desired target spike sequence. In later epochs the output spike converges towards the desired sequence. Consequently, the error as defined in Equation 10 decreases in succeeding epochs (right part of Figure 2). We note that the neuron is able to reproduce the desired spike output pattern very precisely in less than 30 learning epochs.

Figure 3 shows the evolution of the average error over the performed 100 runs. We note the logarithmic scale of the y-axis and the exponential decrease of the error. The slope of the curve suggests further improvement of the error, if the neuron was trained longer than the 100 performed learning epochs.

From this simple experiment, we conclude that the proposed learning method is indeed able to train a input-output behavior to a spiking neuron.

3.2 Classification of spatio-temporal data

The second experiment is a spatio-temporal classification task. The objective is to learn to classify five classes of input spike patterns. The pattern for each class

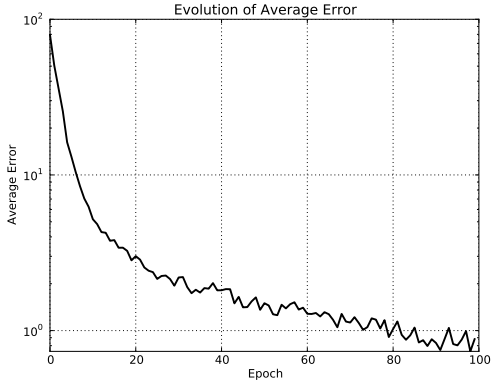
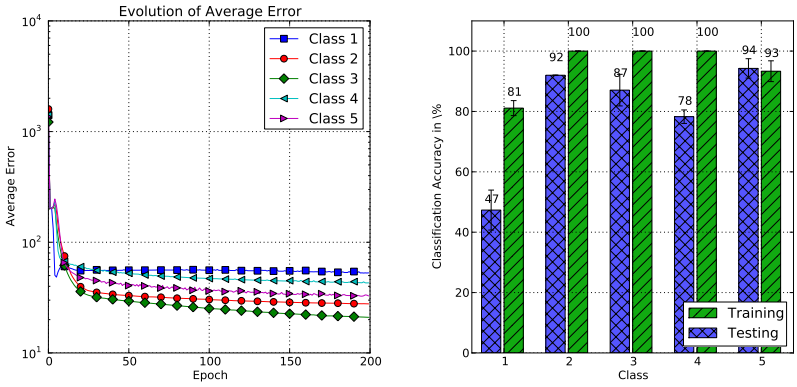


Fig. 3: Evolution of average error obtained in 100 independent runs

is given as a random input spike pattern that was created in a similar fashion as for the first experiment. Fifteen copies for each of the five pattern are then generated by perturbing each pattern using a Gaussian jitter with a standard deviation of 3ms resulting in a total of $15 \times 5 = 75$ samples in the training data set. Additionally, we create $25 \times 5 = 125$ testing samples using the same procedure. The output neuron is then trained to emit a single spike at a specific time for each class. Only the training set is used during training, while the testing set is used to determine the generalization ability of the trained neuron. The spike time of the output neuron encodes the class label of the presented input pattern. The neuron is trained to spike at the time instances 33, 66, 99, 132, and 165ms respectively, each spike time corresponding to one of the five class labels. We allow 200 epochs for the learning method and we repeat the experiment in 30 independent runs. For each run we chose a different set of random initial weights.

Figure 4a shows the evolution of the average error for each of the five classes. In the first few epochs, the value of the error oscillates and then starts to stabilize and decrease slowly. The learning error decreases for some classes faster than for others, *e.g.* class 3. We also note that the class reporting the highest error is class 1. This behavior is expected and confirms a quite similar finding in [4]. In order to classify samples of class 1 correctly, the output neuron has to emit a very early spike at $t \approx 33$ms. Consequently, the neuron needs to be stimulated by input spikes occurring at times before $t = 33$ms. However, due to the random generation of the input data, only few input spikes occur before $t = 33$ms. Most input spikes arrive after that time at the output neuron and therefore do not contribute to the correct classification of class 1 samples. The relationship between the accuracy and the output spike time was also noted in [4]. Future studies will further investigate this interesting observation.

In order to report the classification accuracy of the trained neuron, we define a simple error metric. We consider a pattern as correctly classified, if the neuron



(a) (b)

Fig. 4: Evolution of the average errors obtained in 30 independent trails for each class of the training samples (**a**). The average accuracies obtained in the training and testing phase (**b**).

fires a single spike within $[t_d^{(f)} - 3, t_d^{(f)} + 3]$ms of the desired spike time $t_d^{(f)}$. Any other output is considered as incorrect. It is noteworthy to mention that using this definition, an untrained neuron is very likely to produce incorrect outputs resulting in accuracies close to zero. Figure 4b shows the average classification error for each class in the training and testing phase. As mentioned above, for testing, the 125 unseen patterns of the test set are used. The neuron is able to learn to classify the 75 training patterns with an average accuracy of 94.8% across all classes. Once more, we note the comparatively poor classification performance of samples belonging to the first class. For the test patterns, the neuron is able to achieve average accuracy of 79.6% across all classes which demonstrates a satisfying generalization ability.

4 Conclusion

In this paper, we have proposed a novel learning algorithm that allows the training of a precise input-output behavior to a SNN. Besides the benefits of an efficient supervised training method for a spiking neuron, we see the main advantage of the proposed approach in its algorithmic simplicity promoting its straightforward application to engineering problems. As a proof of concept, we have demonstrated the suitability of the method for classification problems on a small synthetic data set. Although the method can only train a single layer of the SNN, it was argued that in combination with the LSM approach [9], the method allows the processing of complex non-linearly separable classification problems. Considering the difficulty of the spike-based classification task, the results are both satisfying and encouraging.

Acknowledgements

The work on this paper has been supported by the Knowledge Engineering and Discovery Research Institute (KEDRI, www.kedri.info). One of the authors, NK, has been supported by a Marie Curie International Incoming Fellowship with the 7^{th} European Framework Programme under the project 'EvoSpike', hosted by the Neuromorphic Cognitive Systems Group of the Institute for Neuroinformatics of the ETH and the University of Zurich.

References

1. Bell, C.C., Han, V.Z., Sugawara, Y., Grant, K.: Synaptic plasticity in a cerebellum-like structure depends on temporal order. Nature 387, 278–281 (May 1997)
2. Bi, G.q., Poo, M.m.: Synaptic modifications in cultured hippocampal neurons: Dependence on spike timing, synaptic strength, and postsynaptic cell type. J. Neurosci. 18(24), 10464–10472 (1998)
3. Bohte, S.M., Kok, J.N., Poutré, J.A.L.: SpikeProp: backpropagation for networks of spiking neurons. In: ESANN. pp. 419–424 (2000)
4. Florian, R.V.: The chronotron: a neuron that learns to fire temporally-precise spike patterns. http://precedings.nature.com/documents/5190/version/1 (Nov 2010)
5. Gerstner, W., Kistler, W.M.: Spiking Neuron Models: Single Neurons, Populations, Plasticity. Cambridge University Press, Cambridge, MA (2002)
6. Gewaltig, M.O., Diesmann, M.: Nest (neural simulation tool). Scholarpedia 2(4), 1430 (2007)
7. Gutig, R., Sompolinsky, H.: The tempotron: a neuron that learns spike timing-based decisions. Nat Neurosci 9(3), 420–428 (Mar 2006)
8. Kasiński, A.J., Ponulak, F.: Comparison of supervised learning methods for spike time coding in spiking neural networks. Int. J.of Applied Mathematics and Computer Science 16, 101–113 (2006)
9. Maass, W., Natschläger, T., Markram, H.: Real-time computing without stable states: A new framework for neural computation based on perturbations. Neural Computation 14(11), 2531–2560 (2002)
10. Nordlie, E., Gewaltig, M.O., Plesser, H.E.: Towards reproducible descriptions of neuronal network models. PLoS Comput Biol 5(8), e1000456 (08 2009)
11. Ponulak, F.: ReSuMe – new supervised learning method for spiking neural networks. Tech. rep., Institute of Control and Information Engineering, Poznań University of Technology, Poznań, Poland (2005)
12. Ponulak, F.: Analysis of the resume learning process for spiking neural networks. Applied Mathematics and Computer Science 18(2), 117–127 (2008)
13. Ponulak, F., Kasiński, A.: Supervised learning in spiking neural networks with ReSuMe: sequence learning, classification, and spike shifting. Neural Computation 22(2), 467–510 (Feb 2010), PMID: 19842989
14. van Rossum, M.C.: A novel spike distance. Neural Computation 13(4), 751–763 (Apr 2001)
15. Victor, J.D., Purpura, K.P.: Metric-space analysis of spike trains: theory, algorithms and application. Network: Computation in Neural Systems 8(2), 127–164 (1997)